Solutions to the small problems after each section

This document should be used only in connection with the textbook "Java the UML Way", written by Else Lervik and Vegard B. Havdal, John Wiley & Sons, Ltd. 2002.

Chapter 2.2

Problem 2

```
class Prob2_2_2 {
  public static void main(String[] args) {
    double length = 5.0;
    double height = 2.3;
    double area = length * height;
    System.out.println("The area of the wall is " + area + " square meters.");
    double perimeter = 2 * (length + height);
    System.out.println("The perimeter of the wall is " + perimeter + " meters");
  }
}
```

Chapter 2.4

Problem 1

The following words are valid Java names:

```
TheBookOfMary      The_bike_of_Eve
NUMBER             _Number34
```

Problem 2

Keywords: class, public, static, void, final, double

Variables: factor, numberOfCalories, numberOfKJoules.

Chapter 2.5

Problem 1

```
final int numberOfYears = 35
```

Error: "';' expected." Semicolon is missing at the end of the line.

```
double Amount = 40,95;
```

Error: "<identifier> expected." Java doesn't understand us. A guess is that an identifier should be placed after the comma. This is an example where the error is a bit far from what the message tells us. Unfortunately, this situation is not that seldom. In this case: Full stop, not comma, should be used as a decimal point.

```
final aConstant = 789.6;
```

Error: "Identifier expected." The same message, but this time it has a meaning: What is missing is an identifier; here it is the name of a data type. The statement should be as follows:

```
final double aConstant = 789.6;
```

Next statement:

```
DOUBLE number = 15.7;
```

Error: "cannot resolve symbol, symbol: class DOUBLE". Java doesn't understand the word "DOUBLE". It is not a primitive data type, and we have not declared this name. We suppose that lowercase double is meant:

```
double number = 15.7;
```

Next statement:

```
Amount = Amont * 100;
```

Error: "cannot resolve symbol, symbol: variable Amont ". The same message as before, but this time Java doesn't understand the word "Amont". It is not declared. This is probably a typo. We meant to write "Amount".

Here you have all the statements without errors:

```
final int numberOfYears = 35;
double Amount = 40.95;
final double aConstant = 789.6;
double number = 15.7;
Amount = Amount * 100;
```

Problem 2

Amount: Variable names should not start with a capital letter.

Chapter 2.6

Problem 1

12.45	double
"Hello!"	String
"\""	String
'\"'	char
12345L	long
0456	int (in the 8-digit system)
true	boolean
3.245e-5	double

Problem 2

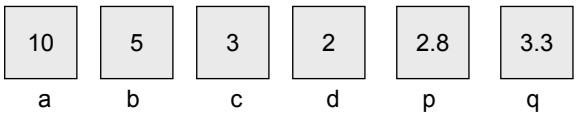
a) the price - double
b) quantity measured in kilograms - double
c) quantity measured in number - int
d) color code (a letter) - char
e) bar code - String
f) the name - String

Problem 3

```
Ann Isabelle Adams
The number is 0.023
The symbol is a
```

Chapter 2.7

Problem 1



Problem 2

Only two statements are valid Java statements:

```
b = b + c;
d = d;
```

Problem 3

```
c + d * a,  value 23
a * b / c + a, value 26
c % d, value 1
d % c, value 2
p % q, value 2.8
```

q % p, value 0.5
c % d % a + b / c, value 2
a = b = 16, value 16

Problem 4

b * b - 4 * a * c
x * x + y * y + z * z
x * x * x
(a - b) * (a + b)
(a + b) / (c + d)

Note! Java doesn't have an operator for raising a number to a power.

Chapter 2.8

Problem 1

p + (double) a / q, value 5.83
p + a / q, value 5.83
(int) p + (int) q, value 5
(int) (p + q), value 6

Problem 2

The right side of the assignment operator in the statement a = p + q; should be cast, because the data type of the variable of the left side is "smaller" than that of the right side. The casting gives: a = (int) (p + q);

The variables a, b and d have the following contents after the four statements are executed: a = 10, b = 0, d = 0.

Problem 3

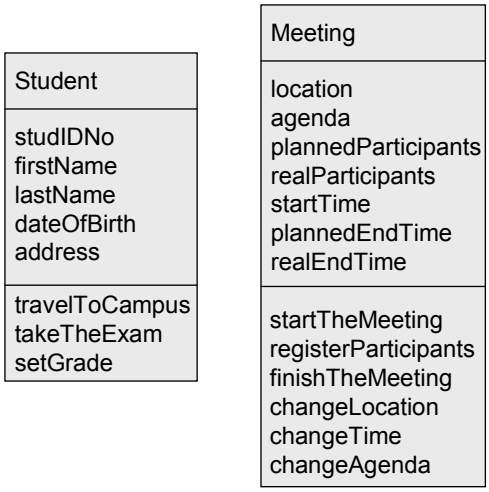
```
a = -129542144, b = 60000000000
```

The value of a is not correct. The reason for this is that the right side of the assignment operator is calculated as an int expression. That calculation gives a wrong result, because it brings us outside the range of the int data type.

The value of b is correct. This is because the right side now is calculated as a long expression.

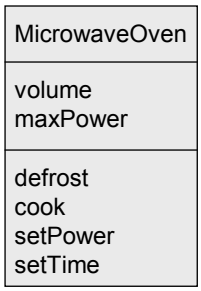
Chapter 3.1

Problem 1



Problem 2

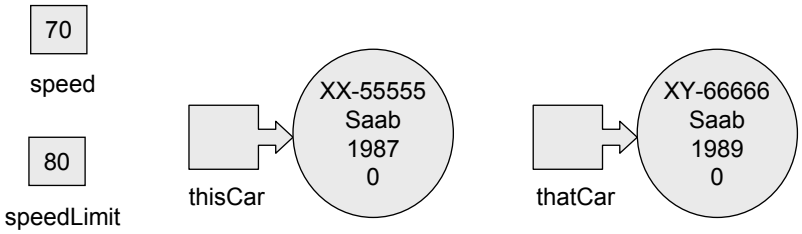
The owen has volume and maximum power. Common functions are defrosting and cooking. Power and time may be set.



States: The owen is not in use, it is cooking food on half power, it is defrosting.

Chapter 3.2

Problem 1



speed and speedLimit are variables of primitive data types. The contents in the variable speedLimit are constant. thisCar and thatCar are references to instances of the Car class. The objects are shown as circles in the figure.

Problem 2

```
Car theCarOfJohn = new Car(1234, "Mazda", 1990, 0); // the name carOfJohn has to be
                                                    //declared, we have to tell that it is a Car object
Car myCar = new Car("ZD-44444", "Peugeot", 1995, 0); // after new a classname should
                    // be found, and two arguments are missing in the constructor call
theCarOfJohn.start(); // no arguments
theCarOfJohn.speedUp(80); // no errors here
theCarOfJohn.slowDown(30); // one argument here
theCarOfJohn.stop(); // object name and not class name here
```

Problem 3

```
Car myCar = new Car(“AB-12345”, “Opel”, 1998, 0);
myCar.start();
myCar.speedUp(20);
myCar.speedUp(50);
myCar.slowDown(30);
myCar.stop();
```

Problem 4

```
Student peter = new Student(“56789”, “Peter Smith”);
peter.takeExam(“LO-189D”);
peter.takeExam(“LO-190D”);
peter.takeExam(“LO-191D”);
peter.setGrade(2.0, “LO-189D”);
peter.setGrade(1.8, “LO-190D”);
peter.setGrade(2.3, “LO-191D”);
peter.applyForStudentLoan();
```

Chapter 3.3

Problem 1

```
import java.util.Random;
class Prob3_3_1 {
```

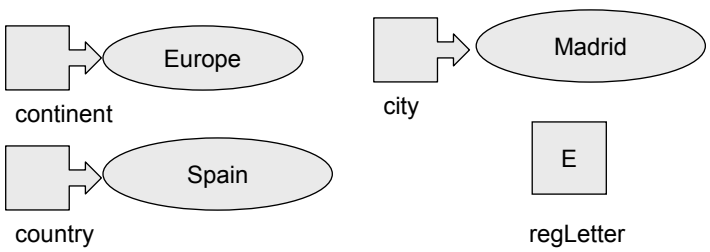
```
  public static void main(String[] args) {
    Random randomGen = new Random();

    /* Problem a) */
    long number1 = randomGen.nextLong();
    long number2 = randomGen.nextLong();
    long number3 = randomGen.nextLong();
    System.out.println("Three random numbers, long: " +
              number1 + ", " + number2 + ", " + number3);

    /* Problem b) */
    int numberI1 = randomGen.nextInt(1001);
    int numberI2 = randomGen.nextInt(1001);
    int numberI3 = randomGen.nextInt(1001);
    System.out.println("Three random number, int [0..1000]: " +
               numberI1 + ", " + numberI2 + ", " + numberI3);

    /* Problem c) */
    double num1 = randomGen.nextDouble();
    double num2 = randomGen.nextDouble();
    System.out.println("Two random numbers, decimal [0.0..1.0>: " + num1 + ", " + num2);

    /* Problem d) */
    int numA = (randomGen.nextInt(1001) - 500);
    int numB = (randomGen.nextInt(1001) - 500);
    System.out.println("Two random numbers, integer [-500..500]: " + numA + ", " + numB);

    /* Problem e) */
    double numC = 100.0 * randomGen.nextDouble();
    double numD = 100.0 * randomGen.nextDouble();
    System.out.println(
          "Two random numbers, decimal [0.0..100.0>: " + numC + ", " + numD);
  }
}

/* Example Run:
Three random numbers, long: 1867604404187107707, 6292911881391410402,
 -9162625934092467911
Three random number, int [0..1000]: 274, 83, 454
Two random numbers, decimal [0.0..1.0>: 0.5246851081768323, 0.12566507647809322
Two random numbers, integer [-500..500]: 379, -332
Two random numbers, decimal [0.0..100.0>: 15.140725040643554, 15.095543360880848
*/
```

Problem 2

```
import java.util.Random;
class Prob3_3_2 {
  public static void main(String[] args) {
    Random randomGen = new Random();
     for (int i = 0; i < 20; i++) System.out.println("A random number: " + randomGen.nextInt());
  }
}
```

Chapter 3.4

Problem 1



continent, country, and city are references to objects which are instances of the String class. regLetter is a variable of the char data type. char is a primtive data type, while String is a reference type.

Problem 2

```
yesterdayy9
```

Problem 3

```
String text = "Yellow Submarine";  // not single, but double quotes
String vocalist = new String("Ringo Starr"); // always a class name after new
int noOfChars = vocalist.length(); // parenthesis after length; it's a method
char lastLetter =  // the method charAt() returns a char, not a String
        vocalist.charAt(noOfChars - 1); // the first position in the string has no. 0
System.out.println("The length of the vocalist's name is: " +
   noOfChars + " The last letter is " + lastLetter);  // no errors in this statement
vocalist = vocalist.toUpperCase(); /* To get the result of the conversion from lower to upper
   case, we have to use the return value from the method. This is actually a new String
   object, which we set our "old" reference to refer to. */
System.out.println("The vocalist's name in upper case: " + vocalist); // no errors here
```

Problem 4

```
28
2
5
This is one of many problems.
This is one of many problems.
This is ona of many problams.
```

Problem 5

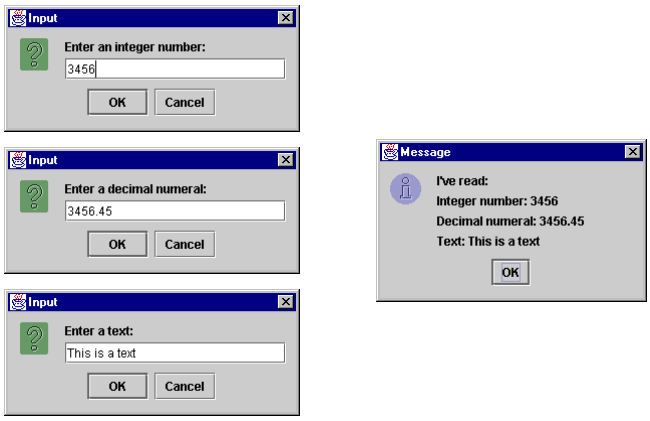
```
String text = "Today it's June 16th. In two weeks the summer holiday is here.";
int pos1 = text.indexOf('.');
int pos2 = text.indexOf('.', pos1 + 1);
int pos3 = text.indexOf("16");
int pos4 = text.indexOf("In");
int pos5 = text.indexOf("In", pos4 + 1);
System.out.println(pos1 + " " + pos2 + " " + pos3 + " " + pos4 + " " + pos5);
```

The output is: 20 61 16 22 -1

The last value (-1) shows that the second "In" is not in the text.

Chapter 3.7

Problem 1



The figure shows the simplest input dialog.

Here is the whole program:

```
import javax.swing.JOptionPane;
class Prob3_7_1 {
  public static void main(String[] args) {
    /* The simplest showInputDialog() method */
    String integerRead = JOptionPane.showInputDialog("Enter an integer number: ");
    int integerNumber = Integer.parseInt(integerRead);
    String decimalRead = JOptionPane.showInputDialog("Enter a decimal numeral: ");
```

```
    double decimalNumeral = Double.parseDouble(decimalRead);
    String text = JOptionPane.showInputDialog("Enter a text: ");
    JOptionPane.showMessageDialog(null, "I've read:\n" +
                          "Integer number: " + integerNumber + "\n" +
                          "Decimal numeral: " + decimalNumeral + "\n" +
                          "Text: " + text);

    /* The most complicated showInputDialog() method */
    integerRead = JOptionPane.showInputDialog(null, "Enter an integer number: ",
       "showInputDialog() with four parameters", JOptionPane.PLAIN_MESSAGE);
    integerNumber = Integer.parseInt(integerRead);
    decimalRead = JOptionPane.showInputDialog(null, "Enter a decimal numeral: ",
       "showInputDialog() with four parameters", JOptionPane.PLAIN_MESSAGE);
    decimalNumeral = Double.parseDouble(decimalRead);
    text = JOptionPane.showInputDialog(null, "Enter a text: ",
       "showInputDialog() with four parameters", JOptionPane.PLAIN_MESSAGE);
    JOptionPane.showMessageDialog(null, "I've read:\n" +
                          "Integer number: " + integerNumber + "\n" +
                          "Decimal numeral: " + decimalNumeral + "\n" +
                          "Text: " + text);
    System.exit(0);
  }
}
```

Problem 2

Here is the revised program:

```
import javax.swing.JOptionPane;
class Prob3_7_2 {
  public static void main(String[] args) {
    final double factor = 4.2;
    String numberOfCaloriesRead =
                          JOptionPane.showInputDialog("The number of calories: ");
    double numberOfCalories = Double.parseDouble(numberOfCaloriesRead);
    double numberOfKJoules = numberOfCalories * factor;
    JOptionPane.showMessageDialog(null,
                                  "This is the same as " + numberOfKJoules + " kJ");
    System.exit(0);
  }
}
```

Chapter 4.2

Problem 1

Step 1 in the FloorCalculations has to be changed:

```
/* Step 1: We instantiate an object of the class. */
String myLengthRead = JOptionPane.showInputDialog("The length of the floor: ");
double myLength = Double.parseDouble(myLengthRead);
String myWidthRead = JOptionPane.showInputDialog("The width of the floor: ");
double myWidth = Double.parseDouble(myWidthRead);
Surface aFloor = new Surface("Mary's floor", myLength, myWidth);
```

Problem 2

a) Edward Johnson is a particular person, and should therefore be modelled as an object, not as a class. A class is a description of several objects.

b) The statement is a method head. Inside the parenthesis, there should have been parameters. A parameter is a description of an argument. "January" is not a description, it is an example of an argument. String month would have been an appropriate parameter.

c) The same as in b). 1756 is an argument. An example of a parameter is int year.

d) The concepts are mixed. What we can say is, for example: "The person object has name as an attribute."

e) This is a method head. But void or a data type before the method name is missing. The head might, for example, be like this:

```
double getSalary(String month)
```

f) This statement has no meaning. The parameters are in the head of a method. The arguments are the values we send with the method call when we use it to send a message to an object. We use the parameters and arguments in the same way for constructors. An example:

The class:

```
class Person {
  public Person(String initName) {   // initName is a parameter
    .....
  public double getSalary(String month) {       // month is a parameter
    .....
}
```

From the client program:

```
....
person thePerson = new Person("Peter");  //  "Peter" is an argument
.....
double theSalary = thePerson.getSalary("January");  // "January" is an argument
```

g) An attribute has neither head nor body. A method (and a class and a constructor) has head and body. The properties of an object are called the object's attributes. Example: A person has the following attributes: first name, last name, date of birth, height, weight.

h) A local variable can only be used inside the block where it is declared, as distinct from an instance variable which may be accessed from inside of all the methods in the class where it is declared. (And in special situations from outside the class, too. But more about that later in this chapter.)

Problem 3

```
class Circle {
  private double radius;  // data type was missing
  public Circle(double initRadius) { // the C was a lower case letter
    radius = initRadius;
  }
  public double getArea() { // the area should be double, not int
    return Math.PI * radius * radius; // Math.PI is a pre-defined constant that is used this way
  }
  public double getCircumference() {
    double circumference = 2.0 * Math.PI * radius; // circumference was not declared
    return circumference;
  }
}
```

Problem 4

```
class CircleCalculations {
  public static void main(String[] args) {
    Circle aCircle = new Circle(20);
    double area = aCircle.getArea();
    System.out.println("The area is calculated to be " + area);
    double circumference = aCircle.getCircumference();
    System.out.println("The circumference is calculated to be " + circumference);
  }
}
```

Problem 5

There are *local variables* only in the main() method: aFloor, name, width, length, area, circumference

We find the *instance variables* in the beginning of the class declaration: name, length, width. Despite these variables have the same names as some of the local variables, they do not occupy the same location in the computer's internal memory.

We search for *parameters* in the heads of the constructors and the methods: The constructor head Surface() has three parameters: initName, initLength, initWidth. None of the instance methods have parameters. But the main() method has one parameter: args. You will learn how to use this parameter in problem 5, chapter 10.1.

Chapter 4.4

Problem 1

If we do not program any constructor, a constructor with empty parameter list will be made. This is the default constructor. We use it in this way:

```
Surface aWall = new Surface();
```

The instance variables name, width and length will now have the values they've got in the declaration. If no specific values are set up there, they will get the values, null, 0.0 and 0.0, respectively.

We may choose to make set methods for these values, example:

```
public void setWidth(double newWidth) {
  width = newWidth;
}
```

These set methods may then be used to give the instance variables other values than 0:

```
aWall.setWidth(5);
```

Problem 2

The parameter and the instance variable have the same name. We try to handle this by using the this reference. But the keyword this is on the wrong side of the assignment operator. If the parameter and the instance variable have the same name, the method should look like this:

```
public void setWidth(double width) {
  this.width = width;
}
```

Problem 3a

```
public int getMerchNo() {
  return merchandiseNo;
}
```

In the client program:

```
System.out.println("The merch. no. is: " + aMerchandise.getMerchNo())
```

Problem 3b

```
public String getName() {
  return merchandiseName;
}
```

In the client program:

```
System.out.println("The merch. name is: " + aMerchandise.getName());
```

Problem 3c

The values set by the constructor are the significant values.

Problem 3d

```
class Merchandise {
  private static final double VAT = 20.0;
  private static final double VATfactor = 1.0 + VAT / 100.0;

  private int inStock;
  .....
  public Merchandise(String initMerchandiseName, int initMerchandiseNo, double initPrice,
          int initInStock) {
    merchandiseName = initMerchandiseName;
    merchandiseNo = initMerchandiseNo;
    price = initPrice;
    inStock = initInStock;
  }

  public int getInStock() {
    return inStock;
  }

  public void increaseInStock(int number) {
    inStock = inStock + number;
  }

  public void decreaseInStock(int number) {
    inStock = inStock - number;
```

```
}
....
```

In the client program:

```
Merchandise aMerchandise = new Merchandise("Brie", 123, kiloPrice1, 50);
aMerchandise.increaseInStock(20);
aMerchandise.decreaseInStock(15);
System.out.println("In stock: " + aMerchandise.getInStock());
```

Problem 4a

Here is the client program:

```
public static void main(String[] args) {
  Room roomA = new Room("A", 20);
  Room roomB = new Room("B", 100);
  Room roomC = new Room("C", 50);

  System.out.println("The number of room A is " + roomA.getRoomNo());
  System.out.println("The number of room B is " + roomB.getRoomNo());
  System.out.println("The number of room C is " + roomC.getRoomNo());
}
```

The output from this program:

```
The number of room A is 1
The number of room B is 2
The number of room C is 3
```

Problem 4b

Here is the method:

```
public static int getLastUsedRoomNo() {
  return lastUsedRoomNo;
}
```

It is invoked in this way:

```
int lastNo = Room.getLastUsedRoomNo();
```

Problem 4c

No. An instance method may use instance variables, and such variables do not have any meaning without an object. A class method is not allowed to use instance variables.

Chapter 4.5

Problem 1

```
class Prob4_5_1 {
  public static void main(String[] args) {
    YesNoCounter counter = new YesNoCounter();
    counter.increaseNumberOfYes();
    counter.increaseNumberOfNo();
    System.out.println("Number of yes votes: " + counter.getNumberOfYes() +
      " Number of no votes: " + counter.getNumberOfNo());
    counter.increaseNumberOfYes(10);
    counter.increaseNumberOfNo(20);
    System.out.println("Number of yes votes: " + counter.getNumberOfYes() +
      " Number of no votes: " + counter.getNumberOfNo());
  }
}
```

Problem 2

```
2
2
```

Problem 3

a = 121
b = 20
c = 1

Chapter 4.7

Problem 2

```
class Drawing extends JPanel {
  public void paintComponent(Graphics window) {
    super.paintComponent(window);
    setBackground(Color.green);
    window.setColor(Color.blue);
    window.drawString("Hello hello", 50, 100);
    window.setColor(Color.red);
    window.fillOval(40, 30, 55, 40);
  }
}
```

Chapter 5.1

Problem 1

```
class Prob5_1_1 {
  public static void main(String[] args) {
    double number1 = 1;
    double number2 = 0;
    Calculator calc = new Calculator(number1, number2);
    System.out.println("Tests with the numbers " + number1 + " and " + number2);
    System.out.println("Sum: " + calc.calculateSum());
    System.out.println("Difference: " + calc.calculateDifference());
    System.out.println("Product: " + calc.calculateProduct());
    System.out.println("Quotient: " + calc.calculateQuotient());
    calc.setNumbers(5, 10);
    System.out.println("The new numbers " + calc.getNumber1() +
      " and " + calc.getNumber2());
    System.out.println("Sum: " + calc.calculateSum());
    System.out.println("Difference: " + calc.calculateDifference());
    System.out.println("Product: " + calc.calculateProduct());
    System.out.println("Quotient: " + calc.calculateQuotient());
          calc.setNumbers(0, 0);
          System.out.println("The new numbers " + calc.getNumber1() +
            " and " + calc.getNumber2());
          System.out.println("Sum: " + calc.calculateSum());
          System.out.println("Difference: " + calc.calculateDifference());
          System.out.println("Product: " + calc.calculateProduct());
          System.out.println("Quotient: " + calc.calculateQuotient());
  }
}

/* Example Run:
Tests with the numbers 1.0 and 0.0
Sum: 1.0
Difference: 1.0
Product: 0.0
Quotient: Infinity
The new numbers 5.0 and 10.0
Sum: 15.0
Difference: -5.0
Product: 50.0
Quotient: 0.5
The new numbers 0.0 and 0.0
Sum: 0.0
Difference: 0.0
Product: 0.0
Quotient: NaN
*/
```

Problem 2

The method:

```
public double solve1Degree() {
  return -number2 / number1;
}
```

A method call in a client program:

```
System.out.println(
  "The equation of 1.degree has the following solution " + calc.solve1Degree());
```

Problem 3

We need one more instance variable:

```
private double number3;
```

A constructor with three parameters:

```
public Calculator(double initNumber1, double initNumber2, double initNumber3) {
   number1 = initNumber1;
   number2 = initNumber2;
   number3 = initNumber3;
 }
```

Two methods to calculate the roots:

```
double findRoot1() {
  double discriminant = number2 * number2 - 4 * number1 * number3;
  return (-number2 + Math.sqrt(discriminant)) / (2 * number1);
}

double findRoot2() {
  double discriminant = number2 * number2 - 4 * number1 * number3;
  return (-number2 - Math.sqrt(discriminant)) / (2 * number1);
}
```

In the client program:

```
Calculator calc = new Calculator(number1, number2, number3);
System.out.println("The roots in the 2.degree equation are: " +
                          calc.findRoot1() + " and " + calc.findRoot2());
```

Chapter 5.2

Problem 1

```
int answer = JOptionPane.showConfirmDialog(null, "Multiply the numbers? ",
            "Calculator", JOptionPane.YES_NO_OPTION);

/* Calculating results */
Calculator calcus = new Calculator(number1, number2);
double calculatedAnswer;
char operator;
if (answer == JOptionPane.YES_OPTION) { // Yes is pressed
  calculatedAnswer = calcus.calculateProduct();
  operator = '*';
} else { // No or Esc is pressed, or the dialog is closed
  calculatedAnswer = calcus.calculateQuotient();
  operator = '/';
}
```

Chapter 5.3

Problem 1

sum is equal to 10.

Problem 2a

Five variables are declared:

In block 1: number1 and number2
In block 2: number3 and number4
In block 3: number3

Two of the variables have the same name.

Problem 2b

The declarations in block 1:

The scope of number1 is line 2-17.
The scope of number2 is line 3-17.

The declarations in block 2:

The scope of number3 is line 6-12.
The scope of number4 is line 9-12.

The declaration in block 3:

The scope of number3 is line 13-15.

Problem 2c

```
number1 = 60, number2 = 50
number1 = 30, number2 = 100
number3 = 20, number4 = 150
number1 = 30, number2 = 100
```

Problem 2d

```
number1 = 60, number2 = 50
number3 = 65
number1 = 60, number2 = 50
```

Chapter 5.4

Problem 1a

```
if (number > 20) code = 'M';
else code = 'F';
```

Problem 1b

```
if (weight / (height * height) > 25) System.out.println("You weigh too much");
```

Problem 2

From the first if statement, the recommended code style looks like this:

```
if (a < b) a = b;
b = 10;
if (p == 20) q = 13;
else q = 17;
if (r > s) {
  q = 100;
}
s = 200;
```

The values of the variables after run are:

a = 30
b = 10
p = 20
q = 100
r = 30
s = 200

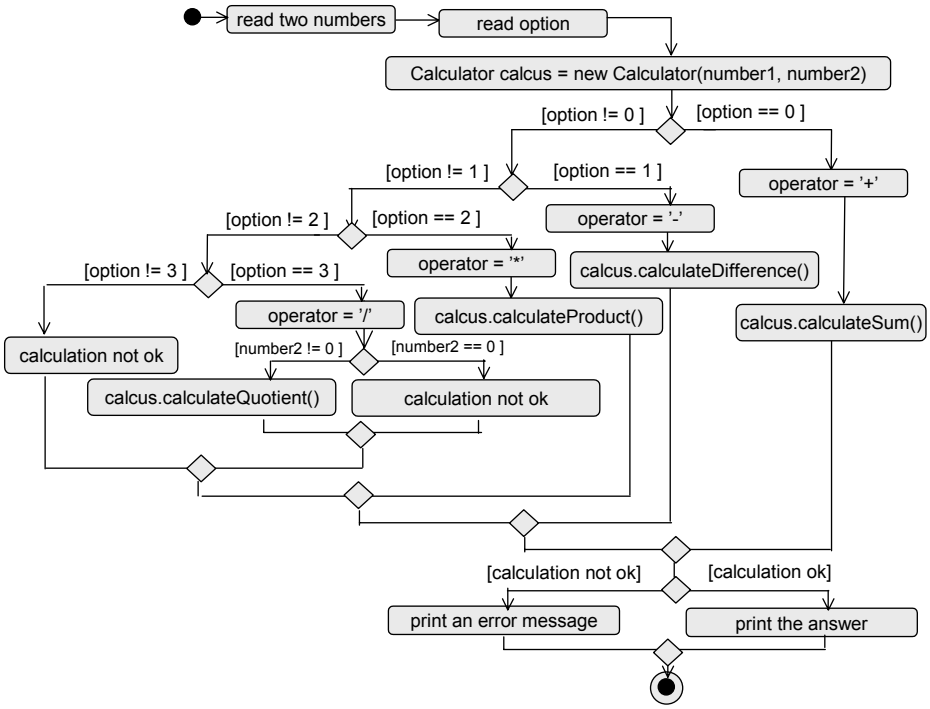
Problem 3

If size is greater than 38 nothing happens. If not, the text "Small!" is written to the console. The code segment should be rewritten as follows:

```
if (size <= 38) System.out.println("Small!");
```

Chapter 5.5

Problem 1



Problem 2

```
int a = 20;
int b = 30;
int c = 40;
if (a > b) a = b;
else {
  a = c;
  b = 50;
  if (a > 50) a = 100;
}
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
```

Output:

```
a = 40, b = 50, c = 40
```

Problem 3

```
public char getGrade() {  // returns an 'X' if to many point, an 'Z' if to few points
  if (points < 0) return 'Z';
  else if (points < 60) return 'F';
  else if (points < 70) return 'D';
  else if (points < 80) return 'C';
  else if (points < 90) return 'B';
  else if (points <= 100) return 'A';
  else return 'X';
}
```

Problem 4

```
import javax.swing.*;
class Grade {
  private int points;
  public Grade(int initPoints) {
    points = initPoints;
  }
  public int getPoints() {
    return points;
  }
  public char getGrade() {  // returns an 'X' if to many point, an 'Z' if to few points
    if (points > 100) return 'X';
```

```
    else if (points >= 90) return 'A';
    else if (points >= 80) return 'B';
    else if (points >= 70) return 'C';
    else if (points >= 60) return 'D';
    else if (points >= 0) return 'F';
    else return 'Z';
  }
}
class Prob5_5_4 {
  public static void main(String[] args) {
    String pointsRead = JOptionPane.showInputDialog("The number of points: ");
    int points = Integer.parseInt(pointsRead);
    Grade theGrade = new Grade(points);
    char grade = theGrade.getGrade();
    if (grade == 'Z') {
      JOptionPane.showMessageDialog(null,
                                    "Negative number of points not allowed.");
    } else if (grade == 'X') {
      JOptionPane.showMessageDialog(null, "Max. number of points is 100");
    } else {
      JOptionPane.showMessageDialog(null,
                                    points + " points gives the grade " + grade);
    }
    System.exit(0);
  }
}
```

Chapter 5.6

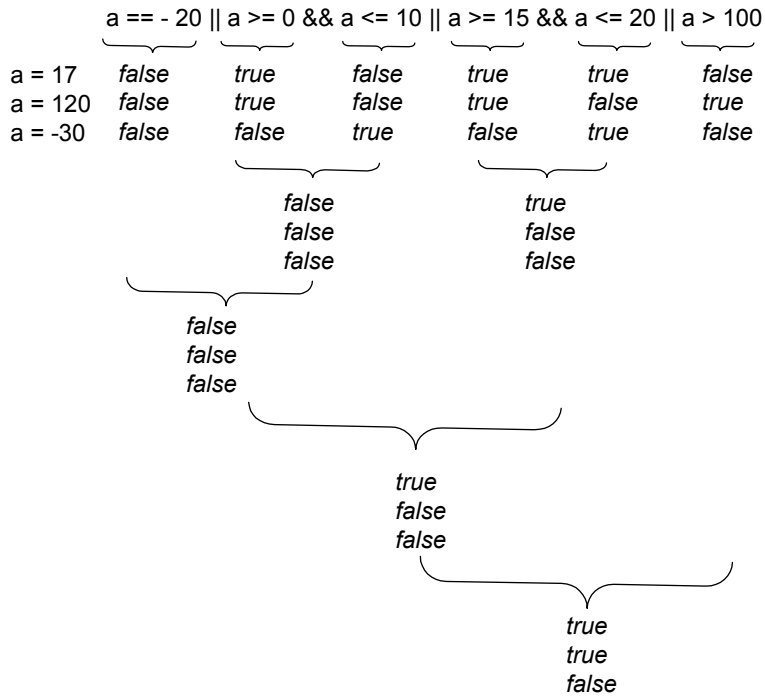
Problem 1

a) true b) false c) false d) true

Problem 2

```
a) noOfStudents > 20 && noOfStudents < 30
b) winningNo == 3 || winningNo == 18 || winningNo == 25
c) answer == 'y' || answer == 'Y'
d) temp < 15 || temp > 25
e) sum > 0 && sum <= 10 || sum >= 100
f)  character >= 'A' && character <= 'Z' || character >= 'a' && character <= 'z'
g) character >= '0' && character <= '9'
```

Problem 3



a = 17 and a = 120 makes the expression true, while a = -30 makes the expression false.

Chapter 5.7

Problem 1

Syntax error: We are not allowed to list more than one value in the same case label.

Logical error: We have to put break as the last statement under the case labels where we want the program control to jump out of the switch block.

The semicolon after the } at the end of the switch statement is not necessary. It is an empty statement.

We think that the following code segment is what the programmer has meant:

```
switch (dayOfWeek) {
  case 1:
    /* falls through */
  case 2:
    System.out.println("In the beginning of the week");
    break;
  case 3:
    /* falls through */
  case 4:
    System.out.println("In the middle of the week");
    break;
  case 5:
    System.out.println("Near the end of the week");
    break;
  case 6:
    /* falls through */
  case 7:
    System.out.println("Weekend");
    break;
  default:
    System.out.println("Invalid day");
    break;
}
```

Chapter 6.1

Problem 1

No times at all:

```
counter = 0;
while (counter < 0) {
  System.out.println("This is a line");
  counter++;
}
```

Infinite number of times (the counter is not updated):

```
counter = 0;
while (counter < 5) {
  System.out.println("This is a line");
}
```

Chapter 6.2

Problem 1

```
class Prob6_2_1 {
  public static void main(String[] args) {
    int smallNumbers = 0;
    int bigNumbers = 0;
    int number = (int)(500 * Math.random() + 1);
    while (number != 250) {
      if (number > 250) bigNumbers++;
      else smallNumbers++;
      number = (int)(500 * Math.random() + 1);
    }
    System.out.println("The number of big numbers: " + bigNumbers);
    System.out.println("The number of small numbers: " + smallNumbers);
  }
}
```

Problem 2

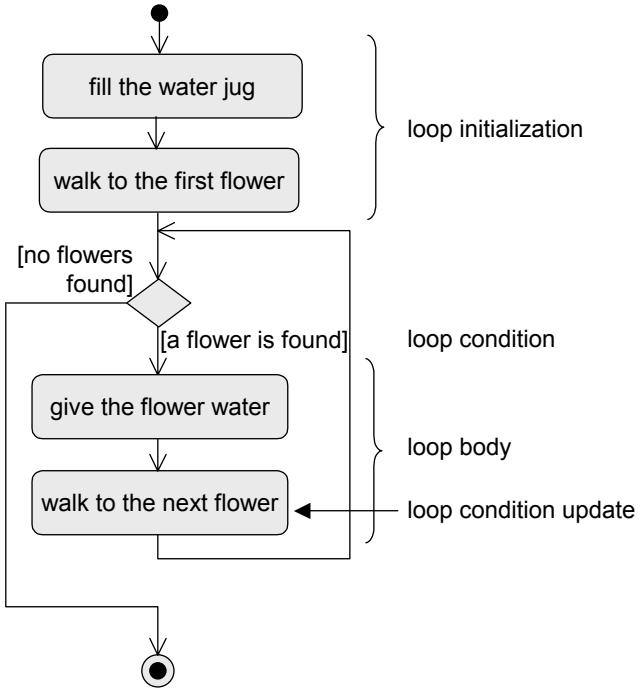
```
import javax.swing.JOptionPane;
class Prob6_2_2 {
  public static void main(String[] args) {
    String inputBase = JOptionPane.showInputDialog("What is the base? ");
    String inputExponent = JOptionPane.showInputDialog("What is the exponent? ");
    double x = Double.parseDouble(inputBase);
    int n = Integer.parseInt(inputExponent);
    double answer = 1; // a number raised to 0.power is 1, not 0
    int counter = 0; // the counter has to start at 0, if the condition should be the same
    while (counter < n) {
      answer *= x; // answer should be multiplied by x, not by n
      counter++; // counter should be increased, not n
    }
    JOptionPane.showMessageDialog(null, "The answer is " + answer);
    System.exit(0);
  }
}
```

Problem 3

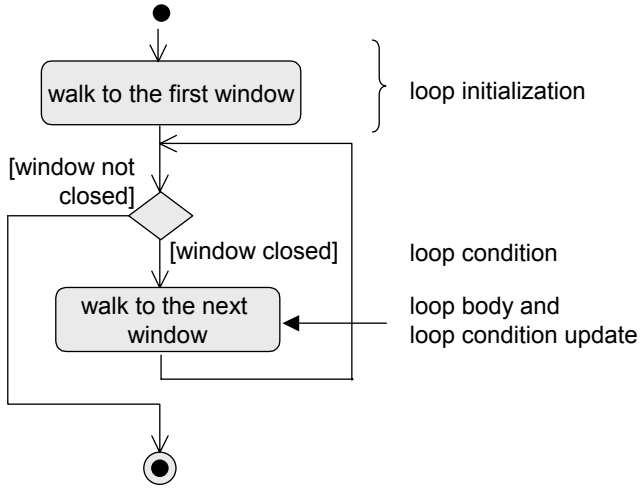
data set no.	n	expected answer with base -2	expected answer with base 0	expected answer with base +2
1	0	1	1	1
2	1	-2	0	2
3	3	-8	0	8

Problem 4a

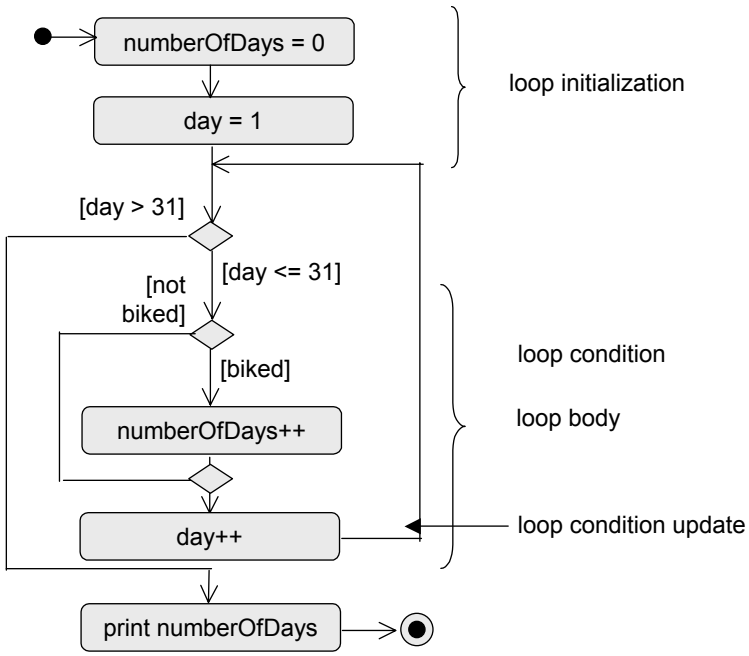
We assume one full jug of water is enough.



Problem 4b



Problem 4c



Chapter 6.3

Problem 1

```
public void paintComponent(Graphics window) {
  super.paintComponent(window);
  int x = (int) ((windowWidth - diameter) * Math.random() + 1);
  int y = (int) ((windowHeight - diameter) * Math.random() + 1);
  int x1 = x;  // Saves the coordinates for the first point
  int y1 = y;
  window.fillOval(x, y, diameter, diameter);
  int counter = 0;
  while (counter < noOfLines) {
    int previousX = x;
    int previousY = y;
    x = (int) ((windowWidth - diameter) * Math.random() + 1);
    y = (int) ((windowHeight - diameter) * Math.random() + 1);
```

```
    window.drawLine(previousX + center , previousY + center , x + center , y + center );
    window.fillOval(x, y, diameter, diameter);
    counter++;
   }
   /* Draws a line between the first and the last circle */
   window.drawLine(x + center , y + center , x1 + center , y1 + center );
  }
```

Problem 2

A boolean variable, filled, controls whether the circle should be filled or not. This variable changes its value before every circle drawing.

```
public void paintComponent(Graphics window) {
   super.paintComponent(window);
   int x = (int) ((windowWidth - diameter) * Math.random() + 1);
   int y = (int) ((windowHeight - diameter) * Math.random() + 1);
   boolean filled = true;
   window.fillOval(x, y, diameter, diameter);
   int counter = 0;
   while (counter < noOfLines) {
    int previousX = x;
    int previousY = y;
    x = (int) ((windowWidth - diameter) * Math.random() + 1);
    y = (int) ((windowHeight - diameter) * Math.random() + 1);
    window.drawLine(previousX + center, previousY + center, x + center, y + center);
    filled = !filled;
    if (filled) window.fillOval(x, y, diameter, diameter);
    else window.drawOval(x, y, diameter, diameter);
    counter++;
   }
  }
```

Chapter 6.4

Problem 1

```
public void paintComponent(Graphics window) {
  super.paintComponent(window);
  int x = 0;
  int y = 50;
  int diameter = 20;
  boolean filled = true;
  for (int counter = 0; counter < 30; counter++) {
    if (filled) window.fillOval(x, y, diameter, diameter);
    else window.drawOval(x, y, diameter, diameter);
    filled = !filled;
    x += diameter;
    diameter += 2;
  }
 }
```

Chapter 6.5

Problem 1

ABVVUIHJV

gives the output

```
VV
VVVV
VVVVVVVV
```

The program searches for V's in the text. The first V found results in two V's printed to the console, the second results in four V's, the third in eight V's, and so on.

Problem 2

```
class Drawing extends JPanel {
  private static final int windowWidth = 400;
```

```
  private static final int windowHeight = 300;
  private static final int startDiameter = 6;
  private static final int diameterIncr = 4;
  private static final int noOfLines = 10;

  public void paintComponent(Graphics window) {
    super.paintComponent(window);
    int x = (int) (windowWidth * Math.random() + 1);
    int y = (int) (windowHeight * Math.random() + 1);
    window.drawOval(x, y, startDiameter, startDiameter);
    int noOfCircles = 1;
    int counter = 0;
    while (counter < noOfLines) {
      int previousX = x;
      int previousY = y;
      x = (int) (windowWidth * Math.random() + 1);
      y = (int) (windowHeight * Math.random() + 1);
      int diameter = startDiameter;
      int radius = diameter / 2;
      window.drawLine(previousX + radius, previousY + radius, x + radius, y + radius);
      noOfCircles++;
      int thisX = x;
      int thisY = y;
      for (int circleNo = 0; circleNo < noOfCircles; circleNo++) {
        window.drawOval(thisX, thisY, diameter, diameter);
        diameter += diameterIncr;
        thisX -= diameterIncr / 2;
        thisY -= diameterIncr / 2;
      }
      counter++;
    }
  }
}
```

Chapter 6.8

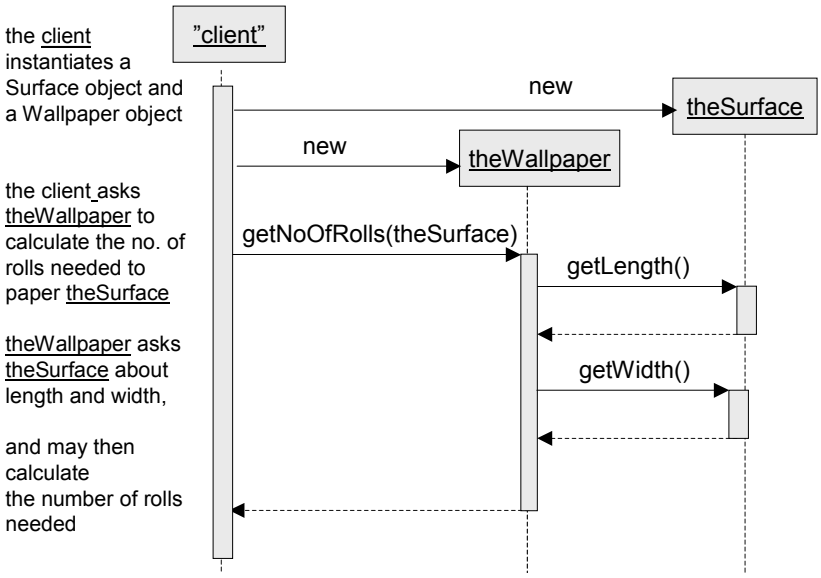
Problem 1

```
import javax.swing.JOptionPane;
class MyInputReader {

  public static double inputDecimalNumeralInInterval(
        String prompt, double lowerLimit, double upperLimit) {
    double number = 0.0;
    boolean ok = false;
    do {
      try {
        String theInput = JOptionPane.showInputDialog(prompt);
        number = Double.parseDouble(theInput);
        if (number >= lowerLimit && number <= upperLimit) ok = true;
      } catch (NumberFormatException e) {
      }
      /* We have to put the message after the catch-block to intercept the case where
         the input is a number outside the valid interval. */
      if (!ok) JOptionPane.showMessageDialog(null,
          "Your input cannot be converted into a number \n" +
          "between " + lowerLimit + " and " + upperLimit + ". Try again!");
    } while (!ok);
    return number;
  }
}

class Prob6_8_1 {
  public static void main(String[] args) {
    final double lowerLimitHeight = 1.5;
    final double upperLimitHeight = 7;
    final double lowerLimitLength = 1;
    final double upperLimitLength = 15;
```

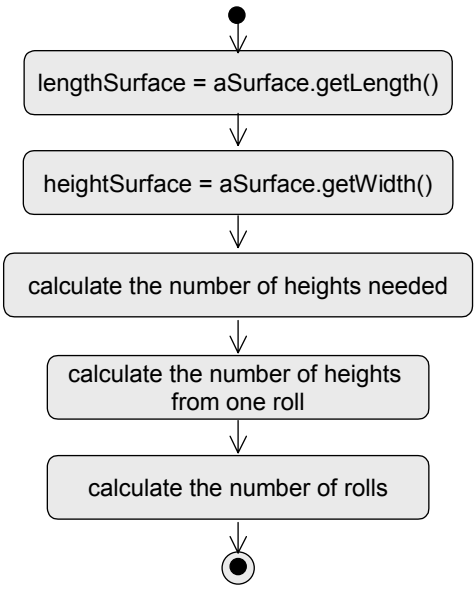
```
        double length = MyInputReader.inputDecimalNumeralInInterval(
           "The length of the wall (meters): ", lowerLimitLength, upperLimitLength);
        double height = MyInputReader.inputDecimalNumeralInInterval(
           "The height of the wall (meters): ", lowerLimitHeight, upperLimitHeight);

        double area = length * height;
        JOptionPane.showMessageDialog(null,
                "The area of the wall is " + area + " square meters.");
        System.exit(0);
      }
    }
```

Chapter 7.1

Problem 1a



Problem 1b



Problem 2

```
class Paint {
  private static final double limit = 0.02;  // limit to buy 0.5 liters more
  private String name;  // identification
  private double price;  // price per liter
  private int noOfCoats;
  private double noOfSqMPerLiter;
```

```
  public Paint(String initName, double initPrice, int initNoOfCoats,
     double initNoOfSqMPerLiter) {
   name = initName;
   noOfCoats = initNoOfCoats;
   noOfSqMPerLiter = initNoOfSqMPerLiter;
   price = initPrice;
  }

  public String getName() {
   return name;
  }

  public double getPricePerLiter() {
   return price;
  }

  public int getNoOfCoats() {
   return noOfCoats;
  }

  public double getNoOfSqMPerLiter() {
   return noOfSqMPerLiter;
  }

  /*
   * This method calculates the amount of paint required to cover the surface.
   * The result is rounded upward to the nearest 0.5 liters.
   */
  public double getNoOfLiters(Surface aSurface) {
   double area = aSurface.getArea();
   double noOfLiters = area * noOfCoats / noOfSqMPerLiter;
   int noOfLitersInteger = (int) noOfLiters;
   double more = noOfLiters - noOfLitersInteger;
   if (more >= 0.5 + limit) return noOfLitersInteger + 1.0;
   else if (more >= limit) return noOfLitersInteger + 0.5;
   else return noOfLitersInteger;
  }

  public double getTotalPrice(Surface aSurface) {
   return getNoOfLiters(aSurface) * price;
  }
 }
```

Problem 3

```
  public boolean isTheSameAge(Person p) {  // method a)
   if (p.yearOfBirth == yearOfBirth) return true;  // or we may use p.getYearOfBirth()
   else return false;
  }

 public int compareAge(Person p) {  // method b)
  if (yearOfBirth < p.yearOfBirth) return -1; // "this" is the elder one
  else if (yearOfBirth > p.yearOfBirth) return +1;
  else return 0;
 }

 public boolean severalYearsOlderThan(Person p, int noOfYears) {  // method c)
  if (yearOfBirth + noOfYears < p.yearOfBirth ) return true;
  else return false;
 }
```

We use the methods in a very simple client program:

```
 public static void main (String args []) {
  Person p1 = new Person("Peter", 1980);
  Person p2 = new Person("Jane", 1984);
  if (p1.isTheSameAge(p2)) System.out.println("The same age");
  else System.out.println("Not the same age");
  if (p1.compareAge(p2) < 0) System.out.println("p2 is the younger one ");
  else if (p1.compareAge(p2) > 0) System.out.println("p2 is the elder one");
```

```
    else System.out.println("p1 and p2 are the same age");
    if (p1.severalYearsOlderThan(p2, 3)) {
      System.out.println("p1 is more than 3 years older than p2");
    }
    if (p2.severalYearsOlderThan(p1, 3)) {
      System.out.println("p2 is more than 3 years older than p1");
    }
  }
```

Output:

```
Not the same age
p2 is the younger one
p1 is more than 3 years older than p2
```

Chapter 7.2

Problem 1

There are many instances of the String class. We do not mention every one of them. The JOptionPane class is used too, but there is no instances of this class in the programs. Only class methods are used.

We then go through the code, and see where instances of classes are created:

Program listing 7.2:

An instance of the ProjectChap7 class is created in the main() method in the RenovationChap7 class. In this object we have:

- The details object which is an instance variable of the ReaderRenovationCase class. It is instantiated in the declaration.
- Inside the carryOutTheRightThing() method we have references to instances of the Surface class, the Wallpaper class, the Paint class and the Flooring class. But the objects are *not* created here. They are created when the readAndInstantiate..()-messages are sent to the details-object. See the description of program listing 7.3 below.

Program listing 7.3:

Each of the readAndInstantiate..() methods create one instance of the actual class.

Problem 2

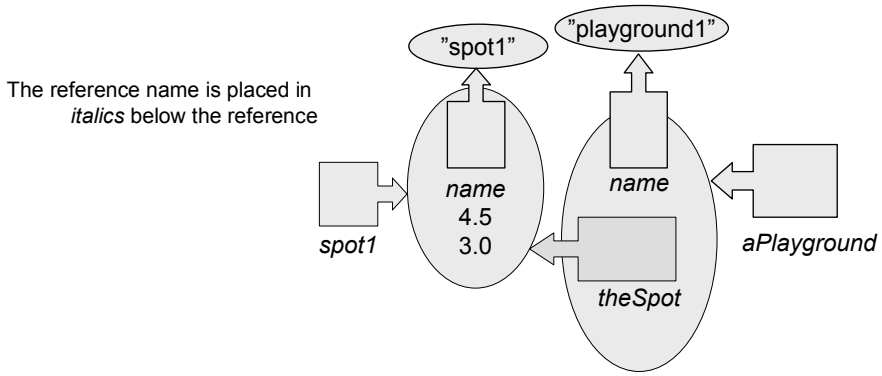
When this alternative loop is used and "exit" is selected, the program control will enter the carryOutTheRightThing() method and ask the user about the surface data. This should of course not happen when the user wants to exit the program. One solution to this is to insert an if statement to prevent this when "exit" is selected. But then we have a loop with the condition test in the beginning of the loop body, and then while should be used instead of do while.

Problem 3

The return value from the showOptionDialog() is 0, 1, 2 or 3, and this value is stored in the locale variable option in the main() method. This variable is passed to the carryOutTheRightThing() method, and it is there used in the switch statement, and the named constants are used in the case expressions.

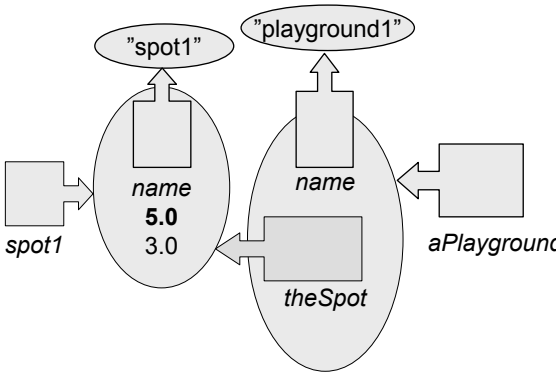
Chapter 7.3

Problem 1a



Problem 1b

The length of the spot is first chenged to 8.5, then to **5.0**.



Problem 1c

```
A: spot1 length: 4.5 width: 3.0
B: spot1 length: 5.0 width: 3.0
```

Problem 2

The method which does make a copy:

```
public void setFlooring1(Flooring newFlooring) {
  flooring = new Flooring(newFlooring.getName(),
          newFlooring.getPricePerM(), newFlooring.getWidth());
}
```

The method which does not make a copy:

```
public void setFlooring2(Flooring newFlooring) {
  flooring = newFlooring;
}
```

Chapter 7.4

Problem 1

The contents of the two variables are swapped:

```
Before: 3 4
Afterwards: 4 3
```

Problem 2

Here is the output:

```
Before: 10 4
Afterwards: 10 4
```

Swapping of parameters of primitive data types inside a method do not have any effect on the arguments, because the method only works with copies of the arguments.

