Programming paradigms

Learning Objectives

- You know the concept of a programming paradigm.
- You know what is meant by procedural and object-oriented programming.

A programming paradigm is a way of thinking about and structuring a program's functionality. Programming paradigms differ from one another, for example in how the program's execution and control are defined and what components the programs consist of.

Most programming languages that are currently in use support multiple programming paradigms. Part of a programmer's growth involves the ability, through experience, to choose the appropriate programming language and paradigm; there currently is no single ubiquitous programming language and programming paradigm.

The most common programming paradigms today are object-oriented programming, procedural programming, and functional programming. The first two of these are briefly discussed in what follows.

Object-Oriented Programming

In object-oriented programming, information is represented as classes that describe the concepts of the problem domain and the logic of the application. Classes define the methods that determine how information is handled. During program execution, objects are instantiated from classes that contain runtime information and that also have an effect c program execution: program execution typically proceeds through a series of method calls related to the objects. As mentioned a few weeks ago, "the program is built from small, clear, and cooperative entities."

The basic ideas of object-oriented programming, i.e., the representation of information and its processing methods with he help of classes and objects, first appeared in Simula 67, which was designed for developing simulations and the Smalltalk programming language. Its breakthrough came in the 1980s through the C++ programming language and Java has made it one of the most widely used programming paradigms in the world.

One of the major benefits of object-oriented programming is how problem-domain concepts are modeled through classes and objects, which makes programs easier to understand. In addition, structuring the problem domain into classes facilitates the construction and maintenance of programs. However, object-oriented programming is not inherently suited to all problems: for example, scientific computing and statistics applications typically make use of languages, such as R and Python.

Procedural programming

Whereas in object-oriented programming, the structure of a program is formed by the data it processes, in procedural programming, the structure of the program is formed by functionality desired for the program: the program acts as a step-by-step guide for the functionality to be performed. The program is executed one step at a time, and subroutines (methods) are called whenever necessary.

In procedural programming, the state of the program is maintained in variables and tables, and any methods handle only the values provided to them as parameters. The program tells the computer what should happen. As an example, the code below demonstrates the swapping of values for two variables a and b

```
int a = 10;
int b = 15;

// let's swap the values of variables a and b
int c = b;
b = a;
a = c;
```

When comparing object-oriented programming with procedural programming, a few essential differences emerge. In object-oriented programming, the state of an object can, in principle, change with any object method, and that change of state can also affect the working of the methods of other objects. As a consequence, other aspects of a program's execution may also be affected since objects can be used in multiple places within the program.

The difference between object-oriented programming and procedural programming are shown concretely in the clock example presented at the beginning of Part Five. The solution below depicts a procedural style where the printing of the time is transferred to a method.

```java
int hours = 0;
int minutes = 0;
int seconds = 0;

while (true) {
    // 1. printing the time
    print(hours, minutes, seconds);
    System.out.println();

    // 2. advancing the second hand
    seconds = seconds + 1;

    // 3. advancing the other hands when necessary
    if (seconds > 59) {
        minutes = minutes + 1;
        seconds = 0;

        if (minutes > 59) {
            hours = hours + 1;
            minutes = 0;

            if (hours > 23) {
                hours = 0;
            }
        }
    }
}
```

```java
public static void print(int hours, int minutes, int seconds) {
    print(hours);
    print(minutes);
    print(seconds);
}

public static void print(int value) {
    if (value < 10) {
        System.out.print("0");

    System.out.print(value);
}
```

The same implemented in an object-oriented way:

```
public class Hand {
    private int value;
    private int upperBound;

    public Hand(int upperBound) {
        this.upperBound = upperBound;
        this.value = 0;
    }

    public void advance() {
        this.value = this.value + 1;

        if (this.value >= this.upperBound) {
            this.value = 0;
        }
    }

    public int value() {
        return this.value;
    }

    public String toString() {
        if (this.value < 10) {
            return "0" + this.value;
        }

        return "" + this.value;
    }
}
```

```
public class Clock() {
    private Hand hours;
    private Hand minutes;
    private Hand seconds;
```

```java
    public Clock() {
        this.hours = new Hand(24);
        this.minutes = new Hand(60);
        this.seconds = new Hand(60);
    }

    public void advance() {
        this.seconds.advance();

        if (this.seconds.value() == 0) {
            this.minutes.advance();

            if (this.minutes.value() == 0) {
                this.hours.advance();
            }
        }
    }

    public String toString() {
        return hours + ":" + minutes + ":" + seconds;
    }
}
```

```java
Clock clock = new Clock();

while (true) {
    System.out.println(clock);
    clock.advance();
}
```

Programming exercise:

Liquid containers (3 parts)

Points

3/3

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Let's create an interactive program to control two liquid containers. The containers are named "first" and "second". Each are capable of containing 100 liters of liquid at a time.

