Co-creation

Programming - memo

In real life we talk about **concepts**, but we use **things**. In Java we define **classes**, but we use **objects**.
When **co-creating** in Java we will identify **responsibilities**. We will **create** classes that will **take care** of those responsibilities. We will use those classes **understanding what** they can do, but **not caring how** they can do it.

How to define a class

A class is a **concept** that **describes** the object. The **object** is what we use. We **define** a class by explaining **who** it is, **how** it looks like and **what** it can do. This is represented by the **name**, the **attributes** and the **methods**.

- We write **public class**
- We define its **identity** with a name
- The name starts with a **capital letter** and is a **noun**
- We define its **state** with attributes
- We define its **behavior** with methods

```
public class Person {

  private String name;
  private Integer age;

  public void think(){
     System.out.println("I think, therefore I exist.");
  }
}
```

How to build an object

- When we build a **new** object we need a **constructor**
- A **constructor** is a method with no return type that has the same name as the class
- If we don't provide one, the **empty** or **default** one is used
- It is the method used to actually create a new object of that class
- If the class needs something in order to **fulfill its purpose** (attributes), then we provide it as **arguments** with the **constructor**
- We can have more than one constructor but having **only one** is recommended.

```
public class Reader {

  public void read(Book book) {
     System.out.println("Reading " + book.getName());
  }
}
public class Reader {

  public Reader() {} // added automatically

  public void read(Book book) {
     System.out.println("Reading " + book.getName());
  }
}
```

- If the class needs something in order to **fulfill its purpose** (attributes), then we provide it as **arguments** with the **constructor**
- We can have more than one constructor but having **only one** is recommended.
- The **this** keyword differentiates its **own** attribute from an **argument** that have the same name

```
public class Book {

  private String name;

  public Book(String name) {
     this.name = name;
  }

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

How to use an object

- We create a **new** one using the new keyword
- We **reference** it with a **variable**
- We **use** it exactly the same as **other variables**
- Classes are in reality **data types** like Integer or String that we are able to **co-create**

```java
public class LibraryApplication {

  public static void main(String[] args) {
    Library library = new Library();
    Book book = library.getBook("Siddhartha");
    Reader reader = new Reader();
    reader.read(book);
  }
}
```

The two kinds of objects

Programming is a process of **data manipulation**. Classes either **manipulate data** or **represent data**. Classes that manipulate data are called **agents**. Classes that represent data are called **data**.

Agents

- Agents **manipulate data**
- They **often** have **no attributes**
- If they have attributes they are usually **references** to **other agents**
- Their **methods** are more important than their **attributes**
- What they can **do** is more important that what they **are**
- Other names are services or controllers.

```java
public class Reader {

  public void read(Book book) {
    System.out.println("Reading " + book.getName());
  }
}

public class Library {

  public Book getBook(String name) {
    return new Book(name);
  }
}
```

Data

- Data represents the **information** our solution manipulates
- They **often** have **no special methods** other than sharing the information they contain
- Their **attributes** are more important than their **methods**
- What they **are** is more important that what they can **do**
- Other names are data transfer objects (DTOs) or plain old Java object (POJOs)

```java
public class Book {

  private String name;

  public Book(String name) {
    this.name = name;
  }

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

The Single Responsibility Principle

A class should have **only one** reason to **change**. In the programming world code is **never written**, it is **always rewritten**. When a **change** is needed, ideally we want to change as **few modules** as possible.

If our classes have **only one responsibility**, when a change is needed we will modify as **few classes as possible**. The code for **other responsibilities** is **never affected** this way.