Chapter 4 Object-oriented design with Java

Readers of this book who take an interest in the wider IT world may have heard of Object-Oriented Programming or OOP. Whereas the first chapter dealt with the basics of programming, controls structures and syntax this chapter deals with better organizing code. As you may guess they are organized into Objects. After you finish this part of the book you may decide to continue to hone your programming skills. Perhaps you are looking for a new line of work. Learning about Java and OOP are an excellent idea. Both are popular in the business world: the simple syntax along with organizing code along the way business people think has made both popular.

Java

The Java language, which is at this moment the most popular in the world, was designed by James Gosling in 1995 at Sun Microsystems. It is heavily influenced by C, but Gosling wanted to make the language simpler to use. Java makes extensive use of libraries, thus removing the need to type the nitty-gritty details of many applications. Java also has automatic garbage collection. Unused memory is automatically given back to main memory. Last but not least Java can run on pretty much any computer, without the need to recompile the code for that computer. So if I compile my Java code on Windows the compiled code can also run on Linux, or an ARM based computer. The caveat is that those other systems must also be able to use the Java Virtual Machine, normally contained in a JRE (Java Runtime Environment). Code in Java is not compiled from source to machine code; instead it is compiled into object code. The JVM will at the start of a Java program transform this code to machine code, in real-time, there is no waiting step to compile. This has advantages and disadvantages. The advantage is that Object code can run anywhere, as explained. The major disadvantage is that Java can be slower than other language, especially C. However, with the introduction of the Java Just-In-Time compiler (JIT) the major speed disadvantages have disappeared leaving Java faster than many other languages except C.

So let's start with a simple HelloWorld example. With Gedit open a file named HelloWorld.java and type in the following lines of code

```
public class HelloWorld {
    // start of the program
    public static void main(String[] args) {
        System.out.println("Hello World!"); // print statement
    }
}
```

In the first line we define a class. Java does everything with classes. It is a simple way to divide code into organized places. The third line, beyond the comment, is the official start of the program. Every Java program needs to have one otherwise the computer would not know where to start the program. It is similar in nature to C.

Java needs to be compiled into object code. So you need to ensure a SDK is installed. A standard version would be JAVA SE 8. To test whether you have installed it properly type in javac –version. Output should be something like javac 1.8.0_60. You can compile the code with the command

```
Javac HelloWorld.java
```

Now you should also have a file called HelloWorld.class

To run the class file you need to have the Java Runtime installed. Type in java –version to check. If the output finds a JRE you are all set. Now you can run the class file (object code). Type in

```
java HelloWorld
```

And your output should be *HelloWorld!*

Objects

So what is an Object? To put it bluntly, it is a custom made variable. Where you already know some programming languages have variables such as integers, floats and strings you have not dealt with the possibility that things in this world, around us, may not be easily defined by any of them. As an example, take your own identity. If you write a program to store and manipulate identities you have to deal with names, age, place of residence etc. Each can be defined with a classic variable that I just mentioned, but only taken together are they truly an identity. What if you could take all three variables and place them in a variable called Identity? They would be where they belong with no fears of causing spaghetti code as you try to maintain endless records of people whom each may have dozens of variables. Here Identity is an object. In java, we deal with classes, each class is a blueprint for an object. If I want to change the name of a person I change it using methods found in that class that are suitable. This way what truly defines an Identity are not just the variables but also the functionality given to the Object by the methods. I know what I just explained may be overwhelming so let me show you.

First off, we need a class that can start the program. The code you see below will suffice, I called it StartApplication. It contains the line public static void main, which some have called the GodMethod because it is where the entire program sequence begins. After the start the first true line of the program is line 3. See how it creates an instance of a new variable of type Hotel. The instance is called hotel. This is similar to creating a instance off String string = “”;. That said, String is part of Java and already defined, but the program has no idea what a Hotel is. Java will look for other class files in the same map where we have defined in a new file the class Hotel. See below

```
public class StartApplication {
    public static void main(String[] args) {
        Hotel hotel = new Hotel("Fawlty Towers", "Basil and Sybil");
        hotel.run();
    }
}
```