The program offers the functionality to add, move and remove liquid. Using the "add" command will add liquid to the first container, "move" will move liquid from the first container to the second container and "remove" will remove liquid from the second container.

The commands must be defined as following:

- `add amount` adds the amount of liquid specified by the parameter to the first container. The inserted amount must be specified as an integer. The container can't hold more than a hundred liters and everything added past that will go to waste.

- `move amount` moves the amount of liquid specified by the parameter from the first container to the second container. The given amount must be specified as an integer. If the program is requested to move more liquid than than the first container currently holds, move all the remaining liquid. The second container can't hold more than one hundred liters of liquid and everything past that will go to waste.

- `remove amount` removes the amount of liquid specified by the parameter from the second container. If the program is requested to remove more liquid than the container currently holds, remove all the remaining liquid.

After every command the program will print the contents of both containers. You don't have to take negative values into consideration.

All the functionality must be added to the method `main` in the class `LiquidContainers` (do not create new methods). The template already contains a loop which exits the program with the command "quit".

A reminder of how to split a string below.

```java
String input = scan.nextLine();
String[] parts = input.split(" ");

String command = parts[0];
int amount = Integer.valueOf(parts[1]);
```

Part 1: Adding

Implement the functionality to add liquid to the first container. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
add 5

First: 5/100
Second: 0/100
add 25

First: 30/100
Second: 0/100
add 60

First: 90/100
Second: 0/100
add 1000

First: 100/100
Second: 0/100
add -5

First: 100/100
Second: 0/100
quit
```

Part 2: Moving

Implement the functionality to move liquid from the first container to the second. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
add 1000
```

```
First: 100/100
Second: 0/100
move 50

First: 50/100
Second: 50/100
add 100

First: 100/100
Second: 50/100
move 100

First: 0/100
Second: 100/100
quit
```

Second example:

Sample output

```
First: 0/100
Second: 0/100
move 30

First: 0/100
Second: 0/100
add 10

First: 10/100
Second: 0/100
move -5

First: 10/100
Second: 0/100
move 20

First: 0/100
Second: 10/100
move 10

First: 0/100
Second: 10/100
```

```
quit
```

Part 3: Removing

Implement the functionality to remove liquid from the second container. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
remove 10

First: 0/100
Second: 0/100
add 20

First: 20/100
Second: 0/100
remove 5

First: 20/100
Second: 0/100
move 15

First: 5/100
Second: 15/100
remove 5

First: 5/100
Second: 10/100
remove 20

First: 5/100
Second: 0/100
quit
```

Exercise submission instructions

How to see the solution

Programming exercise:

Liquid Containers 2.0 (2 parts)

Points

2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Let's redo the previous program for handling two liquid containers. This time we'll create a class "Container", which is responsible for managing the contents of a container.

Part 1: Container

Make a class called Container. The class must have a constructor which does not take any parameters, and the following methods:

- `public int contains()` which returns the amount of liquid in a container as an integer.
- `public void add(int amount)` which adds the amount of liquid given as a parameter to the container. If the amount is negative, no liquid is added. A container can hold a maximum of 100 units of liquid.
- `public void remove(int amount)` which removes the amount of liquid given as a parameter from the container. If the amount is negative, no liquid is removed. A container can never hold less than 0 units of liquid.
- `public String toString()` which returns the container as a string formatted "*amount of liquid*/100, for example "32/100".

The class should work as follows:

```
Container container = new Container();
System.out.println(container);

container.add(50);
System.out.println(container);
System.out.println(container.contains());

container.remove(60);
```

```
System.out.println(container);

container.add(200);
System.out.println(container);
```

Sample output

```
0/100
50/100
50
0/100
100/100
```

Part 2: Functionality

Copy the user interface you implemented for the previous example, and modify it to use the new Container class. The main method in the class `LiquidContainers2` must start the program.

Below is some sample output. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
remove 10

First: 0/100
Second: 0/100
add 20

First: 20/100
Second: 0/100
remove 5

First: 20/100
Second: 0/100
move 15

First: 5/100
Second: 15/100
```

```
remove 5

First: 5/100
Second: 10/100
remove 20

First: 5/100
Second: 0/100
quit
```

Exercise submission instructions

How to see the solution

Quiz:

Pros and cons of different approaches

Points:

1/1

We examined solving the same problem with two different approaches with the exercises 'Liquid container' and 'Liquid container with object' above. Consider and describe what (dis)advantages you see with the approaches we took. Also tell which of the alternatives you found more intuitive.

Your answer should be at least 30 words

Your answer

in the first approach, it was mentioned to create move method first before remove method which is confusing because remove first would be better than move as move method requires the remove method.

Words: 33

Answered The number of tries is not limited

You have reached the end of this section! Continue to the next section:

→ 2. Algorithms

Remember to check your points from the ball on the bottom-right corner of the material!

In this part: