Inheritance and Polymorphism

Zbigniew Dendzik
Institute of Physics, University of Silesia

2025

Introduction

Inheritance and polymorphism are fundamental mechanisms of object-oriented programming. The examples and exercises in this module will allow us to familiarise ourselves with the syntax and concepts behind inheritance and polymorphism in the context of the Java language. We will also use the acquired skills to design and implement a simple example of your own idea using object-oriented programming methods.

Example I

Below you will find the source code of the `Rectangle` class, which inherits from the library class `java.awt.Rectangle`. Familiarise yourself with the documentation of the `java.awt.Rectangle` class and, based on information from the lecture, identify the member elements that the `Rectangle` class will inherit from `java.awt.Rectangle`. Answer the question of what they represent and/or what they are used for.

```
1  import java.awt.Rectangle;
2
3  class Rectangle extends java.awt.Rectangle
4  {
5      Rectangle(int a,int b)
6      {
7          super(a,b);
8      }
9
10     void info()
11     {
12         System.out.println(this);
13     }
14 }
15
16 public class Program
17 {
18     public static void main(String[] args)
19     {
20
21         Rectangle a=new Rectangle(3,4);
```

```
22          a.info();
23
24          Rectangle b=new Rectangle(2,2);
25          b.info();
26
27
28
29          if(a.intersects(b))
30          {
31              System.out.println("-- they intersect --\n");
32          }
33          else
34          {
35              System.out.println("-- they do NOT intersect --\n");
36          }
37
38
39
40          a.translate(5,3);
41          a.info();
42
43          if(a.intersects(b))
44          {
45              System.out.println("-- they intersect --\n");
46          }
47          else
48          {
49              System.out.println("-- they do NOT intersect --\n");
50          }
51
52      }
53  }
```

Listing 1: Example I: inheritance from java.awt.Rectangle class

Ex. 4.1

Analyse the contents of the `main()` method of the `Program` class. Identify the places where `Rectangle` objects were created and the places where method calls are made on these objects. Which of these methods were implemented in the `Rectangle` class, and which are inherited from the `java.awt.Rectangle` class? Based on the information contained in the documentation of the `java.awt.Rectangle` class, answer the question of what each of these methods is used for and for what purpose it was used in the above example.

Ex. 4.2

Answer the question of whether superclass constructors are inherited, and also what `super()` means in the subclass constructor call. Based on the information contained in the documentation of the `java.awt.Rectangle` class, implement the constructor `Rectangle(Point vertex, int width, int height)`, where `vertex` will be an object of the `java.awt.Point` class. Compile and test the example.

Ex. 4.3

Based on the information contained in the documentation of the `java.awt.Rectangle` class, implement in the `Rectangle` class a method that checks whether a given rectangle is adjacent to another rectangle. Compile and test the example.

Example II

Below you will find a simple example illustrating the concept and use of polymorphism. The example consists of an abstract base class `Shape` and several concrete classes containing implementations of methods for calculating the areas and perimeters of geometric figures.

```
1  abstract class Shape //cannot create instances of this class
2  {
3     abstract double area(); //abstract method
4     abstract double perimeter();
5
6     void info()
7     {
8        System.out.println(this);
9     }
10 }
11
12 class Circle extends Shape
13 {
14    double radius;
15
16    Circle(double radius)
17    {
18       this.radius=radius;
19    }
20
21    double area()
22    {
23       return 3.14*radius*radius;
24    }
25
26    double perimeter()
27    {
28       return 2*3.14*radius;
29    }
30
31    public String toString()
32    {
33       return "circle with r. "+radius;
34    }
35 }
36
37 class Rectangle extends Shape
38 {
```

```
39      double width;
40      double height;
41
42      Rectangle(double width,double height)
43      {
44          this.width=width;
45          this.height=height;
46      }
47
48      double area()
49      {
50          return width*height;
51      }
52
53      double perimeter()
54      {
55          return 2*width+2*height;
56      }
57
58      public String toString()
59      {
60          return "rectangle with dim. "+width+" by "+height;
61      }
62  }
63
64  public class Program
65  {
66      public static void main(String[] args)
67      {
68          Shape z=new Circle(2);
69          z.info();
70
71          Shape[] a={new Rectangle(3,5),new Circle(8),new Circle(3)};
72
73          Shape x;
74          double sum=0;
75
76          for(int i=0;i<a.length;i++)
77          {
78              x=a[i];
79              x.info();
80              sum=sum+x.area();
81          }
82
83          System.out.println("sum of shape areas: "+sum);
84      }
85  }
```

Listing 2: Example II: polymorphism with geometric shape classes

Ex. 4.4

Extend the above example by adding classes representing several other geometric figures. Implement appropriate methods for calculating the areas and perimeters of these figures.