Problem 3

The parameters are of reference types.

swapObjects1():

Inside the method the references get new values. This doesn't have any effect on the objects in the client program, because the method works with its own copies of the references.

swapObjects2():

This method does the same thing; it changes the reference values without effecting the objects in the client program.

Problem 4

For a method to be able to change the contents of the objects in the client program, it has to use set methods. The Surface class get three new methods:

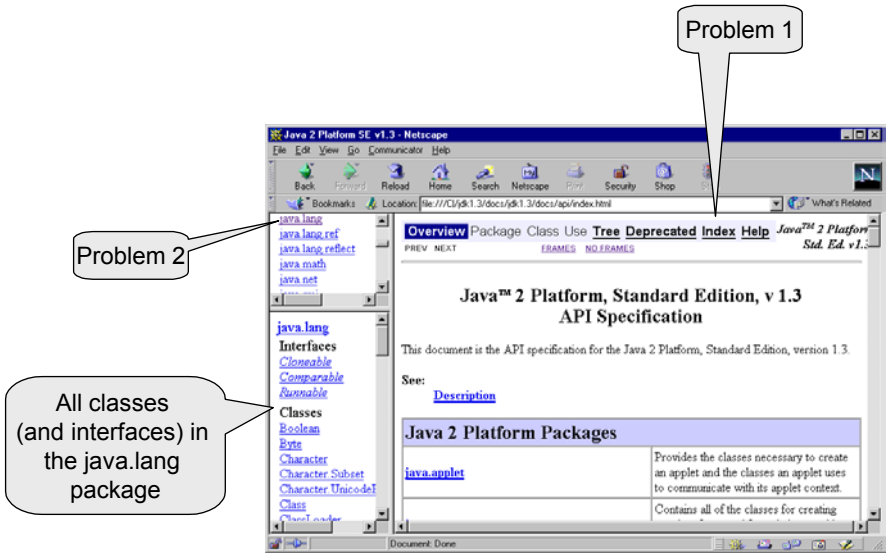
```
public void setName(String newName) {
  name = newName;
}

public void setLength(double newLength) {
  length = newLength;
}
```

```
public void setWidth(double newWidth) {
  width = newWidth;
}
```

The swap method in the Nonsense class comes here:

```
public void swapObjects(Surface f1, Surface f2) {
  Surface help = new Surface(f1.getName(), f1.getLength(), f1.getWidth());
  f1.setName(f2.getName());
  f1.setLength(f2.getLength());
  f1.setWidth(f2.getWidth());
  f2.setName(help.getName());
  f2.setLength(help.getLength());
  f2.setWidth(help.getWidth());
}
```

Chapter 8.1



Problem 1

Search for abs() in the Index. Look at the figure above. You'll find many methods with this name. Probably one of the abs() methods in the Math class is the one you're searching for.

Problem 2

Look at the figure. You should click on the correct package at the upper left of the window. Then, you'll get only the classes (and interfaces) of that package.

Problem 3

The main difference is that while StringBuffer is mutable, String is not. All the String methods which seem to change the string, do not edit the string object at all. Instead they create a new string. An example:

```
String t1 = new String("Arne");
String t2 = t1.toUpperCase();
System.out.println(t1 + " " + t2);
```

Output:

```
Arne ARNE
```

toUpperCase() creates a new object. To use this object in the rest of the program, we have to store the reference to it (t2).

The StringBuffer class offers methods for removing and inserting characters into a string:

```
StringBuffer s1 = new StringBuffer("Arne");
StringBuffer s2 = s1.insert(2, "xxxx");
System.out.println(s1 + " " + s2);
```

The output is:

```
Arxxxxne Arxxxxne
```

Chapter 8.3

Problem 1

```
import java.text.*;
import java.util.*;
class Prob8_3_1 {
  public static void main(String[] args) {
    Locale.setDefault(new Locale("en", "US"));
    DateFormat dateFormat1 =
      DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL);
    DateFormat dateFormat2 =
      DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG);
    DateFormat dateFormat3 =
      DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
                                     DateFormat.MEDIUM);
    DateFormat dateFormat4 =
      DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.SHORT);

    Date toDay = new Date();

    System.out.println("Format FULL: " + dateFormat1.format(toDay));
    System.out.println("Format LONG: " + dateFormat2.format(toDay));
    System.out.println("Format MEDIUM: " + dateFormat3.format(toDay));
    System.out.println("Format SHORT: " + dateFormat4.format(toDay));
  }
}
```

Output:

```
Format FULL: Saturday, October 20, 2001 5:31:53 PM GMT+02:00
Format LONG: October 20, 2001 5:31:53 PM GMT+02:00
Format MEDIUM: Oct 20, 2001 5:31:53 PM
Format SHORT: 10/20/01 5:31 PM
```

Chapter 8.5

Problem 1

Here is the output:

```
The value of i: 0
Division by zero!
For-loop no. 2 starts here:
The value of i: 0
Division by zero!
The value of i: 1
1
The value of i: 2
0
The value of i: 3
0
The value of i: 4
0
```

The first loop has the try-catch-block outside of it. Then the program control jumps out of the loop the first time an ArithmeticException is thrown.

The second loop has the try-catch-block around the loop body. If division by zero occurs, the message is output, and the program control continues with the first statement after the try-catch-block, which is the next run through the loop.

The reason why we get 0 when i = 2, 3 and 4, is that the result from the integer division is the quotient without the remainder.

Chapter 8.6

Problem 1

```
public Employee(int initNumber, String initName, int initSalary)
                                  throws IllegalArgumentException {
  if ((initNumber < limitNo1 || initNumber > limitNo2) && initSalary < limitSal) {
    throw new IllegalArgumentException("Both salary and number are invalid." +
      "\nSalary: $" + initSalary + ", it should be at least $" + limitSal +
```

```
        "\nNumber: " + initNumber + ", it should be in the interval [" +
                                        limitNo1 + ", " + limitNo2 + "].");
    }
    ....to be continued as in program listing 8.2...
```

Problem 2

```
int sum = 0;
try {
  String aLine = reader.readLine();
  while(aLine != null) {
    StringTokenizer text = new StringTokenizer(aLine);
    while (text.hasMoreTokens()) {
      String s = text.nextToken();
      try {
        int number = Integer.parseInt(s); // may throw NumberFormatException
        sum += number;
      } catch (NumberFormatException e) {
      }
    }
    aLine = reader.readLine();
  }
}
```

Chapter 9.1

Problem 1a

```
int[] noOfDays = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Problem 1b

```
int answer = JOptionPane.showConfirmDialog(null,
                                "Leap year?", "Year", JOptionPane.YES_NO_OPTION);
if (answer == JOptionPane.YES_OPTION) noOfDays[1] = 29;
```

Problem 2

```
public static void main(String[] args) {
  char[] name = {'A', 'N', 'N', 'E'};
  for (int i = name.length - 1; i >= 0; i--) System.out.print(name[i]);  // no new line here
  System.out.println();  // one new line at the end
}
```

Problem 3

The contents of the array are: 3, 8, 8, 3, 7, 14, 3, 17, 8, 9

Chapter 9.2

Problem 1

```
int arrLength = array1.length; // the same for array2 as for array1
  for (int i = 0; i < arrLength; i++) {
    array2[arrLength - 1 - i] = array1[i];
  }
```

Problem 2

The following exceptions might be thrown:

- IndexOutOfBoundsException: This exception is thrown if the copying process leads to invalid indexing in one or both of the arrays. You should perform a test like this to avoid the exception:

  ```
  if (fromIndex >= 0 && toIndex >= 0 && number >= 0 &&
        fromIndex + number <= fromArray.length &&
        toIndex + number <= toArray.length)  // copying is ok
  ```

- ArrayStoreException: This exception is thrown if either fromArray or toArray is not an array, or if the two arrays have conflicting data types. For example:

  ```
  int[] arr2 = {7, 14, -6, 0};
  double[] arr3 = new double[10];
  System.arraycopy(arr2, 1, arr3, 0, 3);  // ArrayStoreException is thrown
  ```

- NullPointerException: This exception is thrown if either fromArray or toArray is null. For example:

```
int[] arr2 = {7, 14, -6, 0};
int[] arr4 = null;
System.arraycopy(arr4, 1, arr2, 0, 3);  // NullPointerException is thrown
```

Problem 3

The printout is as follows:

```
Novemberr
Mayay
```

Chapter 9.3

Problem 1

```
public int getNoOfDaysMax() {
  int max = getMaximum();
  int numberOfMax = 0;
  for (int i = 0; i < precipitation.length; i++) {
    if (precipitation[i] == max) numberOfMax++;
  }
  return numberOfMax;
}
```

We try the method by inserting the following into the client program:

```
int noOfDays = oneMonth.getNoOfDaysMax();
System.out.println("The number of days with max. precipitation: " + noOfDays);
```

Problem 2

```
public int numberOfDaysLessThan(int limit) {
  int number = 0;
  for (int i = 0; i < precipitation.length; i++) {
    if (precipitation[i] < limit) number++;
  }
  return number;
}
```

In the client program:

```
int noOfDaysLessThan = oneMonth.numberOfDaysLessThan(4);
System.out.println("The number of days with precipitation less than 4: " +
                                                          noOfDaysLessThan);
```

Problem 3

```
public double getStdDev() {
  if (precipitation.length > 1) {
    double average = getAverage();
    double sumSquares = 0;
    for (int i = 0; i < precipitation.length; i++) {
      sumSquares += (average - precipitation[i]) * (average - precipitation[i]);
    }
    System.out.println(sumSquares);
    double radicand = sumSquares/ (precipitation.length - 1);
    return Math.sqrt(radicand);   // Return. Std.deviation according to formula
  } else return 0.0;        // Return. Too few data values.
}
```

In the client program:

```
double stdDev = oneMonth.getStdDev();
System.out.println("Standard deviation is: " + stdDev);
```

Chapter 9.4

Problem 1

The inequality sign must be "greater than" in stead of "less than". The smallestToNow variable should change its name to largestToNow to better describe its function:

```
public static void sortIntegerArray(int[] array) {
  for (int start = 0; start < array.length; start++) {
    int largestToNow = start;
    for (int i = start + 1; i < array.length; i++) {
      if (array[i] > array[largestToNow]) largestToNow = i;
    }
```

```
      int help = array[largestToNow];
      array[largestToNow] = array[start];
      array[start] = help;
    }
  }
```

Problem 2

```
  import myLibrary.*;
  class Prob9_4_2 {
    public static int[] sortAndRemoveDuplicates(int[] array) {
      Sort.sortIntegerArray(array);  // sorts the array
      if (array.length > 0) {
        int[] newArray = new int[array.length];
        newArray[0] = array[0];
        int number = 1; // the number of different numbers in the array
        int previousNumber  = array[0];
        int index = 1; // index in the sorted array

        /* runs through the whole array */
        while (index < array.length) {
          /* skips all duplicates */
          while (index < array.length && array[index] == previousNumber ) index++;

          /* if end of the array is not reached, we have found an element with another value */
          if (index < array.length) {
            previousNumber  = array[index];
            newArray[number] = array[index];
            number++;
          }
        }

        /* creates an array with the correct length */
        int[] newArray2 = new int[number];
        for (int i = 0; i < number; i++) {
          newArray2[i] = newArray[i];
        }
        return newArray2;
      }
      return null;
    }

    public static void main(String[] args) {
      int[] test = {3, 4, -5, 13, 10, 4, 11, 0, 8, -2, 11, 22, 15, 11, 9, 17, 4};
      int[] array = sortAndRemoveDuplicates(test);
      System.out.println("Sorted array, duplicates are removed: ");
      for (int i = 0; i < array.length; i++) System.out.print(array[i] + " ");
      System.out.println();
    }
  }

  /* Example Run:
  Sorted array, duplicates are removed:
  -5 -2 0 3 4 8 9 10 11 13 15 17 22
  */
```

Problem 3

```
  import myLibrary.*;

  class Prob9_4_3 {

    public static int[] addArrays(int[] arr1, int[] arr2) {
      if (arr1.length == arr2.length) {
        int[] sum = new int[arr1.length];
        for (int i = 0; i < arr1.length; i++) {
          sum[i] = arr1[i] + arr2[i];
        }
        return sum;  // return
      } else return null; // return, not the same length
```

```
  }

  public static void main(String[] args) {
    int[] test1 = {3, 4, -5, 13, 10, 4};
    int[] test2 = {3, 5, 6, 8, 9, 10};
    int[] sum = addArrays(test1, test2);
    for (int i = 0; i < sum.length; i++) {
      System.out.println(test1[i] + " + " + test2[i] + " = " + sum[i]);
    }
  }
}

/* Example Run:
3 + 3 = 6
4 + 5 = 9
-5 + 6 = 1
13 + 8 = 21
10 + 9 = 19
4 + 10 = 14
*/
```

Chapter 9.5

Problem 1

```
class Prob9_5_1 {

  public static int search(int[] array, int value) {
    int index = 0;
    while (index < array.length && array[index] <= value) {
      if (array[index] == value) return index;
      index++;
    }
    return -1;
  }

  public static void main(String[] args) {
    int[] test = {3, 4, 5, 13, 23, 40};
    System.out.println(search(test, 3));
    System.out.println(search(test, 40));
    System.out.println(search(test, 23));
    System.out.println(search(test, 30));
  }
}

/* Example Run:
0
5
4
-1
*/
```

Problem 2

```
class Prob9_5_2 {

  public static int search(int[] array, int value) {
    int index = 0;
    int indexSearchedFor = -1;  // if the value dosn't exist
    while (index < array.length && array[index] <= value) {
      if (array[index] == value) indexSearchedFor = index;
      index++;
    }
    return indexSearchedFor;
  }

  public static void main(String[] args) {
    int[] test = {3, 4, 5, 13, 23, 40};
    System.out.println(search(test, 3));
```

```
    System.out.println(search(test, 40));
    System.out.println(search(test, 23));
    System.out.println(search(test, 30));
  }
}

/* Example Run:
0
5
4
-1
*/
```

Chapter 9.6

Problem 1

```
class Prob9_6_1 {
  public static void main(String[] args) {
    int[] test = new int[8];
    java.util.Arrays.fill(test, 15);
    System.out.println("The array is filled with the number 15:");
    for (int i = 0; i < test.length; i++) System.out.print(test[i] + " ");
    System.out.println();
    int fromIndex = 3;
    int toIndex = 6;
    if (fromIndex >= 0 && fromIndex < test.length &&
        toIndex >= 0 && toIndex < test.length &&
        fromIndex <= toIndex) {
      java.util.Arrays.fill(test, fromIndex, toIndex, 20);
      System.out.println("The array is filled with the number 20 from index no. "
          + fromIndex + " inclusive to index no. " + toIndex + " exclusive" );
      for (int i = 0; i < test.length; i++) System.out.print(test[i] + " ");
      System.out.println();
    } else System.out.println("Invalid index.");
  }
}

/* Example Run:
The array is filled with the number 15:
15 15 15 15 15 15 15 15
The array is filled with the number 20 from index no. 3 inclusive to index no. 6 exclusive
15 15 15 20 20 20 15 15
*/
```

Problem 2

```
 class Prob9_6_2 {
  public static void main(String[] args) {
    int[] test = {-5, -2, 0, 3, 4, 4, 4, 8, 9, 11, 11};
    int value = 7;
    int index = java.util.Arrays.binarySearch(test, value);
    System.out.println("Return value: " + index);
    if (index < 0) { // the value is not found in the array
      int position = - index - 1; // the value should be inserted at this position
      int[] testNew = new int[test.length + 1];
      for (int i = 0; i < position; i++) testNew[i] = test[i];
      testNew[position] = value;
      for (int i = position; i < test.length; i++) testNew[i + 1] = test[i];
      test = testNew;
      System.out.println("The new array: ");
      for (int i = 0; i < test.length; i++) System.out.print(test[i] + " ");
      System.out.println();
    } else System.out.println("The value is found at index no." + index);
  }
}

/* Example Run:
Return value: -8
```

```
The new array:
-5 -2 0 3 4 4 4 7 8 9 11 11
*/
```

Chapter 9.7

Problem 1

The week numbers is 1 before the index number, for example: Week number 10 has index number 9. The same is true for the day numbers; the day with index number 0 is a Monday.

a)

```
Sales theYear2001 = new Sales("shoes", 52, 5);
```

b)

```
theYear2001.setSales(9, 0, 10000);
```

c)

```
theYear2001.setSales(7, 3, 12100);
```

d)

```
System.out.println("Total sales week 5: " + theYear2001.getSalesForAWeek(4));
```

e)

```
System.out.println("Sales for Monday, week 6: " + theYear2001.getSales(5, 0));
```

f)

```
System.out.println("Total sales for the whole year: " + theYear2001.getTotalSales());
```

g)

```
int day = theYear2001.getMostProfitDay();
System.out.println("The most profitable day of the week is day no. " + (day + 1)
                + " with sales: " + theYear2001.getSalesForAWeekDay(day));
```

Problem 2a

```
public int getAverageSalesPerWeek() {
  int sum = 0;
  for (int i = 0; i < getNoOfWeeks(); i++) {
    sum += getSalesForAWeek(i);
  }
  if (getNoOfWeeks() > 0) return sum / getNoOfWeeks();
  else return 0;
}
```

Problem 2b

```
 public int[] getMaxProfitableWeeks() {
  if (getNoOfWeeks() > 0) {
    /* searches for the maximum value */
    int max = getSalesForAWeek(0);
    for (int i = 1; i < getNoOfWeeks(); i++) {
      if (getSalesForAWeek(i) > max) max = getSalesForAWeek(i);
    }
    /* searches for all weeks with sales equal to this maximum value */
    int[] weekNo = new int[getNoOfWeeks()];  // an array which is big enough
    int counter = 0; // week counter
    for (int i = 0; i < getNoOfWeeks(); i++) {
      if (getSalesForAWeek(i) == max) {
        weekNo[counter] = i;
        counter++;
      }
    }
    /* creates an array with the correct length, and copyes the values */
    int[] weekNoMaxSales = new int[counter];
    for (int i = 0; i < counter; i++) {
      weekNoMaxSales[i] = weekNo[i];
    }
    return weekNoMaxSales;
  }
  else return null;
}
```

Problem 3a

The result array:

```
int[][] result = new int[noOfTeams][noOfPossibleResults];
```

The number of lines is the same as the number of teams. The number of columns is equal to noOfPossibleResults.

Column 0: number of games won.
Column 1. number of games drawn.
Column 2: number of games lost

Problem 3b

A complete program:

```
class Prob9_7_3 {
  private final int noOfPossibleResults = 3;
  private int pointsWon = 3;
  private int pointsDrawn = 1;
  private int pointsLost = 0;

  /* The result array:
   * The number of lines is the same as the number of teams.
   * The number of columns is equal to noOfPossibleResults.
   * Column 0: number of games won.
   * Column 1. number of games drawn.
   * Column 2: number of games lost
   */

  private int[][] result;

  public Prob9_7_3(int noOfTeams) {
    result = new int[noOfTeams][noOfPossibleResults];
    result[3][0] = 1;   //for testing
    result[3][1] = 2;
    result[3][2] = 3;
    result[5][0] = 1;
    result[5][1] = 1;
    result[5][2] = 1;
    result[6][0] = 4;
    result[6][1] = 2;
    result[6][2] = 1;
  }

  public int[] getNoOfPoints() {
    int noOfTeams = result.length;
    int[] points = new int[noOfTeams];
    for (int i = 0; i < points.length; i++) {
      points[i] = result[i][0] * pointsWon + result[i][1] * pointsDrawn
                  + result[i][2] * pointsLost;
    }
    return points;
  }

  public static void main(String[] args) {
    Prob9_7_3 soccer = new Prob9_7_3(10);
    int[] points = soccer.getNoOfPoints();
    for (int i = 0; i < points.length; i++) {
      System.out.println("Team no. " + i + ": " + points[i] + " points");
    }
  }
}

/* Printout:
Team no. 0: 0 points
Team no. 1: 0 points
Team no. 2: 0 points
Team no. 3: 5 points
Team no. 4: 0 points
```

```
Team no. 5: 4 points
Team no. 6: 14 points
Team no. 7: 0 points
Team no. 8: 0 points
Team no. 9: 0 points
*/
```

Chapter 9.8

Problem a)

```
private int[][][] apples = new int[3][10][5];
```

Problem b)

```
class Prob9_8_1 {

  // the array dimensions should be named constants
  public static final int NoOfFields = 3;
  public static final int NoOfTrees = 10; // in each field
  public static final int NoOfYears = 5;

  /* problem a /*
  private int[][][] apples = new int[NoOfFields][NoOfTrees][NoOfYears];


  /* problem b */
  public int[] getCropOnEachField() {
    int[] crop = new int[NoOfFields];
    for (int field = 0; field < NoOfFields; field++) {
      for (int tree = 0; tree < NoOfTrees; tree++) {
        for (int year = 0; year < NoOfYears; year++) {
          crop[field] += apples[field][tree][year];
        }
      }
    }
    return crop;
  }
}
```

Chapter 10.1

Problem 1

By running the code we'll get two exceptions:

- NullPointerException: The items array is an array of references. But the references do not point to objects. That's the reason why we get the exception. For example; items[1] is equal to null. What we try to do is to set a price. But we cannot set a price to an object that doesn't exist. What we have to do is to set the array elements to point to references, e.g:

  ```
  items[0] = new Merchandise("cheese ", 100, 75.50);
  ```

- ArrayIndexOutOfBoundException: This exception is thrown when we in the last statement tries to refer to the array element with index 3. It doesn't exist. The elements in an array with length 3 are numbered 0, 1 and 2.

Problem 2a

```
names[1] = new String("Paul");
```

or

```
names[1] = "Paul";
```

Problem 2b

```
names[3] = aName;
```

Problem 2c

```
int sumLength = 0;
for (int i = 0; i < names.length; i++) {
  sumLength += names[i].length();
}
System.out.println(sumLength);
```

Problem 2d

```
final char symbol = 'r';
int symbolCounter = 0;
for (int i = 0; i < names.length; i++) {
  int index = names[i].indexOf(symbol);
  while (index >= 0) {
    symbolCounter++;
    index = names[i].indexOf(symbol, index + 1);
  }
}
System.out.println("The number of '" + symbol + "' instances is: " + symbolCounter);
```

Problem 3a

```
Merchandise[] items = new Merchandise[4];
items[0] = new Merchandise("cheese", 100, 7);
items[1] = new Merchandise("hot dogs", 101, 6);
items[2] = new Merchandise("bananas", 102, 0.95);
items[3] = new Merchandise("apples", 103, 1.45);
```

Problem 3b

```
for (int i = 0; i < items.length; i++) {
  System.out.println(items[i].getName());
}
```

Problem 3c

```
double maxPrice = items[0].getPrice();
int indexMostExpensItem = 0;
for (int i = 1; i < items.length; i++) {
  if (items[i].getPrice() > maxPrice) {
    maxPrice = items[i].getPrice();
    indexMostExpensItem = i;
  }
}
System.out.println("The most expensive item is " +
     items[indexMostExpensItem].getName()  +
      ", and it costs $" + maxPrice + " without VAT.");
```

Problem 3d

```
for (int i = 0; i < items.length; i++) {
  double price = items[i].getPrice();
  price *= 1.07;
  items[i].setPrice(price);
}
```

Problem 4

```
for (int i = 0; i < names.length; i++) {
  copyOfNames[i] = new String(names[i]);
}
```

Problem 5

```
public static void main(String[] args) {
  for (int i = 0; i < args.length; i++) {
    System.out.println(args[i]);
  }
}
```

Chapter 10.2

Problem 1

```
ArrayList items = new ArrayList();
int prodNo = 100;
String prodName = JOptionPane.showInputDialog
                                    ("Product name (terminate by inputting no data): ");
prodName = prodName.trim();
while (prodName.length() > 0) {

  /* Reads the price.
     Instead of this bit of code you may use myLibrary.InputReader (sect. 8.2):
```

```
      double price = myLibrary.InputReader.inputDecimalNumeral(
                                                  "Price without VAT: ");
    */
    double price = 0.0;
    boolean ok = false;
    do {
      String theInputText = JOptionPane.showInputDialog("Price without VAT: ");
      try {
        price = Double.parseDouble(theInputText);
        ok = true;
      } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Invalid number!\n");
      }
    } while (!ok);

    Merchandise anItem = new Merchandise(prodName, prodNo, price);
    items.add(anItem);
    prodNo++;
    prodName = JOptionPane.showInputDialog(
                  "Product name (terminate by inputting no data): ");
    prodName = prodName.trim();
  }
```

Problem 2

```
  String result = "";
  for (int i = 0; i < items.size(); i++) {
    Merchandise anItem = (Merchandise)items.get(i);
    result += (anItem.getName() + ", no.: " + anItem.getNo() +
                                        ", price $: " + anItem.getPrice() + "\n");
  }
  JOptionPane.showMessageDialog(null, result);
```

Problem 3

```
  String search =
            JOptionPane.showInputDialog("Searching value (product name): ");
  search = search.trim();
  boolean found = false;
  Merchandise theItem = null;
  int index = 0;
  while (index < items.size() && !found) {
    theItem = (Merchandise)items.get(index);
    String name = theItem.getName();
    if (name.equals(search)) found = true;
    else index++;  // N.B. the index is updated only if the item is not found
  }
  if (found) {
    JOptionPane.showMessageDialog(null, "This item has no.: "
                  + theItem.getNo() + " and price $" + theItem.getPrice());
  } else {
    JOptionPane.showMessageDialog(null, "No item with the name " + search + " is found.");
  }
```

Problem 4

After the code in problem 3 is run, the index variable contains the index of the searched item. To remove it, we simply write:

```
  if (found) items.remove(index);
```

Chapter 10.3

Problem 1

```
  Integer anInteger = new Integer(15); // problem a)
  Character initial = new Character('S'); // problem b)
  int number = anInteger.intValue(); // problem c)
  String text1 = initial.toString(); // problem d)
  String text2 = anInteger.toString(); // problem e)
```

Problem 2

```
try {
  String text = "123456XX";
  int number = Integer.parseInt(text);
  System.out.println("The number is " + number);
} catch (NumberFormatException e) {
  System.out.println("The text cannot be converted to a number.");
}
```

Chapter 10.4

Problem 1

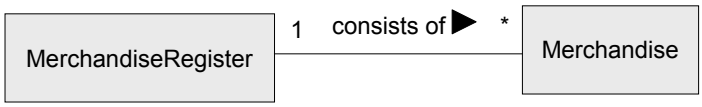
```
public String toString() {
  return "Paint. Name: " + name + ", no. of coats: " + noOfCoats +
                                          ", no. of sqm/liter: " + noOfSqMPerLiter;
}
```

The method may be used in the following way:

```
public static void main(String[] args) {
  Paint aPaint = new Paint("XXX", 12, 2, 12.5);
  System.out.println(aPaint);
}
```

Chapter 10.5

Problem 1



Operations on an object of the MerchandiseRegister class may be:

- Register a new item.
- Get the price of a special item.
- Create a list showing all the items.
- Remove an item from the register.
- Change the price of an item.

Problem 2

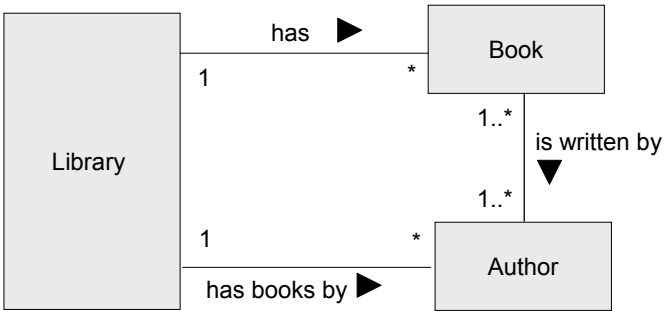
The lower part of the figure tells us the following:

- A person stays at exactly one office, neither more nor less.
- An office is occupied by exactly one person, neither more nor less.

The upper part of the figure tells:

- A book is written by *at least* one author.
- An author has written *at least* one book.

Problem 3



Chapter 10.6

Problem 1

```
public boolean removeSurface(String nameOfSurface) {
  for (int i = 0; i < allSurfaces.size(); i++) {
    Surface thisSurface = (Surface) allSurfaces.get(i);
```

```
      if ((thisSurface.getName()).equals(nameOfSurface)) {
        allSurfaces.remove(i);
        return true; // RETURN, surface is removed
      }
    }
    return false; // no surface with this name is found
  }
```

Problem 2

```
  public boolean removePaint(String nameOfPaint) {
    for (int paintIndex = 0; paintIndex < allPaints.size(); paintIndex++) {
      Paint thisPaint = (Paint) allPaints.get(paintIndex);
      if ((thisPaint.getName()).equals(nameOfPaint)) {
        /* Is the paint in use? */
        for (int surfaceIndex = 0; surfaceIndex < allSurfaces.size(); surfaceIndex++) {
          Surface thisSurface = (Surface) allSurfaces.get(surfaceIndex);
          if (thisSurface.getPaint() == thisPaint) {
            return false;  // RETURN. The paint is in use and cannot be removed.
          }
        }
        allPaints.remove(paintIndex);
        return true; // RETURN. Paint is removed.
      }
    }
    return false; // RETURN. No paint with this name.
  }
```

Test program:

```
  public static void main(String[] args) {
    RenovationProject project = new RenovationProject("Problem");
    Paint m1 = new Paint("Home Extra", 3, 10, 100);
    Paint m2 = new Paint("Home Super", 2, 12, 80);
    Surface f1 = new Surface("ceiling", 6, 1);
    Surface f2 = new Surface("wall", 4, 3);
    f1.setPaint(m1);
    f2.setPaint(m2);
    project.addNewSurface(f1);
    project.addNewSurface(f2);

    if (!project.removeSurface("floor")) System.out.println("floor not removed");
    if (!project.removePaint("HHH")) System.out.println("HHH not removed");
    if (!project.removePaint("Home Super")) System.out.println("Home Super not removed");

    /* changes paint on the "wall" */
    f2.setPaint(m1);

    /* Now it should be possible to remove m2 */
    if (project.removePaint("Home Super")) System.out.println("Home Super is removed");
    if (project.removeSurface("ceiling")) System.out.println("ceiling is removed");

    /* Control printouts */
    for (int i = 0; i < project.getNoOfSurfaces(); i++) {
      Surface aSurface = project.getSurface(i);
      System.out.println("Surface, index: " + i + ": " + aSurface.getName());
    }

    for (int i = 0; i < project.getNoOfPaints(); i++) {
      Paint m = project.getPaint(i);
      System.out.println("Paint, index: " + i + ": " + m.getName());
    }
  }
  /* Example Run:
  floor not removed
  HHH not removed
  Home Super not removed
```

```
Home Super is removed
ceiling is removed
Surface, index: 0: wall
Paint, index: 0: Home Extra
*/
```

Chapter 10.7

Problem 1

The header in the Merchandise class should be:

```
class Merchandise implements Comparable {
```

and the class should have a new method:

```
public int compareTo(java.lang.Object anotherItem) {
  Merchandise item2 = (Merchandise)anotherItem;
  if (merchandiseNo < item2.merchandiseNo) return -1;
  else if (merchandiseNo > item2.merchandiseNo) return 1;
  else return 0;
}
```

Client program:

```
public static void main(String[] args) {
  Merchandise v1 = new Merchandise("cheese", 100, 7);
  Merchandise v2 = new Merchandise("hot dog", 101, 6);
  Merchandise v3 = new Merchandise("banana", 101, 0.95);

  /* The printout should be -1 1 0 */
  System.out.println(v1.compareTo(v2));
  System.out.println(v2.compareTo(v1));
  System.out.println(v2.compareTo(v2));
}
```

Chapter 10.8

Problem 1a

```
String[] names = {"Maud", "Amy", "Nella", "Louis", "John"};
```

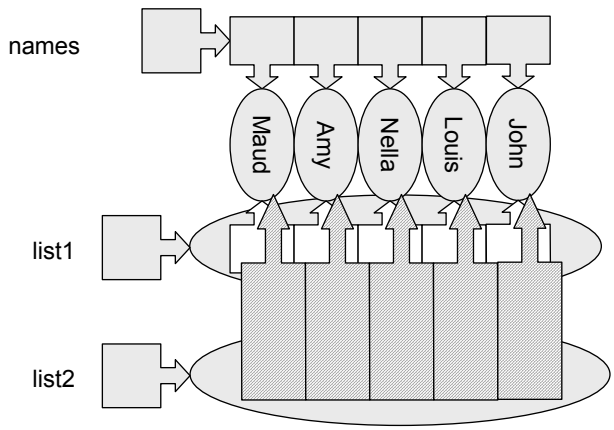
Problem 1b

```
ArrayList list1 = new ArrayList();
for (int i = 0; i < names.length; i++) {
  list1.add(names[i]);
}
```

Problem 1c

```
ArrayList list2 = new ArrayList();
for (int i = 0; i < list1.size(); i++) {
  list2.add(list1.get(i));
}
```

Problem 1d



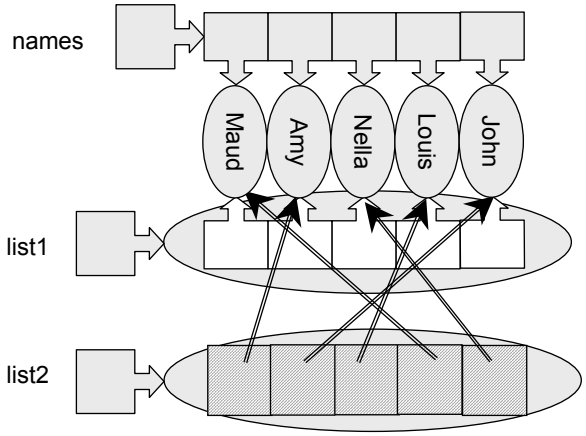
Here we have three references to each object. We have one reference from the names array, and one from each of the arrays encapsulated inside the array list objects list1 and list2.

Problem 1e

```
Collections.sort(list2);
```

Result: Amy, John, Louis, Maud, Nella

Problem 1f



The references in the sorted array list (list2) are now referring to the objects in sorted order.

Problem 2

```
public static void main(String[] args) {
  Merchandise[] items = new Merchandise[4];
  items[0] = new Merchandise("cheese", 110, 7);
  items[1] = new Merchandise("hot dogs", 101, 6);
  items[2] = new Merchandise("banana", 106, 0.95);
  items[3] = new Merchandise("apples", 100, 1.45);
  java.util.Arrays.sort(items);
  for (int i = 0; i < items.length; i++) {
    System.out.println(items[i].getNo() + " " + items[i].getName());
  }
}
```

The printout:

```
100 apples
101 hot dogs
106 banana
110 cheese
```

Problem 3

```
import java.util.*;
import java.text.*;
class Prob10_8_3 {
  public static void main(String[] args) {
    Locale.setDefault(new Locale("no", "No"));
    String[] names = {"Åsmund", "Petter", "Synnøve", "Øyvind", "Einar"};
    Collator collator = Collator.getInstance();
    Arrays.sort(names, collator);
    for (int i = 0; i < names.length; i++) {
      System.out.println(names[i]);
    }
  }
}
```

Chapter 11.3

Problem 1

```
import java.io.*;
class Prob11_3_1 {
  public static void main(String[] args) throws Exception {
    final String nameInFile = "mineData.txt";
    FileReader readConnToFile = new FileReader(nameInFile);
    BufferedReader reader = new BufferedReader(readConnToFile);
```

```
    int lineCounter = 0;
    String aLine = reader.readLine();
    while(aLine != null) {
      lineCounter++;
      aLine = reader.readLine();
    }
    System.out.println("Number of lines read: " + lineCounter);
    reader.close();
  }
}
```

a) If the file doesn't exist, a FileNotFoundException is thrown.

c) The file name has to be changed to: "MineData" + File.separator + "mineData.txt"

Chapter 11.4

Problem 1

```
import java.io.*;
class Prob11_4_1 {
  public static void main(String[] args) throws Exception {
    final String nameInFile = "mineData.txt";
    final String nameOutFile = "yoursData.txt";
    FileReader readConnToFile = new FileReader(nameInFile);
    BufferedReader reader = new BufferedReader(readConnToFile);
    FileWriter writeConnToFile = new FileWriter(nameOutFile, false);
    PrintWriter printer = new PrintWriter(new BufferedWriter(writeConnToFile));

    int lineCounter = 0;
    String aLine = reader.readLine();
    while(aLine != null) {
      lineCounter++;
      aLine = aLine.toUpperCase();
      printer.println(aLine);
      aLine = reader.readLine();
      }
    reader.close();
    printer.close();
    System.out.println("Number of lines read: " + lineCounter);
  }
}
```

Chapter 11.6

Problem 1

```
import java.util.*;
import java.io.*;
class NumberReader {
  private BufferedReader reader;
  public NumberReader(BufferedReader initReader) {
    reader = initReader;
  }

  /*
   * The method returns null if the line contains at least one invalid integer.
   * EOFException is thrown if no data is read.
   */
  public int[] readLineOfIntegers() throws IOException {
    try {
      String line = reader.readLine();
      if (line != null) {
        StringTokenizer text = new StringTokenizer(line);
        int[] array = new int[text.countTokens()];
        int index = 0;
        while (text.hasMoreTokens()) {
          String s = text.nextToken();
          array[index] = Integer.parseInt(s);
          index++;
        }
```

```
        return array;
      } else throw new EOFException("No data found.");
    } catch (NumberFormatException e) {
      return null;
    }
  }
  /*
   * The method returns null if the line contains at least one invalid decimal numeral.
   * EOFException is thrown if no data is read.
   */
  public double[] readLineOfDecimals() throws IOException {
    try {
      String line = reader.readLine();
      if (line != null) {
        StringTokenizer text = new StringTokenizer(line);
        double[] array = new double[text.countTokens()];
        int index = 0;
        while (text.hasMoreTokens()) {
          String s = text.nextToken();
          array[index] = Double.parseDouble(s);
          index++;
        }
        return array;
      } else throw new EOFException("No data found.");
    } catch (NumberFormatException e) {
      return null;
    }
  }
}

class Prob11_6_1 {
  public static void main(String[] args) throws Exception {
    String fileName = "numberFile.txt";
    FileReader connToFile = new FileReader(fileName);
    BufferedReader readerB = new BufferedReader(connToFile);
    NumberReader reader = new NumberReader(readerB);

    /* the data file contains 3 lines of integers and 3 lines of decimal numerals */
    for (int i = 0; i < 3; i++) {
      int[] number1 = reader.readLineOfIntegers();
      if (number1 == null) System.out.println("Invalid integers");
      else {
        for (int j = 0; j < number1.length; j++) System.out.print(number1[j] + " ");
        System.out.println();
      }
    }
    for (int i = 0; i < 3; i++) {
      double[] number2 = reader.readLineOfDecimals();
      if (number2 == null) System.out.println("Invalid decimal numerals");
      else {
        for (int j = 0; j < number2.length; j++) System.out.print(number2[j] + " ");
        System.out.println();
      }
    }
  }
}

/*
Data file:
1 2 3 4 5.0
67 189 78
5
1.2 3.5 6 78
34.56 56 5.7
4e

Printout:
```

```
Invalid integers
67 189 78
5
1.2 3.5 6.0 78.0
34.56 56.0 5.7
Invalid decimal numerals
*/
```

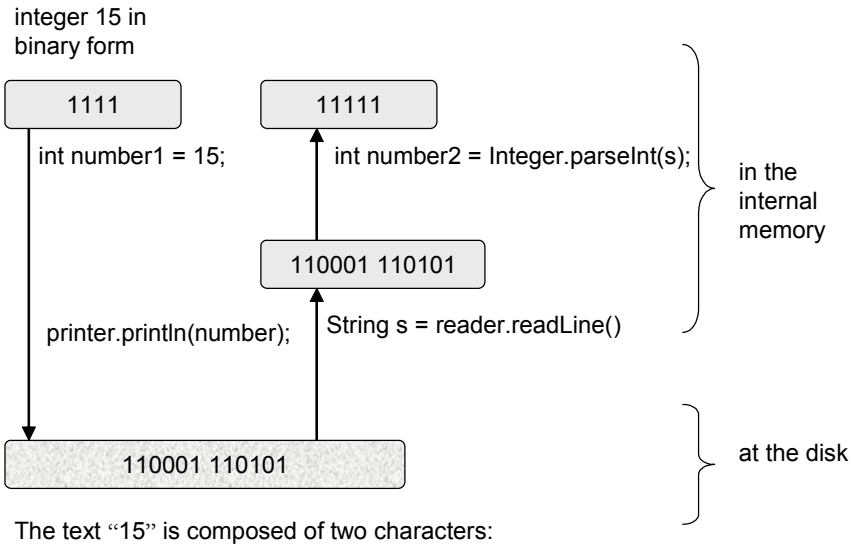
Chapter 11.7

Problem 1

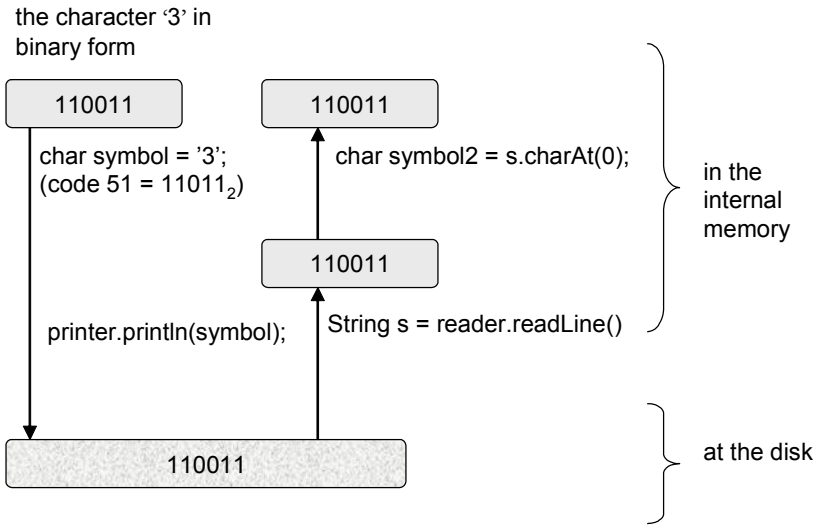
```
import java.util.*;
import java.io.*;
class Prob11_7_1 {
  public static void main(String[] args) throws Exception {
    InputStreamReader readConnToConsole = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(readConnToConsole);
    System.out.print("Enter some integers, terminate by pressing CTRL+Z on its own line: ");
    System.out.flush();   // the buffer has to be emptied after print()
    int sum = 0;
    String aLine = reader.readLine();
    while(aLine != null) {
      StringTokenizer text = new StringTokenizer(aLine);
      while (text.hasMoreTokens()) {
        String s = text.nextToken();
        int number = Integer.parseInt(s);  // the validity is not checked
        sum += number;
      }
      aLine = reader.readLine();
    }
    System.out.println("The sum of the numbers is " + sum);
  }
}
/* Example Run:

Enter some integers, terminate by pressing CTRL+Z on its own line: 3 4
5
6
7 8
^Z
The sum of the numbers is 33
*/
```

Chapter 11.8

Problem 1



The text "15" is composed of two characters:
'1' (code 49 = 110001_2) and '5' (code 53 = 110101_2).
End-of-line follows the text; it is not shown here.



Chapter 11.9

Problem 1

We'll get one space for each character, that is "T e x t" becomes "T e x t" (three spaces between each letter). The following program shows this:

```
import java.io.*;
class Prob11_9_1 {
  public static void main(String[] args) throws Exception{
    String fileName = "text.dat";

    /* Running this program several times with the same file
       gives different results each time (more and more spaces...).
       Therefore the program starts with deleting the file,
       if it exists. */
    File file1 = new File(fileName);  // see online API documentation
    file1.delete();

    RandomAccessFile file = new RandomAccessFile(fileName, "rw");
    file.writeChars("Text\n");
    System.out.println("Length: " + file.length());
    file.seek(0);
    String s = file.readLine();
    System.out.println(s);
    file.seek(0);
    file.writeChars(s);
    file.seek(0);
    s = file.readLine();
    System.out.println(s);
  }
}

/* Printout:
Length: 10
 T e x t
   T   e   x   t
*/
```

Problem 2

We have to save the size of the group (the number of integers in the group) before the data in each group.

Chapter 11.10

Problem 1

The Merchandise has to implement the java.io.Serializable interface. It means that the class header has to be like this:

```
class Merchandise implements Serializable {
```

Problem 2

The following program creates the array, fills it with data, and saves the array as one object to a file. Then the contents of this file are read into a new array, and then this array is printed on the screen:

```
public static void main(String[] args) throws Exception {
  Merchandise[] arr1 = new Merchandise[5];
  arr1[0] = new Merchandise("cheese", 100, 7);
  arr1[1] = new Merchandise("hot dogs", 101, 6);
  arr1[2] = new Merchandise("banana", 102, 0.95);
  arr1[3] = new Merchandise("cucumber", 103, 0.95);
  arr1[4] = new Merchandise("tomato", 104, 1.85);

  FileOutputStream outstream = new FileOutputStream("items.ser");
  ObjectOutputStream out = new ObjectOutputStream(outstream);
  out.writeObject(arr1);
  out.close();

  FileInputStream instream = new FileInputStream("items.ser");
  ObjectInputStream in = new ObjectInputStream(instream);
  Merchandise[] arr2 = (Merchandise[]) in.readObject();
  in.close();

  for (int i = 0; i < arr2.length; i++) {
    System.out.println(arr2[i].getName() + ", " + arr2[i].getNo() + ", " + arr2[i].getPrice());
  }
}
```

The printout:

```
cheese, 100, 7.0
hot dogs, 101, 6.0
banana, 102, 0.95
cucumber, 103, 0.95
tomato, 104, 1.85
```

The conclusion is that array objects can be serialized.

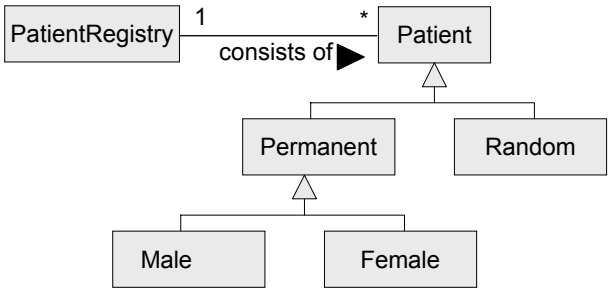
Problem 3

This program creates an array of integers (arr1). The array object is then serialized (saved as one object at a file). Then it is read into a new variable (arr2), and then printed out.

```
public static void main(String[] args) throws Exception {
  int[] arr1 = {1, 12, 45, 34, 56};
  FileOutputStream outstream = new FileOutputStream("numbers.ser");
  ObjectOutputStream out = new ObjectOutputStream(outstream);
  out.writeObject(arr1);
  out.close();

  FileInputStream instream = new FileInputStream("numbers.ser");
  ObjectInputStream in = new ObjectInputStream(instream);
  int[] arr2 = (int[]) in.readObject();
  in.close();

  for (int i = 0; i < arr2.length; i++) {
    System.out.println(arr2[i]);
  }
}
```

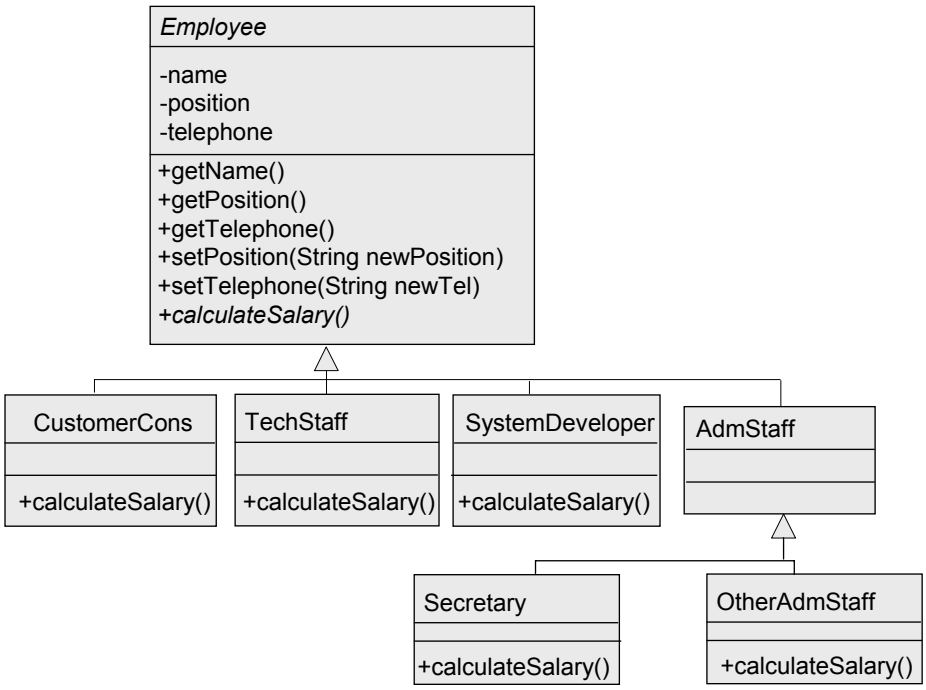
Chapter 12.1

Problem 1



Chapter 12.2

Problem 1



Problem 2

Polymorphism: The calculateSalary() operation is polymorphous, it has one version for each category of employees. We have chosen not to calculate any salary at all in the Employee class. But what *is a point*, is that it is possible to calculate the salary for *all* employees. We tell this by declaring the calculateSalary() operation as abstract in the Employee class. As a client we may send the message calculateSalary() to whichever employee object, the object know by itself the way to calculate the salary.

Problem 3

Inheritance: The CustomerCons, TechStaff, SystemDeveloper, AdmStaff, Secretary and OtherAdmStaff classes inherits the following operations from the Employee class: getName(), getPosition(), getTelephone(), setPosition(), setTelephone().

The AdmStaff class inherits the abstract calculateSalary() operation, and because of that the AdmStaff class becomes abstract, too. All the other subclasses have their own versions (implementations) of that method, and they are therefore not abstract.

Chapter 12.3

Problem 1

```
class Parquet extends Material {
  private static final double limit = 0.00001;
  private double noOfSqmPerPk;
```

```
  public Parquet(String initName, double initPrice, double initNoOfSqmPerPk) {
    super(initName, initPrice);
    noOfSqmPerPk = initNoOfSqmPerPk;
  }

  public double getNoOfSqMPerPk() {
    return noOfSqmPerPk;
  }

  /* The return type should have been int (a whole no. of packages),
     but it has to be double, because this is an overridden method. */
  public double getMaterialReq(Surface aSurface) {
    double area = aSurface.getArea();
    double req = area / noOfSqmPerPk;

    /* Rounds up to an integer number of packages */
    if (area % noOfSqmPerPk > limit) req = (int)(req + 1);
    return req;
  }
}

class Prob12_3_1 {
  public static void main(String[] args) {
    Surface diningRoom = new Surface("Floor in dining room", 3, 4);
    Surface hall = new Surface("Hall", 1.5, 3);
    Parquet parquet = new Parquet("Oak", 28.95, 10);
    System.out.println("Parquet: " + parquet.getName() +
          ", price: " + parquet.getPricePerUnit() + ", no. of sqm. per package.: " +
          parquet.getNoOfSqMPerPk());
    System.out.println("Material req. dining room: " +
         (int) parquet.getMaterialReq(diningRoom) +
         " packages, price: " + parquet.getTotalPrice(diningRoom));
    System.out.println("Material req. hall: " + (int) parquet.getMaterialReq(hall)  +
        " packages, price: " + parquet.getTotalPrice(hall));
  }
}

/* Printout:
Parquet: Oak, price: 28.95, no. of sqm. per package.: 10.0
Material req. dining room: 2 packages, price: 57.9
Material req. hall: 1 packages, price: 28.95
*/
```

Problem 2

```
/* The class is mutable, the position and telephone number may be changed */
abstract class Employee {
  private String name;
  private String position;
  private String telephone;
  public Employee(String initName, String initPosition, String initTelephone) {
    name = initName;
    position = initPosition;
    telephone = initTelephone;
  }

  public String getName() {
    return name;
  }

  public String getPosition() {
    return position;
  }
```

```
  public String getTelephone() {
    return telephone;
  }

  public void setPosition(String newPosition) {
    position = newPosition;
  }

  public void setTelephone(String newTelephone) {
    telephone = newTelephone;
  }

  public abstract double calculateSalary();
}

class CustomerConsultant extends Employee {
  public CustomerConsultant(String initName, String initPosition, String initTelephone) {
    super(initName, initPosition, initTelephone);
  }

  public double calculateSalary() {
    System.out.println(
            "An algorithm to calculate the salary for customer consultants " +
                        "is not implemented yet.");
    return 0.0;
  }
}

class TechnicalStaff extends Employee {
  public TechnicalStaff(String initName, String initPosition, String initTelephone) {
    super(initName, initPosition, initTelephone);
  }

  public double calculateSalary() {
    System.out.println("An algorithm to calculate the salary for technical personell " +
                        "is not implemented yet.");
    return 0.0;
  }
}
```

Problem 3

In the Employee class the method now is as follows:

```
  public double calculateSalary() {
    System.out.println("Salary is calculated in the standard way.");
    return 0.0;
  }
```

The Employee class needs not to be abstract. If the calculateSalary() is not implemented in a subclass, the version above will be inherited.

Problem 4

Now we assume that the Employee and AdmStaff classes are abstract. Then it is not possible to create instances of these classes. However, we may create instances of all the other classes. The following messages may be sent to these objects: getName(), getPosition(), getTelephone(), setPosition(), setTelephone() and calculateSalary().

A little test program:

```
public static void main(String[] args) {
  CustomerConsultant consultant = new CustomerConsultant(
              "Ann Smith", "Rome Office", "56766");
  TechnicalStaff technician = new TechnicalStaff(
               "Mary Brown", "Position 123", "45345");

  /* Tries all the inherited methods */
  consultant.setPosition("Milan Office");
  consultant.setTelephone("45454");
```

```
    System.out.println("The customer consultant: " + consultant.getName() + ", "
      + consultant.getPosition() + ", " + consultant.getTelephone());

    technician.setPosition("Position 999");
    technician.setTelephone("67676");
    System.out.println("the technician: " + technician.getName() + ", "
      + technician.getPosition() + ", " + technician.getTelephone());

    /* Tryes the calculateSalary() method */
    System.out.println("\nSalary for the consultant: " + consultant.calculateSalary());
    System.out.println("\nSalary for the technician: " + technician.calculateSalary());
  }
```

Chapter 12.4

Problem 1

```
  public static void main(String[] args) {
     Employee[] employees = new Employee[5];
     employees[0] = new CustomerConsultant("Anne Hansen", "Trondheim", "23456");
     employees[1] = new CustomerConsultant("Inger Huseby", "Oslo 2", "98767");
     employees[2] = new TechnicalStaff("Anders Jensen", "NT", "67456");
     employees[3] = new TechnicalStaff("Eva Hansen", "Novell", "89563");
     employees[4] = new CustomerConsultant("Torunn Ski", "Oslo 2", "78452");
     for (int i = 0; i < employees.length; i++) {
       System.out.println(employees[i].getName() +
                                              " has salary: " + employees[i].calculateSalary());
     }
```

Problem 2

```
  public static void main(String[] args) {
    ArrayList employees = new ArrayList();
    Employee anEmployee =
                  new CustomerConsultant("Anne Hansen", "Trondheim", "23456");
    employees.add(anEmployee);
    anEmployee = new CustomerConsultant("Inger Huseby", "Oslo 2", "98767");
    employees.add(anEmployee);
    anEmployee = new TechnicalStaff("Anders Jensen", "NT", "67456");
    employees.add(anEmployee);
    anEmployee = new TechnicalStaff("Eva Hansen", "Novell", "89563");
    employees.add(anEmployee);
    anEmployee = new CustomerConsultant("Torunn Ski", "Oslo 2", "78452");
    employees.add(anEmployee);
    for (int i = 0; i < employees.size(); i++) {
      Employee theEmployee = (Employee) employees.get(i);
      System.out.println(theEmployee.getName() +
                                               " has salary: " + theEmployee.calculateSalary());
    }
  }
```

Problem 3

The loop from problem 2:

```
  for (int i = 0; i < employees.size(); i++) {
    Employee theEmployee = (Employee) employees.get(i);
    if (theEmployee instanceof CustomerConsultant) {
       System.out.print("Customer consultant: ");
    } else System.out.print("Technical personnel: ");
   System.out.println(theEmployee.getName() + " has salary: " +
                                                    theEmployee.calculateSalary());
  }
```

Problem 4

We have an abstract method in the Employee class:

```
  public abstract String getClassname();
```

The method is implemented in the CustomerConsultant class:

```
  public String getClassname() {
    return "CustomerConsultant";
```

```
}
```

And in the TechnicalStaff class:

```
public String getClassname() {
  return "TechnicalStaff";
}
```

The loop from problem 3:

```
for (int i = 0; i < employees.size(); i++) {
  Employee theEmployee = (Employee) employees.get(i);
  System.out.println(theEmployee.getClassname() + ": " +
     theEmployee.getName() + " has salary: " + theEmployee.calculateSalary());
}
```

Chapter 12.5

Problem 1

"New Paint", "New Flooring" and "New Wallpaper" are parts of the menu from which the user chooses a material, see the right part of figure 12.7. The length of this menu depends on the number of materials already registered. Take a look at the getMaterialIndex() method. This method fetches the user's choice. If the user has chosen one of the three last alternatives, the choice is equal to one of the constants mentioned. But why do the constants have negative values? See the getMaterial() method. This method receives the return value from the getMaterialIndex() method. A non negative value means that the material is registered on that index, while a negative value means that new data are to be entered.

Problem 2

We may use instanceof or we create a polymorphous method getClassName(). See the solutions to problems 3 and 4, section 12.4.

Problem 3

New method in the ReaderRenovationCase class:

```
public Parquet readAndInstantiateParquet() {
  String nameParquet = InputReader.inputText(
                                          "Name or number to indentify this parquet: ");
  double noOfSqMPerPk =
            readPositiveDecimalNumeral("No. of sq.m. per package: ");
  double price = readPositiveDecimalNumeral("Price per package: ");
  Parquet aParquet = new Parquet(nameParquet, price, noOfSqMPerPk);
  return aParquet;
}
```

The rest of the changes apply to the ProjectChap12 class:

A new named constant:

```
public static final int newParquet = -5;
```

In the getMaterial() method we have to insert a new else-if:

```
if (materialIndex == newPaint) aMaterial = details.readAndInstantiatePaint();
else if (materialIndex == newFlooring) {
  aMaterial = details.readAndInstantiateFlooring();
} else if (materialIndex == newParquet) {
  aMaterial = details.readAndInstantiateParquet();
} else aMaterial = details.readAndInstantiateWallpaper();
```

In the getMaterialIndex() method, the number of alternatives will increase by 4 in stead of 3:

```
int noOfOptions = noOfMaterials + 4;
```

and a few lines further down:

```
options[noOfMaterials + 3] = new String("New Parquet");
```

and further:

```
if (theMaterial < options.length - 4) {
  return theMaterial; // Return, we are going to use the material with this index
}
else if (theMaterial == noOfMaterials) return newPaint;
else if (theMaterial == noOfMaterials + 1) return newFlooring;
else if (theMaterial == noOfMaterials + 2) return newWallpaper;
else if (theMaterial == noOfMaterials + 3) return newParquet;
```

```
return -1;  // Return (error?)
```

Chapter 12.6

Problem 1

The loop looks like this:

```
for (int i = 0; i < materials.length; i++) {
    String name = materials[i].getName();
    double req = materials[i].getMaterialReq(aSurface);
    double price = materials[i].getTotalPrice(aSurface);
    System.out.print("Material name: " + name + ", Requirement: " +
                                              req + " units, price $" + price + ", ");
    if (materials[i] instanceof Flooring) {
      Flooring b = (Flooring) materials[i];
      System.out.println("width: " + b.getWidth());
    } else if (materials[i] instanceof Paint) {
      Paint m = (Paint) materials[i];
      System.out.println("no. of coats: " + m.getNoOfCoats()
                + ", no. of sqm/l: " + m.getNoOfSqMPerLiter());
    } else { // Wallpaper
      Wallpaper t = (Wallpaper) materials[i];
      System.out.println("length per roll: " + t.getLengthPerRoll()
                + ", width per roll: " + t.getWidthPerRoll());
    }
  }
```

Problem 2

toString() in the Material class:

```
public String toString() {
  return getClass().getName() + ": " + name + ", price: " + price;
}
```

toString() in the Paint class:

```
public String toString() {
  return super.toString() + ", no. of coats: " + noOfCoats +
                                                ", no. of sqm/l: " + noOfSqMPerLiter;
}
```

toString() in the Flooring class:

```
public String toString() {
  return super.toString() + ", width: " + widthOfFlooring + " m.";
}
```

toString() in the Wallpaper class:

```
public String toString() {
  return super.toString() + ", length per roll: " + lengthPerRoll +
                                                ", width per roll: " + widthPerRoll;
}
```

The revised loop from problem 1:

```
for (int i = 0; i < materials.length; i++) {
  String name = materials[i].getName();
  double req = materials[i].getMaterialReq(aSurface);
  double price = materials[i].getTotalPrice(aSurface);
  System.out.print("Material name: " + name + ", Requirement: "
      + req + " units, price $" + price + ", ");
  System.out.println(materials[i]);
}
```

The name of the material will now be printed twice.

Chapter 12.7

Problem 2

TestClass3 has two methods as members. method1() overrides the method with the same signature inherited from the TestClass1 class. In addition the TestClass3 has a method called method3().

The statement obj1.method3(); in main() results in a compilation error. The reason is that obj1 is an instance of TestClass1, and this class has no method with name method3().

Printout from the program:

```
Method1
Method 1 in TestClass3
Method 3 in TestClass3
Method 1 in TestClass3
Method 3 in TestClass3
```

Chapter 12.8

Problem 1

```
class Prob12_8_1 {
  public static void main(String[] args) {
    Material[] materials = new Material[3];
    FirstSortFlooring flooring1 = new FirstSortFlooring("FirstClassQuality", 20, 5);
    SecondSortFlooring flooring2A = new SecondSortFlooring("SecondClass1", 20, 5);
    SecondSortFlooring flooring2B = new SecondSortFlooring("SecondClass2", 20, 3);
    materials[0] = flooring1;
    materials[1] = flooring2A;
    materials[2] = flooring2B;
    Surface floor = new Surface("Floor in the dining room", 5, 4);
    for (int i = 0; i < materials.length; i++) {
      System.out.print("Flooring: " + materials[i].getName());
      System.out.println(", material requirement: "
        + materials[i].getMaterialReq(floor) + " m, price $" + materials[i].getTotalPrice(floor));
    }
  }
}

/* Printout:
Flooring: FirstClassQuality, material requirement: 4.0 m, price $80.0
Flooring: SecondClass1, material requirement: 4.8 m, price $48.0
Flooring: SecondClass2, material requirement: 9.6 m, price $57.60000000000001
*/
```

Problem 2

The toString() method in the Material class:

```
public String toString() {
  return getClass().getName() + ": " + name + ", price per unit: " + price;
}
```

The toString() method in the Flooring2 class:

```
public String toString() {
  return super.toString() + ", width: " + widthOfFlooring + " m.";
}
```

The toString() method in the FirstSortFlooring class:

```
public String toString() {
  return super.toString() + ", this is 1.class flooring.";
}
```

The toString() method in the SecondSortFlooring class:

```
public String toString() {
  return super.toString() + ", this is 2.class flooring.";
}
```

The loop body in the client program from problem 1 is expanded with:

```
/* Prints all data about the flooring, not only the name */
System.out.println("Flooring: " + materials[i]);  // toString() is implied
```

Chapter 12.10

Problem 1

The difference between "shallow copy" and "deep copy" becomes important if the object has instance variables of reference types (and not only of the primitive data types). Shallow copy means that the references are copied, that is, the original reference and the copy refer to the same object. Deep copy means that the objects are copied, too. In this case the original reference and the copied reference will refer to different (but equal) objects.

The Surface class as an example:

Three instance variables: name, length, width. The difference only effects name. With shallow copy, the two references will point to the same String object, with deep copy they each will have one String object.

Problem 2

The standard version of the clone() method performs a shallow copy. The method is protected in the Object class.

Problem 3

We try:

```
public static void main(String[] args) {
  Surface f1 = new Surface("wall", 3, 4);
  Surface f2 = (Surface) f1.clone();
  System.out.println(f2.getName());
}
```

If we put this main() method in its own class, we have no access to the inherited clone() method on behalf of the Surface object. This is because we do not have any access to protected members inherited from classes in other packages. See section 12.7. We get the compilation error: "clone() has protected access in java.lang.Object."[1]

If we put main() inside the Surface class, we get "unreported exception java.lang.CloneNotSupportedException; must be caught or declared to be thrown".

If we let main() throw the exception and let the Surface class implement the Clonable interface, the inherited version of clone() is used.

If we do not want the shallow copy, we have to implement our own version of the clone() method.

Problem 4

We create the SurfaceClonable class. It has a clone() method which makes a deep copy. In addition we create set methods to make it possible to test that we after the cloning really have two different copies. Extract from the SurfaceClonable class:

```
class SurfaceClonable implements Cloneable {
  ...instance variables
  ...constructor
  ...instance methods as before

  public java.lang.Object clone() {  // deep copy
    String nameCopy = new String(name);
    SurfaceClonable copy = new SurfaceClonable(nameCopy, length, width);
    return copy;
  }

  public void setName(String newName) {
    name = newName;
  }

  public void setLength(double newLength) {
    length = newLength;
  }

  public void setWidth(double newWidth) {
    width = newWidth;
  }
}
```

A little test program:

```
class Prob12_10_4 extends java.lang.Object {
  public static void main(String[] args) {
    SurfaceClonable f1 = new SurfaceClonable("wall", 3, 4);
    SurfaceClonable f2 = (SurfaceClonable) f1.clone();
    f1.setName("floor");
    f1.setLength(2);
```

1 This and the rest of this solution reflects a new behavior in version 1.4 of SDK.

```
    f1.setWidth(2.5);
    System.out.println("F1: " + f1.getName() + ", length: " +
                                                f1.getLength() + ", width: " + f1.getWidth());
    System.out.println("F2: " + f2.getName() + ", length: " +
                                                f2.getLength() + ", width: " + f2.getWidth());
  }
}

/* Example Run:
F1: floor, length: 2.0, width: 2.5
F2: wall, length: 3.0, width: 4.0
*/
```

Chapter 13.2

Problem 1

```
public class PushbuttonApplet extends JApplet {
  public void init() {
    Container guiContainer = getContentPane();
    LayoutManager layout = new FlowLayout();
    guiContainer.setLayout(layout);

    JButton myButton = new JButton("Push here!!");  // creates the button
    guiContainer.add(myButton);  // puts it in the container
    ButtonListener theButtonListener = new ButtonListener(); // creates a listener
    myButton.addActionListener(theButtonListener); // links the listener to the button

    /* Problem 1 */
    JButton myButton2 = new JButton("Or here!!");
    guiContainer.add(myButton2);
    myButton2.addActionListener(theButtonListener);

    /* Problem 2 */
    myButton.setMnemonic('P');
    myButton2.setMnemonic('O');

    /* Problem 3 */
    myButton2.setEnabled(false); // the user is not allowed to press myButton2
  }
}

class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {

    /* Problem 4, the println() statement is changed */
    String text = event.getActionCommand();
    System.out.println("You pushed the button with text: " + text);
  }
}
```

Chapter 13.3

Problem 1

Insert

```
nameField.setEditable(false);
```

at the end of the actionPerformed() method.

Problem 2

We have to count the number of pushes on each button. Both the buttons and the counters have to be instance variables.

```
private JLabel greeting = new JLabel();
private JButton buttonRed = new JButton("Red");
private JButton buttonBlue = new JButton("Blue");
private JButton buttonCyan = new JButton("Cyan");
private int pushRedCounter = 0;
private int pushBlueCounter = 0;
```

```
    private int pushCyanCounter = 0;
    private static final int maxNoOfPushes = 3;
```

Here is the new version of the actionPerformed() method:

```
    public void actionPerformed(ActionEvent event) {
       String colorName = event.getActionCommand();
       Color color;
       if (colorName.equals("Red")) {
         color = Color.red;
         pushRedCounter++;
         if (pushRedCounter > maxNoOfPushes) buttonRed.setEnabled(false);
       }
       else if (colorName.equals("Blue")) {
         color = Color.blue;
         pushBlueCounter++;
         if (pushBlueCounter > maxNoOfPushes) buttonBlue.setEnabled(false);
       }
       else { // cyan
         color = Color.cyan;
         pushCyanCounter++;
         if (pushCyanCounter > maxNoOfPushes) buttonCyan.setEnabled(false);
       }
       greeting.setForeground(color);
       String name = nameField.getText();
       greeting.setText("Hallo, " + name + "!");
     }
```

Problem 3

```
    import java.applet.*;
    import java.awt.*;
    import java.awt.event.*;
    import javax.swing.*;

    public class NameApplet3Buttons extends JApplet {
      private Container guiContainer;
      private JTextField nameField = new JTextField(20);
      private JLabel greeting = new JLabel();
      private JLabel nameLabel = new JLabel("What's your name?");
      private Font sansSerif;
      private Font monoSpaced;
      private Font dialog;
      private JButton buttonSansSerif = new JButton("SansSerif");
      private JButton buttonMonoSpaced = new JButton("MonoSpaced");
      private JButton buttonDialog = new JButton("Dialog");

      public void init() {
        guiContainer = getContentPane();
        guiContainer.setLayout(new FlowLayout());

        guiContainer.add(nameLabel);
        guiContainer.add(nameField);

        /* Creates three different fonts */
        Font defaultFont = guiContainer.getFont();
        int style = defaultFont.getStyle();
        int size = defaultFont.getSize();
        sansSerif = new Font("SansSerif", style, size);
        monoSpaced = new Font("Serif", style, size);
        dialog = new Font("Dialog", style, size);

        buttonSansSerif.setFont(sansSerif);
        guiContainer.add(buttonSansSerif);

        buttonMonoSpaced.setFont(monoSpaced);
        guiContainer.add(buttonMonoSpaced);

        buttonDialog.setFont(dialog);
        guiContainer.add(buttonDialog);
```

```
    /*
     * Creates a listener and links all the three
     * buttons to this listener.
     */
    ButtonListener theButtonListener = new ButtonListener();
    buttonSansSerif.addActionListener(theButtonListener);
    buttonMonoSpaced.addActionListener(theButtonListener);
    buttonDialog.addActionListener(theButtonListener);

    guiContainer.add(greeting);
  }

  private class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {

      /* The font is changed in stead of the color */
      String font = event.getActionCommand();
      Font newFont;
      if (font.equals("SansSerif")) newFont = sansSerif;
      else if (font.equals("MonoSpaced")) newFont = monoSpaced;
      else newFont = dialog;
      greeting.setFont(newFont);
      nameField.setFont(newFont);
      nameLabel.setFont(newFont);

      String name = nameField.getText();
      greeting.setText("Hallo, " + name + "!");
    }
  }
}
```

Problem 4

A user press at the Enter key in a text box generates an ActionEvent. We create a listener class to handle this:

```
private class nameFieldListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    String name = nameField.getText();
    greeting.setText("Hallo, " + name + "!");
    buttonSansSerif.requestFocus();
  }
}
```

The listener is registered:

```
nameField.addActionListener(new nameFieldListener());
```

Chapter 13.4

Problem 1

The most important changes are in the actionPerformed() method in the ButtonListener class. The section below the comment "Gets the color of the button" is as follows:

```
JButton buttonPushed = (JButton) event.getSource();
Color color;
if (buttonPushed == redButton) color = Color.red;
else if (buttonPushed == blueButton) color = Color.blue;
else color = Color.cyan;
```

We refer to the button objects. These cannot longer be locale variables in the constructor; they have to be instance variables in the outer class DrawApplet. Then they can be accessed from all methods and classes declared inside the DrawApplet class. The statements

```
JButton redButton = new JButton("Red");
JButton blueButton = new JButton("Blue");
JButton cyanButton = new JButton("Cyan");
```

are moved out of the ButtonPanel constructor and inserted *before* the init() method in the outer class. And they are made private:

```
private JButton redButton = new JButton("Red");
private JButton blueButton = new JButton("Blue");
private JButton cyanButton = new JButton("Cyan");
```

Problem 2

Parallel arrays with colors and their names are declared as named constants in the DrawApplet class:

```
private static final String[] nameOfAllColors =
   {"Black", "Blue", "Cyan", "Dark grey", "Grey", "Green",
    "Light grey", "Magenta", "Orange", "Pink", "Red", "White", "Yellow"};
private static final Color[] allColors =
 {Color.black, Color.blue, Color.cyan, Color.darkGray,
  Color.gray, Color.green, Color.lightGray, Color.magenta, Color.orange,
 Color.pink, Color.red, Color.white, Color.yellow};
```

The section below the comment "Gets the color of the button" is now as follows:

```
String colorName = event.getActionCommand();
Color color = Color.black;
int colorIndex = 0;
while (colorIndex < nameOfAllColors.length &&
  !nameOfAllColors[colorIndex].equals(colorName)) colorIndex++;
if (colorIndex < nameOfAllColors.length) color = allColors[colorIndex];
else System.out.println("Error in the ButtonListener class.");
```

We gets the color name, and then we search through the nameOfAllColors array to find the index of that color. From the allColors array we find the color object with the same index.

The new version of southern panel's constructor is as follows:

```
public ButtonPanel() {
  ButtonListener theButtonListener = new ButtonListener();
  setLayout(new GridLayout(3, 5, 5, 5)); // Three rows, five buttons each
  for (int buttonNo = 0; buttonNo < nameOfAllColors.length; buttonNo++) {
    JButton myButton = new JButton(nameOfAllColors[buttonNo]);
    myButton.setBackground(allColors[buttonNo]);
    myButton.addActionListener(theButtonListener);
    add(myButton);
  }
}
```

The buttons are instantiated in the loop.

Chapter 14.1

Problem 1

Insert the statement textArea.selectAll(); after the statement textArea.setEditable(true); in the focusLost() method in the PasswordListener class.

Chapter 14.3

Problem 1

We have to do several insertions in the program:

- New instance variable in the CheckboxApplet class:

  ```
  private JCheckBox breakfast = new JCheckBox("Breakfast");
  ```

- The instance variable has to be added to the selection panel:

  ```
  add(breakfast); // in the constructor SelectionPanel()
  ```

- The listener object has to be linked to this check box.

  ```
  breakfast.addActionListener(listener); // in the constructor SelectionPanel()
  ```

- And printout to the console:

  ```
  if (breakfast.isSelected()) { // in the actionPerformed() method
    System.out.println("Will have breakfast.");
  }
  ```

Problem 2

A textarea will be inserted below the group box.

- The actionPerformed() method in the CheckBoxListener class must have access to this area. Therefore it has to be an instance variable:

```
private JTextArea output = new JTextArea(3, 30);
```

- We insert the text area in the south of the guiContainer (in place of an empty panel):

```
guiContainer.add(output, BorderLayout.SOUTH);  // in the init() method
```

- The new version of the actionPerformed() method:

```
public void actionPerformed(ActionEvent event) {
  String text = "";
  if (dinner.isSelected()) text += "Will have dinner.";
  if (lunch.isSelected()) text += "Will have lunch.";
  output.setText(text);
}
```

Chapter 14.4

Problem 1

The selection panel will be divided in two parts, one for sex and one for residence. We have two listener classes, one for sex and one for residence. (Or we could have used only one listener class.) Here is the program:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class Prob14_4_1 extends JApplet {

  private JRadioButton female = new JRadioButton("Female", true);
  private JRadioButton male = new JRadioButton("Male", false);
  private JRadioButton apartment = new JRadioButton("Apartment", true);
  private JRadioButton condo = new JRadioButton("Condo", false);
  private JRadioButton townhouse = new JRadioButton("Townhouse", false);
  private JRadioButton house = new JRadioButton("House", false);

  public void init() {
    Container guiContainer = getContentPane();
    guiContainer.add(new JPanel(), BorderLayout.NORTH);
    SelectionPanel middle = new SelectionPanel();
    guiContainer.add(middle, BorderLayout.CENTER);
    guiContainer.add(new JPanel(), BorderLayout.SOUTH);
  }

  /* Describes the panel with two group boxes for selections */
  private class SelectionPanel extends JPanel {
    public SelectionPanel() {
      add(new SexPanel());
      add(new ResidencePanel());
    }
  }

  private class SexPanel extends JPanel {
    public SexPanel() {
      ButtonGroup group = new ButtonGroup();
      group.add(female);
      group.add(male);
      add(female);
      add(male);
      listenerSex listener = new listenerSex();
      female.addActionListener(listener);
      male.addActionListener(listener);
      SoftBevelBorder border = new SoftBevelBorder(BevelBorder.RAISED);
```

```
    Border groupBox = BorderFactory.createTitledBorder(border, "Sex");
    setBorder(groupBox);
  }
 }

 private class ResidencePanel extends JPanel {
  public ResidencePanel() {
    ButtonGroup groupResidence = new ButtonGroup();
    groupResidence.add(apartment);
    groupResidence.add(condo);
    groupResidence.add(townhouse);
    groupResidence.add(house);
    add(apartment);
    add(condo);
    add(townhouse);
    add(house);
    RadioButtonListenerResidence listenerResidence =
                                    new RadioButtonListenerResidence();
    apartment.addActionListener(listenerResidence);
    condo.addActionListener(listenerResidence);
    townhouse.addActionListener(listenerResidence);
    house.addActionListener(listenerResidence);
    SoftBevelBorder border = new SoftBevelBorder(BevelBorder.RAISED);
    Border groupBox = BorderFactory.createTitledBorder(border, "Type of Residence");
    setBorder(groupBox);
  }
 }

 /* Describes listeners for the radio buttons in the sex group box */
 private class listenerSex implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    String kjønn = event.getActionCommand();
    if (kjønn.equals("Female")) System.out.println("Female is selected");
    else System.out.println("Male is selected");
  }
 }

 /* Describes listeners for the radio buttons in the residence group box */
 private class RadioButtonListenerResidence implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    String bolig = event.getActionCommand();
    if (bolig.equals("Apartment")) System.out.println("Apartment is selected");
    else if (bolig.equals("Condo")) System.out.println("Condo is selected");
    else if (bolig.equals("Townhouse")) System.out.println("Townhouse is selected");
    else System.out.println("House is selected");
  }
 }
}
```

Problem 2

The push button with a listener is inserted into the init() method:

```
JButton button = new JButton("Register my choice");
guiContainer.add(button, BorderLayout.SOUTH); // the empty panel in south is removed
button.addActionListener(new ButtonListener());
```

The radio buttons will have no listeners. Remove the three statements after add(male); in the SelectionPanel() constructor.