The previous piece of code creates a Hotel instance but on the right side it also does something strange. The section new Hotel("Fawlty Towers", "Basil and Sybil"); may seem alien at first, but what it essentially does is pass along two strings in the creation of hotel. Now our unique variable Hotel already has two interesting details, a name and an owner. Both strings were passed on to what is known as the Constructor. The constructor is always named similarly to the class but without the word class before it. The constructor is passed two strings, name and owner. It is up to the programmer to understand what they mean. We place both values into global variable that are similarly named, but that is not necessary. The variable could have been called cat and mouse but it wouldn't make much sense. Yet the constructor does more. After it assigns the two strings to the global variables it also sets the integer variable numberOfRooms to 26 and the Boolean isOpen to true.

```
public class Hotel {
    String name = "";
    String owner = "";
    int numberOfRooms;
    boolean isOpen;

    public Hotel(String name, String owner) {
        this.name = name;
        this.owner = owner;
        this.numberOfRooms = 26;
        this.isOpen = true;
    }

    public void run() {
        while (isOpen == true) {
        }
    }
}
```

Now that the hotel instance was created the program can move to the second line. Remember I said you can assign functionality to an object. Well, in the class file Hotel we have a method called run. Only hotel instances can run this method and we have just created one. We call the method run on line 4 of the previous code section. Afterwards the program ends. So if we want to give our little example any kind of functionality it has to happen in method run. Below is an extended section of the method plus an additional method that is called.

```
public void run() {
    while (isOpen == true) {
        Scanner user_input = new Scanner( System.in );
        System.out.print("Should Basil close the hotel? Yes or No!");
        String decisions;
        decisions = user_input.next();

        if (decisions.equals("Yes")) {
            closeHotel();
            System.out.print("The hotel will be closed");
        }
    }
}

public void closeHotel() {
    this.isOpen = false;
}
```

You may already guess what it does. The while loop in run will last until the boolean isOpen is set to false. We are going to try to do that by asking input from the user. The first line in the while loop creates a Scanner Object called user_input. It is created by sending the argument System.in into its constructor. Don't worry about memorizing all this. I usually look at previous code examples that I have written. You can do that as well. The next interesting line is decisions = user_input.next(); this asks for user input and places that into the String called decisions. Next we have a conditional statement that checks if our input is equal to the String word Yes. If the user filled in anything else, even nonsense it would never evaluate to true. If the user has inputted Yes the conditional statement is entered and the method closeHotel is called. This method is depicted below and should be part of the class Hotel. All it does is switch the Boolean value isOpen around. We could have called the method from the other class where we have a Hotel instance called hotel. The method is public. If the method is set to private instead than it can only be called from within the class itself. So Objects have properties that are more than just variables, they also have functionality.

Note that we have written two classes. You must compile both files, use the java *.java to compile all java source files. However, only the object code for StartApplication needs to be called to start the application as it contains the line static void main.

Overloading

Now that you understand that all Java classes (blue-prints for Objects) inherit from class Object the sticky issue arises of what to do if a method already exists, but you do not have exactly the right input to use it. With method overloading you can write different versions of the same method suitable for different types of input. I want to perform calculations on my bank account, but instead of Double values I also want to use non-decimal Integer values. To overwrite my current balance I can write two methods.

```
public void newBalance (int balance) {
    this.balance = balance;
}
public void newBalance (double balance) {
    balance = (Integer) balance; // cast the double to int
    this.balance = balance;
}
```

If another part of the program which I do not have control over (not unusual in large programming applications) either gives me a double or an integer I do not have to change my code. I have to call function newBalance just once and either version will be called depending on what the input value is. This takes complexity out of my hands. In fact, it is well possible to also overload Constructors. If I either have a double or an integer value and I want to open a new bank account I can write two constructors.

Polymorphism

A related issue to overloading and overwriting is polymorphism. What this means is that certain parts of code will accept different types of Objects as long as they are related. Whether or not a program will get a particular Object has not been decided at the moment it was written but is instead decided at run-time? As an example, a person can be a male or female (leaving out third-gender for clarity). If you're using the program and it asks for your gender, it can create a gender object that is either male or female. Instead of using control statements to decide what to do in either case you can just pass the object to where you want it to go. For both to be similar they need to be subtypes of a class that need to be accepted by the program. You can imagine that organizations and businesses prefer Object design as it allows them to clearly demarcate functionality. At times it does become more complex. I hope you have learned how to organize applications. Enjoy!