Example III

Below you will find the skeleton of a document database model that illustrates the mechanism of using interfaces. It is worth remembering that in the Java language, classes can implement multiple interfaces, unlike in the case of inheritance.

```
1  class Person
2  {
3
4  }
5
6  interface Searchable
7  {
8      boolean matches(String pattern);
9  }
10
11 abstract class Document implements Searchable
12 {
13
14 }
15
16 class Passport extends Document
17 {
18     public boolean matches(String pattern)
19     {
20         return false;
21     }
22
23     public String toString()
24     {
25         return "";
26     }
27 }
28
29 class IDCard extends Document
30 {
31     public boolean matches(String pattern)
32     {
33         return false;
34     }
35
36     public String toString()
37     {
38         return "";
39     }
40 }
```

```
41
42 public class Program
43 {
44    public static void main(String[] args)
45    {
46       Document[] database={new Passport(),new IDCard(),new
            Passport()};
47
48       Document z;
49       String pattern="Smith";
50
51       for(int i=0;i<database.length;i++)
52       {
53          z=database[i];
54          if(z.matches(pattern))System.out.println("found: "+z);
55       }
56    }
57 }
```

Listing 3: Example III: using interfaces

Ex. 4.5

Extend the above example. Add appropriate fields and implement appropriate methods to obtain a document database model that performs searching according to a given pattern. Note: comparing the contents of two `String` objects can be done, for example, by calling the `equalsIgnoreCase(String anotherString)` function from the `java.lang.String` class.

Example IV

Below you will find an example illustrating object serialization to a data stream using the `Serializable` interface included in the Java Platform API.

```
1  import java.io.*;
2
3  class Person implements Serializable
4  {
5     String firstName;
6     String lastName;
7     int birthYear;
8
9     Person(String firstName,String lastName,int birthYear)
10    {
11       this.firstName=firstName;
12       this.lastName=lastName;
13       this.birthYear=birthYear;
14    }
15
16    Person(BufferedReader br)
```

```
17      {
18          try
19          {
20              System.out.print("first name: ");
21              this.firstName=br.readLine();
22
23              System.out.print("last name: ");
24              this.lastName=br.readLine();
25
26              System.out.print("birth year: ");
27              this.birthYear=Integer.parseInt(br.readLine());
28          }
29          catch(IOException e){}
30      }
31
32      public String toString()
33      {
34          return this.firstName+" "+this.lastName+" "+this.birthYear;
35      }
36  }
37
38  class IDCard implements Serializable
39  {
40      Person holder;
41      String number;
42
43      IDCard(BufferedReader br)
44      {
45          try
46          {
47              this.holder=new Person(br);
48
49              System.out.print("ID number: ");
50              this.number=br.readLine();
51          }
52          catch(IOException e){}
53      }
54
55      public String toString()
56      {
57          return "<id:> "+holder.toString()+" "+this.number;
58      }
59
60      void info()
61      {
62          System.out.println(this);
63      }
64  }
65
66  public class Program
67  {
```

```
68      public static void main(String[] args)
69      {
70          System.out.println("-- to save --");
71          BufferedReader br=new BufferedReader(new InputStreamReader(
                System.in));
72
73          IDCard z=new IDCard(br);
74          z.info();
75
76          try
77          {
78              ObjectOutputStream outp=new ObjectOutputStream(new
                    FileOutputStream("file.dat"));
79              outp.writeObject(z);
80              outp.close();
81          }
82          catch(Exception e){System.out.println(e);}
83
84
85
86          System.out.println("\n-- from file --");
87          ObjectInputStream inp;
88
89          try
90          {
91              inp=new ObjectInputStream(new FileInputStream("file.dat"
                    ));
92              Object o=inp.readObject();
93              IDCard x=(IDCard)o;
94              inp.close();
95              x.info();
96          }
97          catch(Exception e){System.out.println(e);}
98      }
99  }
```

Listing 4: Example IV: object serialization

Ex. 4.6

Extend the above example. Identify and handle the exception that occurs when a file with the given name cannot be opened.

Ex. 4.7

Write, according to your own idea, a program for simulating the state of savings placed in several savings accounts and/or bank deposits over the course of several months of the year. Include aspects such as the current account balance, monthly account maintenance fee, interest capitalisation, and the so-called capital gains tax. To do this, implement, for example, an abstract class “Account” and interfaces “Interest” and “Tax”, as well as classes representing specific types of accounts and bank deposits, and write a program

that allows you to calculate the state of savings after several months. Implement appropriate algorithms for calculating interest and tax (you will find the necessary information easily on the Internet). Since floating-point arithmetic should not be used for financial applications, use an object of the `java.math.BigInteger` library class to store the account balance. Alternatively, instead of the above "financial" example, you can implement another example of your own idea, provided that you use inheritance and polymorphism mechanisms in it.