The new listener class (the name of the class is ButtonListener in stead of RadioButtonListener):

```
private class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    String sex = event.getActionCommand();
    if (female.isSelected()) System.out.println("Female is selected");
    else System.out.println("Male is selected");
  }
}
```

Chapter 14.5

Problem 1

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class Prob14_5_1 extends JApplet {

  static final String [] cities =
     {"Tokyo", "New York City", "Mexico City", "Mumbai", "Sao Paulo", "Los Angeles",
     "Shanghai", "Lagos", "Calcutta", "Buenos Aires"};
  static final double[] populations = {28, 20.1, 18.1, 18, 17.7, 15.8,
    14.2, 13.5, 12.9, 12.5};
 private JList cityList = new JList(cities);

  public void init() {
    Container guiContainer = getContentPane();
    JLabel label = new JLabel("Select a city:");
    guiContainer.add(label, BorderLayout.NORTH);
    JScrollPane scrollPaneWithList = new JScrollPane(cityList);
    guiContainer.add(scrollPaneWithList, BorderLayout.CENTER);
    ListListener listener = new ListListener();
    cityList.addListSelectionListener(listener);
    cityList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    guiContainer.add(new JPanel(), BorderLayout.SOUTH);
  }

  class ListListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent event) {
      int index = cityList.getSelectedIndex();
      /*
       * A choise in a list generates several events. Every event will generate
       * a dialog box. Several events generate several boxes on top of each
       * other. To avoid this, we use getValueIsAdjusting().
       */
      if (!cityList.getValueIsAdjusting() && index >= 0) {
        JOptionPane.showMessageDialog(null, "The population of " + cities[index] +
            " is " + populations[index] + " million.");
        cityList.clearSelection();
      }
    }
  }
}
```

Problem 2

"New city" is a push button below the list. The city names are stored in an instance of the DefaultListModel class, while the populations are stored in an array list.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import java.text.*;

public class Prob14_5_2 extends JApplet {
  static final String [] cities =
       {"Tokyo", "New York City", "Mexico City", "Mumbai", "Sao Paulo",
         "Los Angeles",  "Shanghai", "Lagos", "Calcutta", "Buenos Aires"};
  static final double[] populations = {28, 20.1, 18.1, 18, 17.7, 15.8,
    14.2, 13.5, 12.9, 12.5};

  private DefaultListModel data = new DefaultListModel();
  private JList citylist = new JList(data);  // NB!
```

```
  private ArrayList thePopulations = new ArrayList();

  public void init() {
    for (int i = 0; i < cities.length; i++) data.addElement(cities[i]);
    for (int i = 0; i < populations.length; i++) {
      Double populationObject = new Double(populations[i]);
      thePopulations.add(populationObject);
    }

    Container guiContainer = getContentPane();
    JLabel label = new JLabel("Select a city:");
    guiContainer.add(label, BorderLayout.NORTH);
    JScrollPane scrollPaneWithList = new JScrollPane(citylist);
    guiContainer.add(scrollPaneWithList, BorderLayout.CENTER);
    ListListener listener = new ListListener();
    citylist.addListSelectionListener(listener);
    citylist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    JButton newCityButton = new JButton("New city");
    guiContainer.add(newCityButton, BorderLayout.SOUTH);
    newCityButton.addActionListener(new NewCityListener());
  }

  class ListListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent event) {
      int index = citylist.getSelectedIndex();
      if (!citylist.getValueIsAdjusting() && index >= 0) {
        JOptionPane.showMessageDialog(null, "The population of " + data.get(index) +
            " is " + thePopulations.get(index) + " million.");
        citylist.clearSelection();
      }
    }
  }

  class NewCityListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      String newCity = JOptionPane.showInputDialog(null, "New city: ");
      data.addElement(newCity);
      thePopulations.add(inputNewPopulation());
    }

    /* Help method: Inputs the number. Asks again if it is not a positive number. */
    private Double inputNewPopulation() {
      boolean okNumber = true;
      Double newPopulation = new Double(0);
      do {
        String asText = JOptionPane.showInputDialog(null, "Population (in million): ");
        try {
          okNumber = true;
          newPopulation = new Double(asText);
          if (newPopulation.doubleValue() <= 0) throw new NumberFormatException("");
        } catch (NumberFormatException e) {
          okNumber = false;
          JOptionPane.showMessageDialog(null,
            "Invalid number. Should be a positive number. Try again!");
        }
      } while (!okNumber);
      return newPopulation;
    }
  }
}
```

Chapter 14.7

Problem 1

We have handled three listener interfaces:

- The java.awt.event.ActionListener interface contains only one method, and it is therefore easy to use the interface as it is. It has no adapter class.
- The java.awt.event.FocusListener interface contains two methods. It has an adapter class, the FocusAdapter class.
- The javax.swing.event.ListSelectionListener interface contains only one method. It has no adapter class.

We change the TextApplet from program listing 14.1 to try the FocusAdapter class:

```
/* Only one line have to be changed */
private class PasswordListener extends FocusAdapter {
   public void focusLost(FocusEvent event) {
     ...as before...
   }
 }
 /* The focusGained() method is empty and may be removed,
    because an empty version is inherited from FocusAdapter */
```

Problem 2

```
import javax.swing.JFrame;
class JFrame3 extends JFrame {
  public JFrame3(String title, int windowWidth, int windowHeight, int posX, int posY) {
    super(title);
    setSize(windowWidth, windowHeight);
    setLocation(posX, posY);
  }
}

class Prob14_7_2 {
  public static void main(String[] args) {
    JFrame3 window = new JFrame3("A JFrame3 test", 400, 300, 100, 200);
    window.setVisible(true);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Problem 3

The first part of the JFrame3 class has to be changed:

```
package myLibrary;
import javax.swing.JFrame;
public class JFrame3 extends JFrame {
```

Save the file in the myLibrary directory, and compile it.

The class may be used in the following way:

```
import myLibrary.JFrame3;
class Prob14_7_3 {
  public static void main(String[] args) {
    JFrame3 window =
      new JFrame3("A JFrame3 test", 400, 300, 100, 200);
    window.setVisible(true);
    window.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
  }
}
```

Chapter 14.8

Problem 1

The ListApplet in program listing 14.4 would be changed into an application. The class is no longer public, and it is a subclass of JFrame, not of JApplet:

```
class ListApplication extends JFrame {
```

The head of the init() method is changed into a constructor head, and the title is passed to the constructor's superclass:

```
  public ListApplication(String title) {
    super(title);
```

Finally, we have to make a class with main():

```
class Prob14_8_1 {
  public static void main(String[] args) {
    ListApplication window = new ListApplication("City List");
    window.pack();
    window.setVisible(true);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Chapter 15.1

Problem 1

```
class WindowWithMenu extends JFrame {
  private static final String[] allColorsName = {"Black", "Blue", "Cyan", "Dark Gray",
      "Gray", "Green", "Light Gray", "Magenta", "Orange", "Pink", "Red",
      "White", "Yellow"};
  private static final Color[] allColors = {Color.black, Color.blue,
  Color.cyan, Color.darkGray, Color.gray, Color.green, Color.lightGray, Color.magenta,
      Color.orange, Color.pink, Color.red, Color.white, Color.yellow};
  private Container guiContainer;

  public WindowWithMenu(){
    super("Test Menu");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    guiContainer = getContentPane();
    MenuListener theListener = new MenuListener();
    JMenu theMenu = new JMenu("Color");
    JMenuItem[] menuItems = new JMenuItem[allColorsName.length];
    for (int i = 0; i < menuItems.length; i++) {
      JMenuItem menypost = new JMenuItem(allColorsName[i]);
      theMenu.add(menypost);
      menypost.addActionListener(theListener);
    }
    JMenuBar menuLine = new JMenuBar();
    menuLine.add(theMenu);
    setJMenuBar(menuLine);
  }

  private class MenuListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      String command = event.getActionCommand();
      int colorIndex = 0;
      while (colorIndex < allColorsName.length
        && !command.equals(allColorsName[colorIndex]))
        colorIndex++;
      if (colorIndex < allColorsName.length)
      guiContainer.setBackground(allColors[colorIndex]);
      else System.out.println("Error in actionPerformed()"); // should not happen
    }
  }
}
```

Problem 2

The WindowWithMenu() constructor has to be expanded with:

```
HelpListener theHelpListener = new HelpListener();
JMenu helpMenu = new JMenu("Help");
JMenuItem menuItem2 = new JMenuItem("Help");
menuItem2.addActionListener(theHelpListener);
helpMenu.add(menuItem2);
menuItem2 = new JMenuItem("About the program...");
menuItem2.addActionListener(theHelpListener);
helpMenu.add(menuItem2);
menuLine.add(helpMenu);
```

A new listener class:

```
private class HelpListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    String command = event.getActionCommand();
    if (command.equals("Help")) {
      JOptionPane.showMessageDialog(null,
                          "Drop down the Color Menu and select background color!");
    } else {
      JOptionPane.showMessageDialog(null,
                                  "This program is written January 15., 2002 by Else Lervik.");
    }
  }
}
```

Chapter 15.2

Problem 1

Insert the statement toolbar.addSeparator(); after e.g. the toolbar.add(yellowButton); statement. Then you will get a wider space between the yellow and the red button than between the red and the blue button.

Problem 2

You may attach a tool tip to a button in the following way:

```
redButton.setToolTipText("The background color becomes red");
```

If the user keeps the mouse over the button for a while, a yellow label with the text will be displayed.

Problem 3

The yellow button:

```
yellowButton = new JButton("Yellow");
yellowButton.setMnemonic('Y');
```

The other buttons are handled in the same way.

Chapter 15.3

Problem 1

Changes in the MiniDialog class:

A new instance variable:

```
private JTextField textField = new JTextField(20);
```

A new inner class to input the text:

```
private class TextPanel extends JPanel {
  public TextPanel() {
    add(new JLabel("Enter a text: "));
    add(textField);
  }
}
```

The constructor is changed a bit:

```
public MiniDialog(JFrame parent) {
  super(parent, "Minidialog", true);
  guiContainer = getContentPane();
  guiContainer.add(new JLabel(), BorderLayout.NORTH);
  guiContainer.add(new TextPanel(), BorderLayout.CENTER);
  guiContainer.add(new ButtonPanel(), BorderLayout.SOUTH);
  pack();
}
```

The revised showDialog():

```
public String showDialog() {
  setVisible(true);
  if (ok) return textField.getText();
  else return null;
}
```

Most of the ParentWindow class has to be changed. We use GridLayout as layout manager. The button and the text box are placed in the central column, while yellow buttons are placed in the right and the left side columns. A yellow button is a specialization of a button:

```
class YellowButton extends JButton {
  public YellowButton() {
    setBackground(Color.yellow);
  }
}
```

Here is the ParentWindow class:

```
class ParentWindow extends JFrame {
  private MiniDialog dialogBox = new MiniDialog(this);
  private Container guiContainer;
  private JLabel theText = new JLabel("", JLabel.CENTER);

  public ParentWindow() { // Problem 1
    setTitle("Testing Dialogs");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    guiContainer = getContentPane();
    guiContainer.setLayout(new GridLayout(2, 3));
    guiContainer.add(new YellowButton());
    JButton button = new JButton("Push here!");
    button.setBackground(Color.darkGray);
    button.setForeground(Color.black);
    guiContainer.add(button);
    ButtonListenerParent buttonListener = new ButtonListenerParent();
    button.addActionListener(buttonListener);
    guiContainer.add(new YellowButton());
    guiContainer.add(new YellowButton());
    theText.setBackground(Color.darkGray);
    theText.setForeground(Color.black);
    guiContainer.add(theText);
    guiContainer.add(new YellowButton());
    pack();
  }

  private class ButtonListenerParent implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      String text = dialogBox.showDialog();
      if (text != null) theText.setText(text);
      else theText.setText("Nothing entered");
    }
  }
}
```

Chapter 15.6

Problem 1

The Surface class is not affected. The other three dialog classes get a common superclass called MaterialDialog:

```
abstract class MaterialDialog extends MyDialog implements Constants {
  private JTextField nameField = new JTextField(textFieldLength);
  private JTextField priceField = new JTextField(textFieldLength);
  private double price = 0.0;

  public MaterialDialog(JFrame parent, String title) {
    super(parent, title);
    Container guiContainer = getContentPane();
    guiContainer.add(new JPanel(), BorderLayout.NORTH);
    guiContainer.add(getButtonPanel(), BorderLayout.SOUTH);
  }

  protected double getPrice() {
    return price;
  }
```

```
  protected String getMaterialName() {
    return nameField.getText();
  }

  /*
   * Every dialog class will have its own subclass of the DataPanel class.
   * This shows an example where an inner class may well have some
   * other access than private.
   */
  protected class DataPanel extends JPanel {
    public DataPanel(String unit, int noOfLines) {
      setLayout(new GridLayout(noOfLines,2));
      add(new JLabel("Name:", JLabel.RIGHT));
      add(nameField);
      add(new JLabel("Price per " + unit + " :", JLabel.RIGHT));
      add(priceField);
    }
  }

  /*
   * The okData() method is not declared in the subclasses. Instead they
   * have their own version of the parseRemainingNumbers() method. That method
   * is abstract here, se below.
   */
  final protected boolean okData() {
    try {
      price = numberFormat.parsePositivDouble(priceField.getText());
      parseRemainingNumbers();
    } catch (ParseException e) {
      JOptionPane.showMessageDialog(null,
                    "Number input could not be converted, Try again!");
      priceField.requestFocus();
      return false;
    }
    return true;
  }

  /*
   * Every subclass has its own showDialog(), which empties the fields in that class.
   * After that, the showDialog() here is called (by using super.showDialog()).
   */
  public Material showDialog() {
    nameField.setText("");
    priceField.setText("");
    setOk(false);
    setVisible(true);
    nameField.requestFocus();

    /* Every subclass has its own instantiateNewMaterial() */
    if (isOk()) return instantiateNewMaterial();
    else return null;
  }

  /* We have two abstract methods. They are commented above. */
  abstract protected void parseRemainingNumbers() throws ParseException;
  abstract protected Material instantiateNewMaterial();
}
```

The PaintDialog is shown as an example of a subclass:

```
class PaintDialog extends MaterialDialog implements Constants {
  private JTextField noOfCoatsField = new JTextField(textFieldLength);
  private JTextField noOfSqmPerLiterField = new JTextField(textFieldLength);
  private int noOfCoats;
  private double noOfLiters;

  public PaintDialog(JFrame parent) {
```

```
    super(parent, "Paint");
    Container guiContainer = getContentPane();
    guiContainer.add(new PaintDataPanel(), BorderLayout.CENTER);
    pack();
  }

  private class PaintDataPanel extends DataPanel {
    public PaintDataPanel() {
      super("liter", 4);
      add(new JLabel("No. of Coats: ", JLabel.RIGHT));
      add(noOfCoatsField);
      add(new JLabel("No. of sqm per Liter: ", JLabel.RIGHT));
      add(noOfSqmPerLiterField);
    }
  }

  protected void parseRemainingNumbers() throws ParseException {
    noOfCoats = numberFormat.parsePositivInteger(noOfCoatsField.getText());
    noOfLiters = numberFormat.parsePositivDouble(noOfSqmPerLiterField.getText());
  }

  protected Material instantiateNewMaterial() {
    return new Paint(getMaterialName(), getPrice(), noOfCoats, noOfLiters);
  }

  public Material showDialog() {
    noOfCoatsField.setText("");
    noOfSqmPerLiterField.setText("");
    return super.showDialog();
  }
}
```

We have to do a very little change in the ProjectChap15 class. The showDialog() method is called in the private class ButtonListener. It now has Material as return type.

Either we cast to the correct type:

```
case 1:  // new paint
  Paint newPaint = (Paint) thePaintDialog.showDialog();
  recordMaterial(newPaint);
  break;
```

or we let the locale variable which receives the return value have the type Material:

```
case 1:  // new paint
  Material newPaint = thePaintDialog.showDialog();
  recordMaterial(newPaint);
  break;
```

Both variants will work in the original program, too.

Chapter 16.3

Problem 1

```
class Printer implements Runnable {
  private int numberToPrint;

  public Printer(int number) {
    numberToPrint = number;
  }

  public void run() {
    while (true) {
      System.out.print(numberToPrint);
      System.out.print(" ");
    }
  }
}

class ShowerOfNumbers {
```

```
  public static void main(String[] args) {
    Thread printer1 = new Thread(new Printer(1));
    Thread printer2 = new Thread(new Printer(2));
    Thread printer3 = new Thread(new Printer(3));
    Thread printer4 = new Thread(new Printer(4));
    Thread printer5 = new Thread(new Printer(5));
    printer1.start();
    printer2.start();
    printer3.start();
    printer4.start();
    printer5.start();
  }
}
```

Chapter 16.6

Problem 1

```
class SequencePrinter {

  public void printSequence() {
    synchronized (this) {
      System.out.print("1 ");
      System.out.print("2 ");
      System.out.print("3 ");
      System.out.print("4 ");
      System.out.print("5 ");
    }
  }
}

class NumberPrinter extends Thread {
  private SequencePrinter thePrinter;

  public NumberPrinter(SequencePrinter printer) {
    thePrinter = printer;
  }

  public void run() {
    while (true) {
      thePrinter.printSequence();
    }
  }
}

class ShowerOfNumbersSync {
  public static void main(String[] args) {
    SequencePrinter aPrinter = new SequencePrinter();
    NumberPrinter printer1 = new NumberPrinter(aPrinter);
    NumberPrinter printer2 = new NumberPrinter(aPrinter);
    NumberPrinter printer3 = new NumberPrinter(aPrinter);
    NumberPrinter printer4 = new NumberPrinter(aPrinter);
    NumberPrinter printer5 = new NumberPrinter(aPrinter);
    printer1.start();
    printer2.start();
    printer3.start();
    printer4.start();
    printer5.start();
  }
}
```

Chapter 17.2

Problem 3

The element class becomes like this:

```
class NameElement {
  private NameElement next, previous;
  private String name;
```

```
  public NameElement(String startName) {
    name = startName;
    next = null;
    previous = null;
  }

  public NameElement(String startName, NameElement startNext,
                                                    NameElement startPrevious) {
    name = startName;
    next = startNext;
    previous = startPrevious;
  }

  public String getName() {
    return name;
  }

  public NameElement getNext() {
    return next;
  }

  public NameElement getPrevious() {
    return previous;
  }

  public void setName(String newName) {
    name = newName;
  }

  public void setNext(NameElement newNext) {
    next = newNext;
  }

  public void setPrevious(NameElement newPrevious) {
    previous = newPrevious;
  }
}
```

And the new NameList becomes like this:

```
class NameList {
  private NameElement firstElement;

  /* Returns the contents of the list in textual form. */
  public String toString() {
    String listText = "";
    NameElement auxReference = firstElement;
    while (auxReference != null) {
      listText = listText + auxReference.getName() + "\n";
      auxReference = auxReference.getNext();
    }
    return listText;
  }

  public void insertNameAtEnd(String newName) {
    NameElement auxReference = firstElement;
    NameElement auxReferenceRightBehind = null;
    /* We start by moving to the end of the list. */
    while (auxReference != null) {
      auxReferenceRightBehind = auxReference;
      auxReference = auxReference.getNext();
    }
    if (auxReferenceRightBehind == null ) {
      /* This happens if the list was empty from before.  */
      firstElement = new NameElement(newName);
    } else {
       /*
```

```
        * This happens if there were at least one element in the list
        * from before. auxReferenceRightBehind is now refering to
        * the last element.
        */
       auxReferenceRightBehind.setNext(new NameElement(newName));
       /* Then we point the previous of the newly inserted element backwards.*/
       auxReferenceRightBehind.getNext().setPrevious(auxReferenceRightBehind);
     }
   }

   public boolean findName(String textToFind) {
     NameElement auxReference = firstElement;
     while (auxReference != null) {
       if (auxReference.getName().equals(textToFind)) return true;
       auxReference = auxReference.getNext();
     }
     return false;
   }

     /*
      * This method deletes all occurences of the given text from the
      * list.
      */
   public void deleteName(String nameToDelete) {
     NameElement auxReference = firstElement;
     NameElement auxReferenceRightBehind = null;

     /* We start by iterating through the list. */
     while (auxReference != null) {
       if (auxReference.getName().equals(nameToDelete)) {
           /*
             * auxReference is refering to an element to be removed. We
            * need a special case if this is the first element in the list.
            */
           if (auxReferenceRightBehind != null) {
             auxReferenceRightBehind.setNext(auxReference.getNext());
           } else {
             firstElement = auxReference.getNext();
           }
           auxReference = auxReference.getNext();
           /* If the element we have removed wasn't the last one,
            * the next one has to get its previous updated. It points to the one
             * to be deleted. (We recommend the pencil problems!)
            */
           if (auxReference != null) {
             auxReference.setPrevious(auxReferenceRightBehind);
           }
       } else {
           auxReferenceRightBehind = auxReference;
           auxReference = auxReference.getNext();
       }
     }
   }
}
```

The test client remains unchanged. (Encapsulation!)

Chapter 17.3

Problem 1

```
import java.util.*;

class NameListTestLinkedList {

  public static void main(String[] args) {
    LinkedList aList = new LinkedList();
    System.out.println("Printout of empty list: ");
    System.out.println(aList.toString());
```

```
    aList.add("Larry");
    aList.add("Winston");
    System.out.println("Printout of two men: ");
    /* The printout differs a bit, it's prettier, because the toString() of LinkedList is
     * different to ours.
     */
    System.out.println(aList.toString());
    System.out.println("Is Bucky in the list: " + aList.contains("Bucky"));
    System.out.println("Is Larry in the list: " + aList.contains("Larry"));
    System.out.println("Is Winston in the list: " + aList.contains("Winston"));
    System.out.println("Inserts an extra Larry...");
    aList.add("Larry");
    System.out.println("Printout with two Larries: ");
    System.out.println(aList.toString());
    System.out.println("Deleting Larry and Winston...");
    while (aList.contains("Larry")) {
      aList.remove("Larry");
    }
    while (aList.contains("Winston")) {
      aList.remove("Winston");
    }
    System.out.println("Printout of empty list: ");
    System.out.println(aList.toString());
  }
}

/* Printout:
Printout of empty list:
[]
Printout of two men:
[Larry, Winston]
Is Bucky in the list: false
Is Larry in the list: true
Is Winston in the list: true
Inserts an extra Larry...
Printout with two Larries:
[Larry, Winston, Larry]
Deleting Larry and Winston...
Printout of empty list:
[]
*/
```

Chapter 17.4

Problem 1

We imagine we first deal with part above the division line. First, the multiplication 5x89 is done, and the result is placed on the stack. Then 9-3, the result is stored on the stack. Then an addition is performed on the upper two numbers on the stack. The part below the division line is calculated in the same way, and finally, a division is performed with the upper two elements on the stack.

Chapter 17.5

Problem 2

```
class Palindrome {

  public boolean isPalindrome(String text) {
    text = text.toLowerCase();
    text = text.trim();
    if (text.equals("")) return true;
    if (text.length() == 1) return true;
    if (text.charAt(0) != text.charAt(text.length()-1)) return false;
    /* If not: */
    return isPalindrome(text.substring(1, text.length()-1));
  }

  public static void main(String[] args) {
    /* This construct is uncommon in the book, but we instantiate the same class that
```

```
     * main() is in. We could also have made the method static.
     */
    Palindrome myPalindrome = new Palindrome();
    System.out.print("Is Summer vacation a palindrome: "
                                + myPalindrome.isPalindrome("Summer vacation") + "\n");
    System.out.print("Is A man a plan a canal Panama a palindrome: " +
                                myPalindrome.isPalindrome("A man a plan a canal Panama") + "\n");

  }
}
/* Printout:
Is Summer vacation a palindrome: false
Is A man a plan a canal Panama a palindrome: true
*/
```

Chapter 17.7

Problem 1

```
import java.util.*;

class BinarySearchTreeTreeMap {
  private TreeMap tree = new TreeMap();

  public String toString() {
    String resultText = "";
    TreeMap auxTree = new TreeMap(tree);
    Integer numberOut = (Integer)auxTree.firstKey();
    while (numberOut != null) {
      resultText = resultText + numberOut.toString() + " ";
      try {
          auxTree.remove(numberOut);
          numberOut = (Integer)auxTree.firstKey();
      }
      catch (NoSuchElementException e) {
          numberOut = null;
      }
    }
    return resultText;
  }

  public void insertValue(int value) {
    /* Both key and data is the same number.*/
    tree.put(new Integer(value), new Integer(value));
  }

  /* Returns true if the value is in the tree. */
  public boolean searchForValue(int valueToFind) {
    /* Remember that equal() is used, we only need another equal object to search
     * with.
     */
    if (tree.get(new Integer(valueToFind)) != null) return true;
    return false;
  }
}
```

Also here can we use the same test client.

Chapter 18.3

Problem 1

This will depend on the browser. In Netscape 4.7 for example, the applet will be finalized, loaded and started again, that is, the browser calls destroy(), then init() and start().

Chapter 18.5

Problem 1

The program becomes like this:

```
import java.awt.*;
import javax.swing.*;

public class ParameterAdding extends JApplet {
  private Container contents = getContentPane();
  private String number1;
  private String number2;

  public void init() {
    number1 = getParameter("number1");
    number2 = getParameter("number2");
    contents.add(new Drawing(number1, number2));
  }
}

/*
 * This JPanel is told the parameters the applet has got, through the
 * constructor.
 */
class Drawing extends JPanel {
  private String number1, number2;
  private int sum = 0;

  public Drawing(String appletsNumber1, String appletsNumber2) {
    number1 = appletsNumber1;
    number2 = appletsNumber2;
    sum = Integer.parseInt(number1) + Integer.parseInt(number2);
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString("Number 1 that was sent to this applet is: " + number1, 5, 50);
    g.drawString("Number 1 that was sent to this applet is: " + number2, 5, 65);
    g.drawString("The sum is: " + sum , 5, 80);
  }
}
```

And the HTML file might look like this:

```
<html>
<head>
</head>
<body>

<h1>Applet which demonstrates use of parameters</h1>

<applet code="ParameterAdding.class" width="500" height="100">
<param name="number1" value="6">
<param name="number2" value="7">

Your browser doesn't support applets, or it's turned off.
</applet>

</body>
</html>
```

Chapter 19.1

Problem 1

On the server side:

```
/* Sends introductory text to the client side program */
toClientWriter.println("***Hello, you are connected to the server!");
toClientWriter.println("Enter a number, one on each line, terminate with 'x'");

/* Receives data fom the client side */
int sum = 0;
String aLine = fromClientReader.readLine();
while (!aLine.equals("x")) {  // the last line is equal to 'x'
```

```
    sum += Integer.parseInt(aLine);  // converts the text to a number, no exception handling
    aLine = fromClientReader.readLine();
  }
  toClientWriter.println(
              "The sum of the numbers is " + sum); // sends the sum to the client side
  connection.close(); // shuts down
```

On the client side:

```
  /* Receives the introductory text from the server, and prints it to the console */
  String intro1 = fromServerReader.readLine();
  String intro2 = fromServerReader.readLine();
  System.out.println(intro1 + "\n" + intro2);

  String text = fromConsoleReader.readLine();
  while (!text.equals("x")) {  // the series are terminated by an 'x'
    toServerWriter.println(text);  // transfers the text to the server side,
                                   // where converting takes place
    text = fromConsoleReader.readLine();
  }

  toServerWriter.println(text);  // sends the terminate sign
  System.out.println(fromServerReader.readLine()); // fetches the sum from the server side

  connection.close();  // shuts down the connection
```

The output on the clients side may be like this:

```
The name of the machine where the server program is running:
localhost
Now the connection to the server program is established.
***Hello, you are connected to the server!
Enter a number, one on each line, terminate with 'x'
3
4
5
x
The sum of the numbers is 12
```

Chapter 19.2

Problem 1

The following statements are inserted after the existing rebind() statement:

```
System.out.println("We create another server object");
counter = new YesNoCounterImpl();
System.out.println("Now it's made!");
Naming.rebind("AddUpInc", counter);
```

Problem 2

```
import java.rmi.*;
import java.rmi.server.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class CounterWindow extends JFrame {
  private Container guiContainer;
  private JList list;
  private JButton yesButton = new JButton("Yes Votes");
  private JButton noButton = new JButton("No Votes");
  private YesNoCounter[] counterMachines;
  private JLabel status1 = new JLabel("Status");
  private JLabel status2 = new JLabel("");
  private String[] names;  // names of the counter machine objects

  public CounterWindow () {
    setTitle("Records Yes and No Votes");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    try {
```

```
    names = Naming.list("//localhost/");
    counterMachines = new YesNoCounter[names.length];
    for (int i = 0; i < names.length; i++) {
      counterMachines[i] = (YesNoCounter) Naming.lookup(names[i]);
    }
  } catch (Exception e) {
    System.out.println("Error: " + e);
  }

  list = new JList(names);
  list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
  list.setSelectedIndex(0);

  guiContainer = getContentPane();
  guiContainer.setLayout(new BorderLayout(10,10));
  guiContainer.add(new JLabel("    Choose Counter: "), BorderLayout.NORTH);
  guiContainer.add(list, BorderLayout.CENTER);
  guiContainer.add(new SouthPanel(), BorderLayout.SOUTH);
  guiContainer.add(new JLabel("    "), BorderLayout.EAST);
  guiContainer.add(new JLabel("    "), BorderLayout.WEST);
}

private class SouthPanel extends JPanel {
  public SouthPanel() {
    setLayout(new BorderLayout());
    add(new ButtonPanel(), BorderLayout.NORTH);
    add(status1, BorderLayout.CENTER);
    add(status2, BorderLayout.SOUTH);
  }
}

private class ButtonPanel extends JPanel {
  public ButtonPanel() {
    add(yesButton);
    add(noButton);
    ButtonListener listener = new ButtonListener();
    yesButton.addActionListener(listener);
    noButton.addActionListener(listener);
  }
}

private class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    int noOfVotes = 0;
    try {
      int index = list.getSelectedIndex();
      JButton source = (JButton) event.getSource();
      if (source == yesButton) {
        String svar = JOptionPane.showInputDialog(null, "Number of Yes Votes: ");
        noOfVotes = Integer.parseInt(svar);
        counterMachines[index].increaseNumberOfYes(noOfVotes);
      }
      else if (source == noButton) {
        String svar = JOptionPane.showInputDialog(null, "Number of No Votes: ");
        noOfVotes = Integer.parseInt(svar);
        counterMachines[index].increaseNumberOfNo(noOfVotes);
      }
      status1.setText("    Status: " + names[index]);
      status2.setText("    No.of Yes: " + counterMachines[index].getNumberOfYes()
          + ", No. of No: " + counterMachines[index].getNumberOfNo());
    }
    catch (NumberFormatException e) {  // the number of votes is in this case zero
    }
    catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

```
  }
}

class Problem19_2_2 {
  public static void main(String[] args) {
    CounterWindow window = new CounterWindow();
    window.setSize(300, 200);
    window.setLocation(400, 300);
    window.setVisible(true);
  }
}
```

Chapter 19.3

Problem 1

The main() method:

```
public static void main(String[] args) {
  MemberFactory factory = new MemberFactory();
  Person person1 = new Person(100, "Anne", "Smith");
  Member member1 = factory.instantiateMember(person1, "7001 Trondheim");
  person1.setLastName("Johnson");  // member1 is changed
  Person person2 = member1.getPerson();
  System.out.println(person2.getFirstName() + " " + person2.getLastName());
  person2.setLastName("Nilson"); // member1 is changed one more time
  member1.setPerson(person2);  // no effect
  Person person3 = member1.getPerson();
  System.out.println(person3.getFirstName() + " " +
  person3.getLastName());
}
```

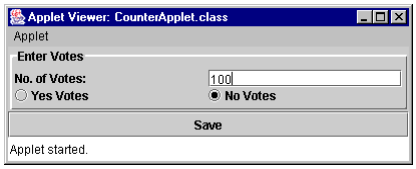
The setLastName() method changes the contents of member1. This is not the case when member1 is remote. The printout from the program above is:

```
Anne Johnson
Anne Nilson
```

Chapter 19.4

Problem 1

Here is the user interface:



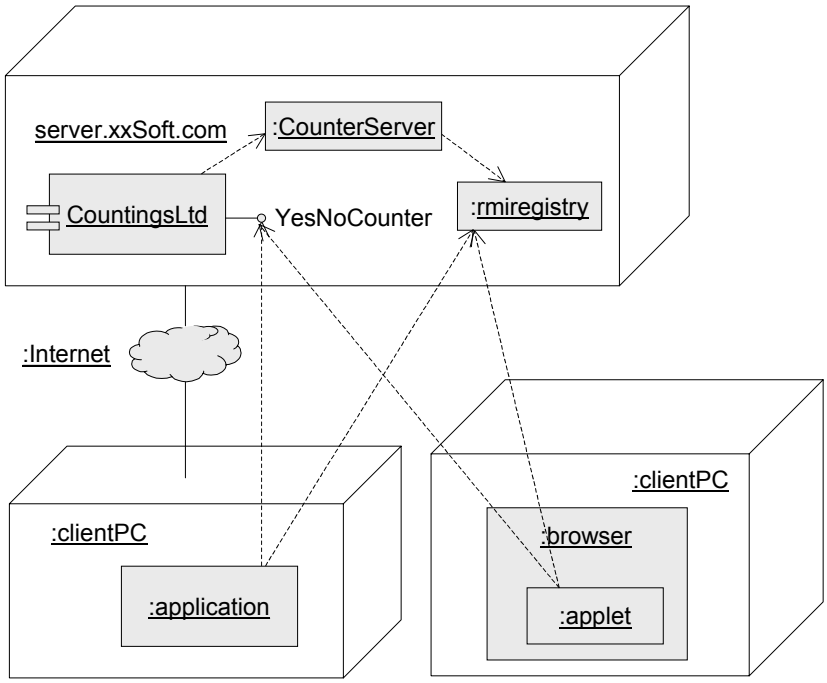
And the source code:

```
/*
 * CounterApplet.java   E.L. 2001-08-25
 *
 */

import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;
import java.awt.*;
import java.rmi.server.*;
import java.rmi.*;

public class CounterApplet extends JApplet {
  private JTextField number = new JTextField(8);
  private JRadioButton yesButton = new JRadioButton("Yes Votes", true);
  private JRadioButton noButton = new JRadioButton("No Votes", false);
  private JButton saveButton = new JButton("Save");
  private YesNoCounter counter;

  public void init() {
```

```
    try {
      String url = "rmi://localhost/";
      counter = (YesNoCounter) Naming.lookup(url + "CountingsLtd");

      Container guiContainer = getContentPane();
      guiContainer.add(new JLabel(), BorderLayout.NORTH);
      guiContainer.add(new InputPanel(), BorderLayout.CENTER);
      guiContainer.add(saveButton, BorderLayout.SOUTH);

      ButtonListener buttonListener = new ButtonListener();
      saveButton.addActionListener(buttonListener);

      number.requestFocus();
    } catch (Exception e) {
      System.out.println("Error in CounterApplet: " + e);
    }
  }

  /* Describes the middle panel */
  private class InputPanel extends JPanel {
    public InputPanel() {
      setLayout(new GridLayout(2, 2));
      add(new JLabel("No. of Votes: "));
      add(number);
      ButtonGroup group = new ButtonGroup();
      group.add(yesButton);
      group.add(noButton);
      add(yesButton);
      add(noButton);
      SoftBevelBorder border = new SoftBevelBorder(BevelBorder.RAISED);
      Border box = BorderFactory.createTitledBorder(border, "Enter Votes");
      setBorder(box);
    }
  }

  private class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent hendelse) {
      int noOfVotes = 0;
      try {
        noOfVotes = Integer.parseInt(number.getText());
      } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Invalid number.");
        number.requestFocus();
      }
      try {
        if (yesButton.isSelected()) counter.increaseNumberOfYes(noOfVotes);
        else counter.increaseNumberOfNo(noOfVotes);
      } catch (Exception e) {
        System.out.println("Error in the listener of the save button: " + e);
      }
      number.setText("");
      number.requestFocus();
    }
  }
}
```

Chapter 19.5

Problem 1



Chapter 19.6

Problem 1

The YesNoCounterFront interface is expanded with the following method:

```
String[] nameAllClients() throws RemoteException;
```

Here is the implementation in the YesNoCounterFrontImpl class:

```
public String[] nameAllClients() throws RemoteException {
  String[] names = new String[allClients.size()];
  for (int i = 0; i < names.length; i++) names[i] = ((Client) allClients.get(i)).getName();
  return names;
}
```

Testing, see below, under the next problem.

Problem 2

A new method in the YesNoCounterFront interface:

```
boolean sendMessage(Client from, String toName, String message)
                                                    throws RemoteException;
```

And the implementation in the YesNoCounterFrontImpl class:

```
public boolean sendMessage(Client from, String toName,
                           String message) throws RemoteException {
  boolean found = false;
  int toIndex = 0;
  Client toClient = null;
  while (toIndex < allClients.size() && !found) {
    toClient = ((Client) allClients.get(toIndex));
    if (toClient.getName().equals(toName)) found = true;
    else toIndex++;
  }
  if (found) {
    toClient.receiveMessage(from.getName(), message);
    return true;
  }
  else return false;
}
```

At the client side, we have a method, printStatus(), which do what we want. However, we create a new method, taking the sender's name as argument and printing to the console window.

The Client interface is expanded:

```
void receiveMessage(String name, String message) throws RemoteException;
```

The implementation:

```
public void receiveMessage(String name, String message) throws RemoteException {
  System.out.println("Message from " + name + ": " + message);
}
```

Test program on the client side, problem 1 and 2:

```
public static void main(String[] args) throws Exception {
  YesNoCounterFront counter = null;
  String nameServer =
        JOptionPane.showInputDialog(null, "The name of the server computer: ");
  String url = "rmi://" + nameServer + "/";
  counter = (YesNoCounterFront) Naming.lookup(url + counterName);
  String clientName = JOptionPane.showInputDialog(null, "Client Name:");
  System.out.println("Console for client: " + clientName);

  /* thisClient will run in its own thread receiving messages from the server side. */
  Client thisClient = new ClientImpl(clientName);

  counter.registerMe(thisClient);

  /*** Problem 1 ****/
  String[] allClients = counter.nameAllClients();
  System.out.println("Now " + allClients.length + " clients are connected:");
  for (int i = 0; i < allClients.length; i++) System.out.println(allClients[i]);

  /*** Problem 2 ***/
  while (true) {
    String toName = JOptionPane.showInputDialog(null, clientName + ", a message to? ");
    String message = JOptionPane.showInputDialog(null, "Enter the message:");
    if (counter.sendMessage(thisClient, toName, message)) {
      System.out.println("Message sent");
    }
    else System.out.println("Message is not sent");
  }
}
```

The server program is as in program listing 19.6.

Chapter 20.2

Problem 1

```
ResultSet res =
      statement.executeQuery("select * from person where lastname = 'JOHNSON'");
```

Problem 2

```
Connection conn
  = DriverManager.getConnection(databaseName, userName, password);

Statement statement = conn.createStatement();
String name =
     JOptionPane.showInputDialog(null, "Enter last name, terminate by Cancel: ");
while (name != null) {
  name = name.trim().toUpperCase();
  ResultSet res = statement.executeQuery
       ("select * from person where lastname = '" + name + "'");
  while (res.next()) {
    int idNo = res.getInt("identNo");
    String firstName = res.getString("firstname");
    String lastName = res.getString("lastname");
    System.out.println(idNo + ": " + firstName + " " + lastName);
  }
  res.close();
  name =
       JOptionPane.showInputDialog(null, "Enter last name, terminate by Cancel: ");
}
statement.close();
conn.close();
```

Chapter 20.3

Problem 1

The Database class is expanded with a new method:

```
public ArrayList getAllDiffLastNames() throws SQLException {
  ArrayList all = new ArrayList();
  String sqlStatement = "select distinct lastname from person";
  System.out.println(sqlStatement);
  ResultSet res = null;
  try {
    res = statement.executeQuery(sqlStatement);
    while (res.next()) {
      String lastname = res.getString("lastname");
      String name = convert(lastname);
      all.add(name);
    }
  } finally {
    if (res != null) res.close();
    return all;
  }
}
```

A little test client:

```
public static void main(String[] args) throws Exception {
  String username = JOptionPane.showInputDialog("User name: ");
  String password = JOptionPane.showInputDialog("Password: ");
  Database db = new Database(username, password);
  System.out.println("All different last names:");
  ArrayList all = db.getAllDiffLastNames();
  for (int i = 0; i < all.size(); i++) {
    String name = (String) all.get(i);
    System.out.println(name);
  }
  db.closeTheConnection();
  System.exit(0);
}
```

Chapter 20.4

Problem 1

The getPerson() method is as follows:

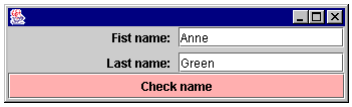
```
public int getPerson(String firstName, String lastName) throws Exception {
  String sqlStatement = "select * from person where firstName = '" +
    firstName.toUpperCase() + "'and lastName = '" + lastName.toUpperCase() + "'";
  System.out.println(sqlStatement);
  ResultSet res = statement.executeQuery(sqlStatement);
  if (res.next()) return res.getInt("identno");
  else return -1;
}
```

Here is the application:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.text.*;
import myLibrary.Person;  // the revised Database class is not in myLibrary

class DatabaseGUI extends JFrame {
  private Database theDatabaseContact;

  private Container guiContainer;
  private JButton checkButton = new JButton("Check name");
  private JTextField firstNameField = new JTextField(20);
  private JTextField lastNameField = new JTextField(20);

  public DatabaseGUI(Database initDatabase) {
```

```
    theDatabaseContact = initDatabase;

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    guiContainer = getContentPane();
    guiContainer.add(new InputPanel(), BorderLayout.CENTER);
    checkButton.setBackground(Color.pink);
    guiContainer.add(checkButton, BorderLayout.SOUTH);
    checkButton.addActionListener(new ButtonListener());
    pack();
  }

  private class InputPanel extends JPanel {
    public InputPanel() {
      setLayout(new GridLayout(2, 2, 5, 5));
      add(new JLabel("Fist name: ", JLabel.RIGHT));
      add(firstNameField);
      add(new JLabel("Last name: ", JLabel.RIGHT));
      add(lastNameField);
    }
  }

  private class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      String firstName = firstNameField.getText().trim();
      String lastName = lastNameField.getText().trim();
      try {
       int id = theDatabaseContact.getPerson(firstName, lastName);
       if (id > 0) {
         JOptionPane.showMessageDialog(null,
           "The person is already registered and has id.no. " + id);
       } else {
         Person p = theDatabaseContact.registerNewPerson(firstName, lastName);
         JOptionPane.showMessageDialog(null,
           "The name is saved, and the person got id.no. " + p.getPersIdent());
       }
      } catch (Exception e) {
        System.out.println("Database error: " + e);
      }
    }
  }
}

class DatabaseApplication {
  public static void main(String[] args) throws Exception {
    String userName = JOptionPane.showInputDialog("User Name: ");
    String password = JOptionPane.showInputDialog("Password: ");
    Database theDatabaseContact = new Database(userName, password);
    DatabaseGUI theApplication = new DatabaseGUI(theDatabaseContact);
    theApplication.setVisible(true);
  }
}
```

The user interface:



Chapter 20.5

Problem 1

The interface is now called Database (no constructor and no private method):

```
import java.rmi.*;
import java.util.*;
import myLibrary.Person;
interface Database extends Remote {
```

```
  void closeTheConnection() throws Exception;
  ArrayList getAll() throws Exception;
  boolean updateName(Person thePerson) throws Exception;
  Person registerNewPerson(String firstName, String lastName)
                                        throws Exception;
  boolean deletePerson(int identNo) throws Exception;
}
```

The implementation is called DatabaseImpl. The first lines:

```
import java.util.*;
import java.sql.*;
import java.rmi.*;
import java.rmi.server.*;
import myLibrary.Person;
public class DatabaseImpl extends UnicastRemoteObject implements Database {
```

The constructor head:

```
public DatabaseImpl(String userName, String password) throws Exception {
```

The head of the public methods have to be changed, for example:

```
public synchronized ArrayList getAll() throws Exception {
```

...or we could have used throws SQLException, RemoteException.

To test with DatabaseTest you have to do two things:

1 A stub class must be created:

```
>rmic -v1.2 DatabaseImpl
```

2 And, in the program, the object has to be an instance of the DatabaseImpl class:

```
Database db = new DatabaseImpl(userName, password);
```

Problem 2

We have to use RMI to get the components communicate.

From problem 1 we have the RMI version of the Database class. The interface is called Database, and the implementation DatabaseImpl.

Every client has to have its own database connection ("theDatabaseContact"). We create a so-called factory method (see section 12.10) that instantiate an object of the DatabaseImpl class. This object resides on the server side.

Here is the interface of the class with the factory method:

```
/*
 * DatabaseConnFactory.java   E.L. 2002-01-18
 *
 * The method returns a database connection
 */
import java.rmi.*;
interface DatabaseConnFactory extends Remote {
  Database createDatabaseConnection(String username, String password)
    throws Exception;
}
```

And the implementation:

```
/*
 * DatabaseConnFactoryImpl.java   E.L. 2002-01-18
 *
 */
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

class DatabaseConnFactoryImpl
    extends UnicastRemoteObject implements DatabaseConnFactory {
  public DatabaseConnFactoryImpl() throws RemoteException {
  }

  public Database createDatabaseConnection(String username, String password)
    throws Exception {
    return new DatabaseImpl(username, password);
```

```
  }
}
```

We have to create stub classes:

```
>rmic -v1.2 DatabaseConnFactoryImpl
>rmic -v1.2 DatabaseImpl //if we don't have it from probl.1
```

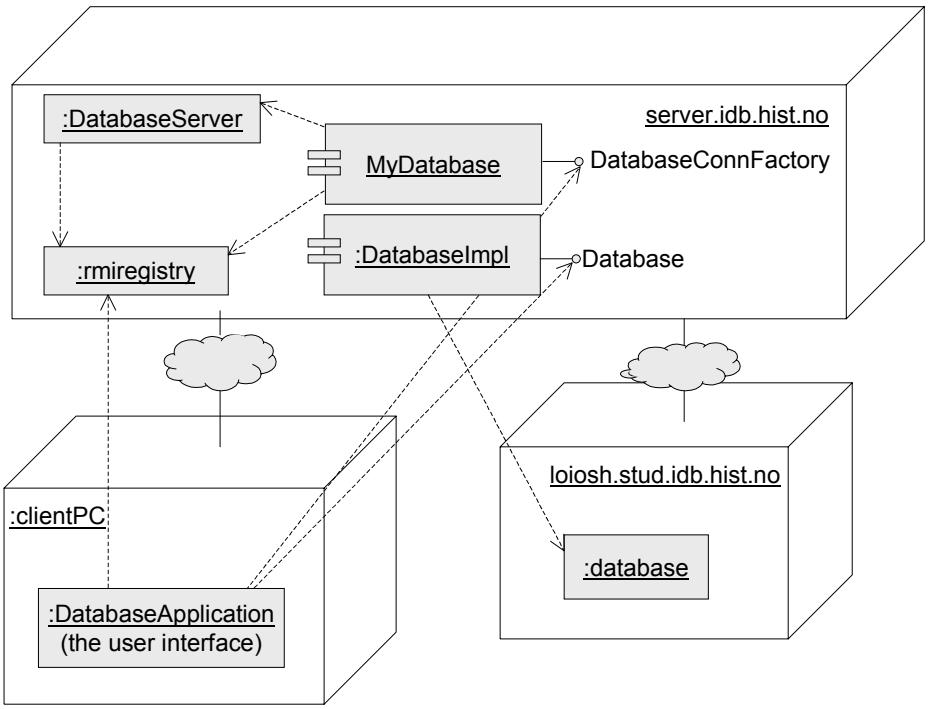
And we need a server program:

```
/*
 * DatabaseServer.java   E.L. 2002-01-18
 */
import java.rmi.*;
import java.rmi.server.*;
class DatabaseServer {
  public static void main(String[] args) throws Exception {
    DatabaseConnFactory connectionFactory = new DatabaseConnFactoryImpl();
    System.out.println("A factory object is created.");
    Naming.rebind("MyDatabase", connectionFactory);
    System.out.println("Waiting...");
  }
}
```

We have to do a few changes in the main() method of the DatabaseApplication:

```
public static void main(String[] args) {
  String host = JOptionPane.showInputDialog("Name of the server computer: ");
  String userName = JOptionPane.showInputDialog("User Name: ");
  String password = JOptionPane.showInputDialog("Password: ");

  Database theDatabaseContact = null;
  try {
    String url = "rmi://" + host + "/MyDatabase";
    DatabaseConnFactory factory = (DatabaseConnFactory) Naming.lookup(url);
    theDatabaseContact = factory.createDatabaseConnection(userName, password);
  } catch (Exception e) {
    JOptionPane.showMessageDialog(null,
      "Problems when establishing database connection, " +
      "the user name or the password may be invalid. \nError: " + e);
    try {
      theDatabaseContact.closeTheConnection(); // disconnects what may be disconnected
    } catch (Exception e1) {
    }
    System.exit(0);
  }
  DatabaseGUI theApplication = new DatabaseGUI(theDatabaseContact);
  theApplication.setVisible(true);
}
```

Problem 3



Chapter 20.6

Problem 1

The Oracle7 SQL script is as follows (not standard number types):

```
create table account(
 accno   varchar(15) not null,
 name    varchar(30) not null,
 balance number not null,
 primary key (accno));

insert into account values ('123456', 'EDWARD BROWN', 1000);
insert into account values ('345678', 'ANN MARGATRET GREEN', 20000);
insert into account values ('678909', 'JOHN JOHNSON', 10000);
commit;
```

Here is a typical transaction for money transfer:

```
update account set balance = balance - 1000 where accno = '123456'
update account set balance = balance + 1000 where accno = '678909'
```

Auto commit is switched off before the first statement is executed. The transaction handling may be tested in two ways:

1 The program controls the rollback.

2 The rollback is caused by a program abortion.

The two alternatives are shown in the program below (alternative 2 is commented away).

```
import javax.swing.*;
import java.sql.*;
class Prob20_6_1 {
  public static void main(String[] args) throws Exception {
    String databaseDriver = "oracle.jdbc.driver.OracleDriver";
    Class.forName(databaseDriver);

    String username = JOptionPane.showInputDialog("User name: ");
    String passord = JOptionPane.showInputDialog("Password: ");
    String databaseName = "jdbc:oracle:thin:@loiosh.stud.idb.hist.no:1521:orcl";
    Connection connection = DriverManager.getConnection(
                                          databaseName, username, passord);
    Statement statement = connection.createStatement();
```

```
    connection.setAutoCommit(false);  // want to set commit/rollback ourselves
    ResultSet res = statement.executeQuery("select * from account");
    System.out.println("The contents of the database: ");
    while (res.next()) {
      System.out.println(res.getString("accno") + ", " + res.getString("name") +
                                                          ", " + res.getString("balance"));
    }
    res.close();

    /* First part of the transaction */
    statement.executeUpdate(
      "update account set balance = balance - 1000 where accno = '123456'");
    res = statement.executeQuery("select * from account where accno = '123456'");
    res.next();
    System.out.println("After withdrawal: " + res.getString("accno") + ", " +
                            res.getString("name") + ", " + res.getString("balance"));
    res.close();

    /* Alternative 1, rollback */
    int option = JOptionPane.showConfirmDialog(null,
                                              "Rollback?", "", JOptionPane.YES_NO_OPTION);

    if (option == JOptionPane.YES_OPTION) connection.rollback();
    else {  // no rollback, finish the transaction
      statement.executeUpdate(
          "update account set balance = balance + 1000 where accno = '678909'");
    }

    /* Alternative 2, program execution is aborted, automatically rollback
    int option = JOptionPane.showConfirmDialog(null,
                  "Skal programmet avbrytes midt i transaksjonen?", "",
                                        JOptionPane.YES_NO_OPTION);

    if (option == JOptionPane.YES_OPTION) System.exit(0); // aborts the program
    else {  // no rollback,finish the transaction
      statement.executeUpdate(
          "update account set balance = balance + 1000 where accno = '678909'");
    }*/

    connection.commit();
    connection.setAutoCommit(true);

    /* Prints all data */
    res = statement.executeQuery("select * from account");
    System.out.println("The contents of the database: ");
    while (res.next()) {
      System.out.println(res.getString("accno") + ", "
        + res.getString("name") + ", " + res.getString("balance"));
    }

    connection.close();
    System.exit(0);
  }
}

/* Example Runs:

Rollback:
The contents of the database:
123456, EDWARD BROWN, -2000
345678, ANN MARGATRET GREEN, 20000
678909, JOHN JOHNSON, 11000
After withdrawal: 123456, EDWARD BROWN, -3000
The contents of the database:
123456, EDWARD BROWN, -2000
345678, ANN MARGATRET GREEN, 20000
```

```
678909, JOHN JOHNSON, 11000

No rollback:
The contents of the database:
123456, EDWARD BROWN, -2000
345678, ANN MARGATRET GREEN, 20000
678909, JOHN JOHNSON, 11000
After withdrawal: 123456, EDWARD BROWN, -3000
The contents of the database:
123456, EDWARD BROWN, -3000
345678, ANN MARGATRET GREEN, 20000
678909, JOHN JOHNSON, 12000
*/
```

Problem 2

We replace

```
private Statement statement;
```

by one statement object for every single SQL statement:

```
private PreparedStatement stmGetAll;
private PreparedStatement stmUpdateName;
private PreparedStatement stmRegNewPerson;
private PreparedStatement stmDelete;
private PreparedStatement stmGetMax;
```

They are initialized in the constructor:

```
stmGetAll = conn.prepareStatement("select * from person order by lastName, firstName");
stmGetMax = conn.prepareStatement("select max(identno) as maxno from person");
stmUpdateName = conn.prepareStatement(
                    "update person set firstname = ?, lastname = ? where identno = ?");
stmRegNewPerson = conn.prepareStatement("insert into person values(?, ?, ?)");
stmDelete = conn.prepareStatement("delete from person where identno = ?");
```

And here are the methods:

```
public void closeTheConnection() throws SQLException {
  if (stmGetAll != null) stmGetAll.close();
  if (stmGetMax != null) stmGetMax.close();
  if (stmUpdateName != null) stmUpdateName.close();
  if (stmRegNewPerson != null) stmRegNewPerson.close();
  if (stmDelete != null) stmDelete.close();
  if (conn != null) conn.close();
  System.out.println("** From the Database class: " +
                "The connection to the database is closed.");
}

public ArrayList getAll() throws SQLException {
  ArrayList all = new ArrayList();
  ResultSet res = null;
  try {
    res = stmGetAll.executeQuery();
    while (res.next()) {
      int identNo = res.getInt("identno");
      String firstName = res.getString("firstname");
      String lastName = res.getString("lastname");
      Person thePerson = new Person(identNo, convert(firstName),
            convert(lastName)); // convert, private method, see below
      all.add(thePerson);
    }
  } finally { // the statements below are executed regardless of exceptions thrown
    /* Releases database resources */
    if (res != null) res.close();
    return all;
  }
}

public boolean updateName(Person thePerson) throws SQLException {
  int identNo = thePerson.getPersIdent();
  String newFirstName = thePerson.getFirstName().toUpperCase();
```

```
    String newLastName = thePerson.getLastName().toUpperCase();
    stmUpdateName.setString(1, newFirstName);
    stmUpdateName.setString(2, newLastName);
    stmUpdateName.setInt(3, identNo);
    if (stmUpdateName.executeUpdate() == 0) return false;
    else return true;
  }

  public Person registerNewPerson(String firstName, String lastName)
                                          throws SQLException {
    firstName = firstName.toUpperCase();
    lastName = lastName.toUpperCase();
    int newIdentNo = 1; // if no persons in the database
    boolean ok = true;
    do {
      ResultSet res = null;
      try {
        res = stmGetMax.executeQuery();
        if (res.next()) newIdentNo = res.getInt("maxno") + 1;
        stmRegNewPerson.setInt(1, newIdentNo);
        stmRegNewPerson.setString(2, firstName);
        stmRegNewPerson.setString(3, lastName);
        stmRegNewPerson.executeUpdate();
      } catch (SQLException e) {
        /* If the error code means that a person with this no. already exists,
           another client must have stored this person in between our two sql statements.
           We therefore rerun the do loop. (This happens very rarely.) */
        if (e.getErrorCode() == code) ok = false;
        else throw e;
      } finally {
        if (res != null) res.close();
      }
    } while (!ok);
    return new Person(newIdentNo, firstName, lastName);
  }

  public boolean deletePerson(int identNo) throws SQLException {
    stmDelete.setInt(1, identNo);
    if (stmDelete.executeUpdate() == 0) return false;
    else return true;
  }

  private String convert(String text) {
    String newText = "";
    StringTokenizer theText = new StringTokenizer(text);
    while (theText.hasMoreTokens()) {
      String s = theText.nextToken();
      newText += s.substring(0, 1);
      if (s.length() > 0) newText += s.substring(1).toLowerCase() + " ";
    }
    newText = newText.trim();
    return newText;
  }
}
```

Chapter 21.3

Problem 2

The "body" now looks like this:

```
out.println("<body>");
out.println("This page was downloaded: " + new java.util.Date());
out.println("<p>Random numbers in the interval [0, 1000]: ");
java.util.Random rand = new java.util.Random();
for (int i = 0; i < 5; i++) out.println(rand.nextInt(1001) + " ");
out.println("</body>");
```

Problem 3

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  System.out.println("A request from " + request.getRemoteHost());
  response.setContentType("text/html");
  .....
```

Chapter 21.5

Problem 1

```
<%@ page import="java.util.*"%>
<html><head><title>Problem 21_5_1</title></head>
<body>
This page was downloaded:
<%
GregorianCalendar calendar = new GregorianCalendar();
int day = calendar.get(Calendar.DAY_OF_WEEK);
String text = (new java.util.Date()).toString();
if (day == Calendar.SATURDAY || day == Calendar.SUNDAY) {
  text += " - weekend";
  if (day == Calendar.SUNDAY) {
    text = "<font color = 'red'>" + text + "</font>";
  }
}
%>
<%= text %>
</body></html>
```

Problem 2

```
<%@ page import="java.text.*"%>
<html><head><title>Problem 21_5_2</title></head>
<body>
<table border=1>
<tr><th>Number</th><th>Square Root</th></tr>
<%
DecimalFormat thisFormat = new DecimalFormat("###0.000");
for (int no = 1; no <= 20; no++) {
  out.println("<tr><td>" + no + "</td><td>" + thisFormat.format(Math.sqrt(no)) +
                                                                    "</td></tr>");
}
%>
</table>
</body></html>
```

Problem 3

```
<%@ page import="java.util.*"%>
<html><head><title>Problem 21_5_3</title></head>
<body>

<%! private int counter = 0; %>
<%! private int sum = 0; %>
<%! private Vector allNumbers = new Vector(); %>

<%
counter++;
int random = (int) (100 * Math.random() + 1);
allNumbers.add(new Integer(random));
sum += random;
%>
<P>The page is accessed <%= counter %> times.
<P>The sum of the random numbers is <%= sum %>.
<P>The random numbers are:
<% for (int i = 0; i < allNumbers.size(); i++) {
    out.print(" " + (Integer) (allNumbers.get(i)));
  }
 %>
</body></html>
```

Chapter 21.6

Problem 1

Here is the *html* file (named *Prob21_6_1.html*):

```
<html><head><title>Input Fields</title></head>
<form action="Prob21_6_1.jsp" method="post">
<p>
<table> <!-- uses table to get tabular format for the fields -->
<tr><td>First Name</td><td><input name="firstName" type="text" size="30"></td>
<tr><td>Last Name</td><td><input name="lastName" type="text" size="30"></td>
<tr><td>Country</td><td><input name="country" type="text" size="30"></td>
</table>
<p>Sex:
<br>Female<input name="sex" type="radio" value="female" checked>
   Male<input name="sex" type="radio" value="male">
<p>
<input type="submit" name="Send" value="Send">
<input type="reset" name="Reset" value="Reset">
</form>
</body></html>
```

And here is the *jsp* file (named *Prob21_6_1.jsp*) which handles the entered data:

```
<html><head><title>Feedback</title></head><body>
<%
String theCountry = request.getParameter("country");
String firstName = request.getParameter("firstName");
String lastName = request.getParameter("lastName");
if (theCountry == null || firstName == null || lastName == null) {
  out.println("You have to fill out all three fields.");
} else out.println("Hallo, " + firstName.trim() + " " + lastName.trim() + " from " + theCountry.trim() + "!");
%>
<p>
<a href = "Prob21_6_1.html">Back</a>
</body></html>
```

Problem 2

Here is the new file, which is fired from *RestaurantEvaluation.html*:

```
<%--
   A help method composing an array of strings into one single string.
   This method was in the RestaurantEvaluation.jsp file before.
--%>
<%!
String composeOneString(String[] values) {
  String text = "";
  if (values != null) {
    for (int i = 0; i < values.length - 1; i++) text += values[i] + ", ";
    text += values[values.length - 1];
  }
  return text;
}
%>

<%
String nameRestaurant = request.getParameter("nameRestaurant");
String like = composeOneString(request.getParameterValues("like"));
String notLike = composeOneString(request.getParameterValues("notLike"));
String menu = request.getParameter("menu");
String comments = request.getParameter("comments");
String sex = request.getParameter("sex");
String age = request.getParameter("age");
%>

<p>You've entered the following data:

<%
out.println("<br>Restaurant: " + nameRestaurant);
out.println("<br>Pleased with: " + like);
```

```
out.println("<br>Not pleased with: " + notLike);
out.println("<br>The guest ate: " + menu);
out.println("<br>Comments: " + comments);
out.println("<br>Sex: " + sex);
out.println("<br>Age: " + age);
%>
<form action="RestaurantEvaluation.jsp" method="post">
<!-- uses hidden fields to transfer the data to the revised RestaurantEvaluation.jsp file -->
<input name="nameRestaurant" type="hidden" value="<%= nameRestaurant %>">
<input name="like" type="hidden" value="<%= like %>">
<input name="notLike" type="hidden" value="<%= notLike %>">
<input name="menu" type="hidden" value="<%= menu%>">
<input name="comments" type="hidden" value="<%= comments%>">
<input name="sex" type="hidden" value="<%= sex %>">
<input name="age" type="hidden" value="<%= age %>">

<input type="submit" name="Save" value="OK to save">
</form>
<font color = "red">If you want to edit the entered data, press the Back button in your browser.
</font>
</body></html>
```

The *RestaurantEvaluation.jsp* file is a bit revised:

```
<%@ page import="java.io.*" %>
<%! String nameOfFile = "EvalRest.txt"; %>
<%
if (request.getParameter("Save") != null) {

  String output =  // from the hidden fields
    "Restaurant: " + request.getParameter("nameRestaurant") + "\n" +
    "Pleased with: " + request.getParameter("like") + "\n" +
    "Not pleased with: " + request.getParameter("notLike") + "\n" +
    "The guest ate: " +  request.getParameter("menu") + "\n" +
    "Comments: " + request.getParameter("comments") + "\n" +
    "Sex: " + request.getParameter("sex") + "\n" +
    "Age: " + request.getParameter("age") + "\n" + "\n" + "\n";

  String fileName = application.getRealPath(nameOfFile);
  FileWriter writeConnToFile = new FileWriter(fileName, true);
  PrintWriter printer = new PrintWriter(new BufferedWriter(writeConnToFile));
  printer.print(output);
  printer.close();
  out.println("Your evaluation is saved.");
} else {
  out.println("Nothing saved");
}
%>

</body></html>
```

Chapter 21.8

Problem 1

```
<table border=1>
  <tr><th>Ident.no.</th><th>First Name</th><th>Last Name</th></tr>
  <%
  int counter = 0;
  try{
    .........
    while (resSet.next()) {
      counter++;
      .............
   }
.....................
</table>
<p>The number of persons is <%= counter%>
```

Problem 2

```
<html><head><title>A Simple Test of Database Connection</title></head>
<body bgcolor="wheat" text="black" link="darkgreen" vlink="steelblue" alink="darkblue">
<font size="+2">A Listing Of Persons</font>
<br><br>
<%@ page import="java.sql.*" %>

<%
String searchValue = request.getParameter("lastName");
if (searchValue == null) searchValue = "";
%>
<p>
<form action="SimplePersonTable.jsp" method="post">
  Enter last name to narrow the search:
  <br><input name="lastName" type="text" size="30" value="<%=searchValue%>">
  <p><input type="submit" name="send" value="Search">
</form>

<table border=1>
  <tr><th>Ident.no.</th><th>First Name</th><th>Last Name</th></tr>
  <%
  try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn = DriverManager.getConnection
      ("jdbc:oracle:thin:@loiosh.stud.idb.hist.no:1521:orcl", "username", "password");
    java.sql.Statement statement = conn.createStatement();
    String sql = "select * from person";
    if (!searchValue.equals("")) sql +=
                                      " where lastname = '" + searchValue.toUpperCase() + "'";
    ResultSet resSet = statement.executeQuery(sql);
    .................
```

Problem 3

The most important changes is in the *NameArchive.jsp* file.

We have to handle the updates in this file, because the update messages should be shown in this page. The updates are handled in the three if statements in the first part of the file. This is necessary because it affect the data contents, and the list should display the updated data. None of these conditions will be true the first time a client downloads this page.

As before, the form in this page fires the *HandleSelection.jsp* file. The form in the *PersonForm.jsp* is the one that fires this *NameArchive.jsp* file.

```
<%@ page import="java.util.StringTokenizer" %>
<html><head><title>Name Archive</title></head>
<body bgcolor="wheat" text="black" link="darkgreen" vlink="steelblue" alink="darkblue">
<h1>NameArchive</h1>

<!-- We have to handle the updates before we display the contents of the archive -->

<% /* Deletes a person from the archive */
 if (request.getParameter("delete") != null) {
   String selectedPerson = request.getParameter("persons");
   StringTokenizer text = new StringTokenizer(selectedPerson, ":, ");
   int identNo = Integer.parseInt(text.nextToken());
 %>
   <%@ include file = "DeletePersonData.jsp"%>
 <%
 }

 /* Saves the new record */
 if (request.getParameter("saveNew") != null) {
 %>
   <%@ include file = "AddPersonData.jsp"%>
 <%
 }

 /* Saves the edited record */
```

```
 if (request.getParameter("saveUpdate") != null) {
 %>
   <%@ include file = "UpdatePersonData.jsp"%>
 <%
 }
%>

<form action = "HandleSelection.jsp" method = "post">
  ...as before...
</form>
</body></html>
```

The *HandleSelection.jsp* file is slightly revised:

```
<html><head><title>Handles User's Choice</title></head>
<body bgcolor="wheat" text="black" link="darkgreen" vlink="steelblue" alink="darkblue">

<%@ page import="java.util.StringTokenizer" %>
<% /* Deletes a person from the archive */
  if (request.getParameter("delete") != null) {
  %>
    <%-- This jsp action gets the request forwarded to the page mentioned. --%>
    <jsp:forward page="NameArchive.jsp"/>
  <%
  }

  /* Adds a new person to the archive*/
  if (request.getParameter("new") != null) {
    String idNo = "not defined";
    String firstName = "";
    String lastName = "";
    String commandName = "saveNew";  /* The name of the submit button */
    %>
    <%@ include file = "PersonForm.jsp"%>
  <%
  }

  /* Changes the name of a person in the archive */
  if (request.getParameter("edit") != null) {
    String selectedPerson = request.getParameter("persons");
    StringTokenizer text = new StringTokenizer(selectedPerson, ":, ");
    String idNo = text.nextToken();
    String lastName = text.nextToken();
    String firstName = text.nextToken();
    int noOfWordsLeft = text.countTokens();  // handles the middle name
    for (int i = 0; i < noOfWordsLeft; i++) firstName += (" " + text.nextToken());
    String commandName = "saveUpdate"; /* The name of the submit button */
    %>
    <%@ include file = "PersonForm.jsp"%>
  <%
  }
%>
</body>
</html>
```

In *PersonForm.jsp* (shown below) we now have only one "managing file", which is *NameArchive.jsp*. But we must have two variants of the submit button. This button is used in the *NameArchive.jsp* file to determine whether it was an update or an insert. The name of this button is determined above, and it is stored in the commandName variable.

Only two small changes in the *PersonForm.jsp* file:

```
<br><strong>Add/change data</strong>
<form action="NameArchive.jsp" method = "post">
  <table>
    ... as before ...
  </table>
  <input type = "submit" name = "<%=commandName%>"  value = "Save">
  <input type = "reset" name = "reset" value = "Reset">
</form>
```

There are three more files: *DeletePersonData.jsp*, *AddPersonData.jsp* and *UpdatePerson.jsp*. The only changes here are that the links to the main page (at the bottom of the files) are removed.

Chapter 21.9

Problem 1

Three more input fields in the *CookiesTest.jsp* file:

```
<table>
 <tr>
  <td align = left>First Name:</td>
  <td align = left><input type = "text" name = "firstName" size = "20"></td>
 </tr><tr>
  <td align = left>Last Name:</td>
  <td align = left><input type = "text" name = "lastName" size = "20"></td>
 </tr><tr>
  <td align = left>Address:</td>
  <td align = left><input type = "text" name = "address" size = "20"></td>
 </tr><tr>
  <td align = left>City:</td>
  <td align = left><input type = "text" name = "city" size = "20"></td>
 </tr><tr>
  <td align = left>Zip Code:</td>
  <td align = left><input type = "text" name = "zip" size = "20"></td>
 </td>
</table>
```

The code structure in the *SaveNameAsCookies.jsp* file is revised a little:

```
<html><head><title>Save the Name as Cookies</title></head><body>
<%
 String firstName = request.getParameter("firstName");
 String lastName = request.getParameter("lastName");
 String address = request.getParameter("address");
 String city = request.getParameter("city");
 String zip = request.getParameter("zip");
 response.addCookie(new Cookie("firstName", firstName));
 response.addCookie(new Cookie("lastName", lastName));
 response.addCookie(new Cookie("lastName", lastName));
 response.addCookie(new Cookie("address", address));
 response.addCookie(new Cookie("city", city));
 response.addCookie(new Cookie("zip", zip));
 out.println("<br>Now the cookies are stored.");
 out.println("<br>Push the 'Back' button and 'Reload' the previous page.");
%>
```

Problem 2

Insert this statement

```
cookie1.setMaxAge(24 * 3600);
```

after instantiating the Cookie object in the *SaveNameAsCookies.jsp* file.

Problem 3

Vi have to do a little change in the *RestaurantEvaluation.jsp* file:

```
<%
if (request.getParameter("Send") != null) {
 if (session.getAttribute("saved") == null) {
  String output =
   "Restaurant: " + request.getParameter("nameRestaurant") + "\n" +
    ... and so on, as before ....
  printer.close();
  session.setAttribute("saved", "ok");
  out.println("<P>Your evaluation is saved.");
 } else out.println("<P>Your evaluation is already saved.");
}
%>
```

Problem 4

The text file consists of one single line with all data, as follows:

```
Annunziato, Kaminaris: JavaServer Pages in 24 Hours*29.99*1*Hougland
and Tavistock: Core JSP*42.99*0*Pekowsky: JavaServer Pages*39.95*3*
Hall: Core Servlets and JavaServer Pages*42.99*0*Bergsten:
JavaServer Pages*39.95*2
```

* is used as delimiter. Exact three data about each book: First the title (and the author), then the price, and finally the number of books ordered.

Why only one line with data? Because it's important to read all data in one read-operation. No other clients may then intervene and disrupt the data.

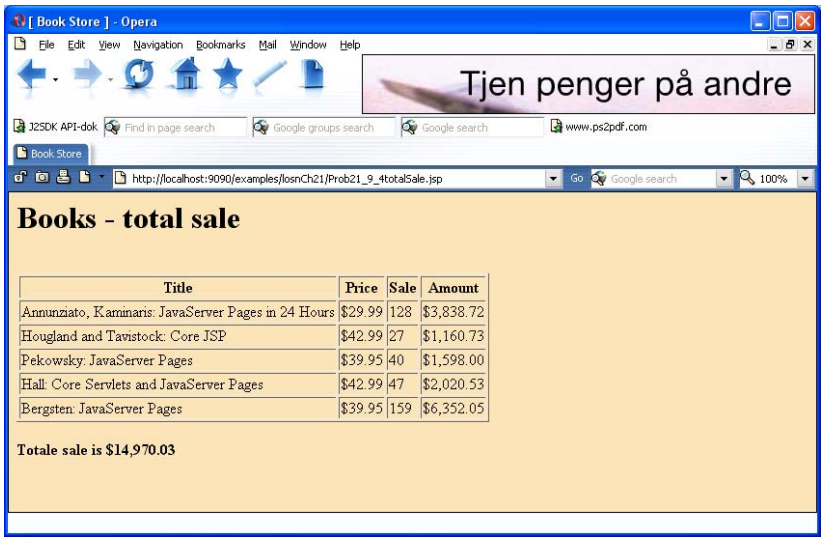
All the *jsp*-files are found in the zip file http://www.tisip.no/javatheumlway/div/Problem21_9_4.zip. The solution consists of one "main" file (Prob21_9_4.jsp) and two inlude-files (Prob21_9_4update.jsp and Prob21_9_4send.jsp).

```
<!--
  Prob21-9-4.jsp  E.L. 2003-03-04
-->
<html><head><title>Book Store</title></head>
<body bgcolor="wheat" text="black" link="darkgreen" vlink="steelblue" alink="darkblue">
<h1>Books</h1>

<%@ page import="myLibrary.Book, java.util.*, java.text.NumberFormat, java.io.*" %>

<%! private NumberFormat currencyFormat; %>
<%! private Vector bookTitles = new Vector(); %>
<%! private static final String filename = "d:/temp/books.txt"; %>
<%! private static final String delimiter = "*";  // at the file %>

<%! public void jspInit() { // initializations common for all clients

    /* the currency format */
    Locale.setDefault(new Locale("en", "US"));  // the prices are in dollars
    currencyFormat = NumberFormat.getCurrencyInstance();

    /* Inputting data from file.
       It is important to read all data at once. This could
       be accomplished by having all data in one line. There
       are other solutions, too, e.g. serialization. Then the
       file has to be created by a program that serializes
       the ArrayList object for the first time.

       The data is separated by a delimiter (asterisks).
       There are exactly three data per book: title, price and number.
       No error checking of input data.
    */
    String dataRead = null;
    try {
      FileReader readConnToFile = new FileReader(filename);
      BufferedReader reader = new BufferedReader(readConnToFile);
      dataRead = reader.readLine();
      reader.close();
    } catch (IOException e) {
      System.out.println("File error: " + e);
    }

    StringTokenizer data = new StringTokenizer(dataRead, delimiter);
    while (data.hasMoreTokens()) {
      String title = data.nextToken();
      String priceS = data.nextToken();
      double price = Double.parseDouble(priceS);
      String numberS = data.nextToken();
      int number = Integer.parseInt(numberS);
      Book newBook = new Book(title, price);
      newBook.setNumber(number);
```

```
        bookTitles.add(newBook);
      }
    }
%>
<%! public void createBookList(HttpSession session) {
      int[] myOrder = new int[bookTitles.size()]; // all elements are 0
      session.setAttribute("myOrder", myOrder);
    }
%>

<%
  if (session.isNew()) {
    createBookList(session);
    out.println("Welcome to the Book Store!");
  }

  int[] myOrder = (int[]) session.getAttribute("myOrder");

  if (myOrder == null) {
    out.println("You've already sent an order in this session.<br>" +
                "If you want to order more books, please, close and reopen the browser.");
  } else {
    double totalprice = 0.0;
    int totalnumber = 0;

    if (request.getParameter("reset") != null) {
      for (int i = 0; i < bookTitles.size(); i++) myOrder[i] = 0;
      out.println("No books ordered.");
    }

    if (request.getParameter("update") != null ||
        request.getParameter("send") != null) {
    %>
    <%@ include file = "Prob21_9_4update.jsp"%>
    <%
    }

    if (request.getParameter("send") == null) {
    %>
      <br>
      <form action = "Prob21_9_4.jsp" method = "post">
      <table border=1>
        <tr><th>Title</th><th>Price</th><th>Number</th>
        <%
          for (int i = 0; i < bookTitles.size(); i++) {
            Book thisBook = (Book) bookTitles.get(i);
           %>
           <tr><td> <%=thisBook.getTitle()%></td>
              <td> <%=currencyFormat.format(thisBook.getPrice())%></td>
              <td> <input type="text" name="number""
                   value="<%=myOrder[i]%>"></td>
           </tr>
           <%
          }
       %>
      </table>
      <br>
      <input type="submit" name="update" value="Update Status">
      <input type="submit" name="reset" value="Reset the Order">
      <input type="submit" name="send" value="Send Order">
      </form>
    <%
    } else {
    %>
    <%@ include file = "Prob21_9_4send.jsp"%>
    <%
    }
```

```
  }
  %>
</body></html>

<!--
  Prob21_9_4update.jsp  EL 2003-03-05
  This file is included in Problem21_9_4.jsp
-->

<%
  int index = 0;
  Book thisBook = null;
  try {
    String[] numbersString = request.getParameterValues("number");
    for (index = 0; index < numbersString.length; index++) {
      thisBook = (Book) bookTitles.get(index);
      int number = Integer.parseInt(numbersString[index]);
      totalnumber += number;
      totalprice += thisBook.getPrice() * number;
      myOrder[index] = number;
    }

  out.println("The number of books up to now: " + totalnumber +
    ", <br>Total price: " + currencyFormat.format(totalprice) + ".");
  } catch (NumberFormatException e) {
     out.println("Invalid number entered for " +
     thisBook.getTitle());
  }
%>

<!--
  Prob21_9_4send.jsp  EL 2003-03-05
  This file is included in Problem21_9_4.jsp
-->
<%
  /* updates the instance variable */
System.out.println("SIZE = " + bookTitles.size());
  for (int i = 0; i < bookTitles.size(); i++) {
    Book book = (Book) bookTitles.get(i);
System.out.println("Titel" + book.getTitle());
    book.setNumber(book.getNumber() + myOrder[i]);
  }

  /* constructs a string for output to file */
  Book book = null;
  String output = "";
  for (int i = 0; i < bookTitles.size(); i++) {
    book = (Book) bookTitles.get(i);
    output += book.getTitle() + delimiter + book.getPrice() +
            delimiter + book.getNumber() + delimiter;
  }
  /* removes the last delimiter */
  int pos = output.lastIndexOf(delimiter);
  output = output.substring(0, pos);

  /* save to file */
  FileWriter writeConnToFile = new FileWriter(filename, false);
  PrintWriter printer = new PrintWriter(new BufferedWriter(writeConnToFile));
  printer.print(output);
  printer.close();

  /* clears the session variable */
  session.removeAttribute("myOrder");
  out.println("<br>The order is sent.");
%>
```

As a utility jsp, I've made a jsp-file presenting the contents of the data file, *Problem 21-9-4totalSale.jsp*:




```
<!--
  Problem 21-9-4totalSale.jsp   E.L. 2003-03-04
-->
<html><head><title>Book Store</title></head>
<body bgcolor="wheat" text="black" link="darkgreen" vlink="steelblue" alink="darkblue">
<h1>Books - total sale</h1>

<%@ page import="myLibrary.Book, java.util.*, java.text.NumberFormat, java.io.*" %>

<%! private NumberFormat currencyFormat; %>
<%! private Vector bookTitles = new Vector(); %>
<%! private static final String filename = "d:/temp/books.txt"; %>
<%! private static final String delimiter = "*";  // at the file %>

<%! public void jspInit() { // initializations common for all clients

     /* the currency format */
     Locale.setDefault(new Locale("en", "US"));  // the prices are in dollars
     currencyFormat = NumberFormat.getCurrencyInstance();
   }
%>

<%

 String dataRead = null;
 try {
   FileReader readConnToFile = new FileReader(filename);
   BufferedReader reader = new BufferedReader(readConnToFile);
   dataRead = reader.readLine();
   reader.close();
 } catch (IOException e) {
   System.out.println("File error: " + e);
 }

 StringTokenizer data = new StringTokenizer(dataRead, delimiter);
 while (data.hasMoreTokens()) {
   String title = data.nextToken();
   String priceS = data.nextToken();
   double price = Double.parseDouble(priceS);
   String numberS = data.nextToken();
   int number = Integer.parseInt(numberS);
   Book newBook = new Book(title, price);
   newBook.setNumber(number);
   bookTitles.add(newBook);
```

```
    }

    double total = 0.0;
  %>
    <br>
    <table border=1>
      <tr><th>Title</th><th>Price</th><th>Sale</th><th>Amount</th>
      <%
        for (int i = 0; i < bookTitles.size(); i++) {
          Book thisBook = (Book) bookTitles.get(i);
          %>
          <tr><td> <%=thisBook.getTitle()%></td>
              <td> <%=currencyFormat.format(thisBook.getPrice())%></td>
              <td> <%=thisBook.getNumber()%></td>
              <td> <%=currencyFormat.format(thisBook.getNumber() *
                                                  thisBook.getPrice())%></td>
          </tr>
          <%
          total += thisBook.getNumber() * thisBook.getPrice();
        }
      %>
    </table>
    <br>
    <strong>Totale sale is <%=currencyFormat.format(total)%></strong>
  </body></html>
```