Objects in JAVA^{TM}

Imagine a database of students . . .

```
public class Students_1
{
    public static void main(String[] args)
        {
            int i = 10;
            String name[] = new String[i];
            int number[] = new int[i];
            int grade[][] = new int[i][5];
            int GPA[] = new int[i];
            ...
        }
}
```

Students_1.java

Although this is a perfectly valid programming approach, wouldn't be great if we could group all those fields in one item ?

```
class student
{
     String name;
     int number, GPA, grade[] = new int[5];
}


     ...
     int i = 10;
     student table[] = new student[10];
     ...
```

Objects are bundle of variables and related methods. Everything that an object knows (state) and can do (behavior) is expressed by the variables and methods within that object.

- Modularity :*Decompose* problems into smaller sub-problems.

- Information hiding : to *hide* implementation details. For example, our *student* object could easily contain a method to calculate the average.

```
class student
{
    String name;
    int number, GPA, grade[] = new int[5];

    int average()
        {
            int sum = 0;
            for (int i = 0; i < grade.length; i++)
                sum += grade[i];

            return sum / grade.length;
        }
}
```

In java, objects are defined through a *class* definition. To create an object from it we use the *new* operator, this process is called an instantiation. It causes RAM to be dynamically allocated and the constructor called to initialize the object. A class, and therefore each instance of it, will have the following content :

- *instance variables* : a set of variables unique to each instance of the class.
- *constructors* : A special-set of methods called when the object is created.
- *methods* : Methods that are logically linked to the data in the object.

```
class student
{
     String name;
     int number, GPA, grade[] = new int[5];

    student(int student_id)
        { // The constructor
            name = "undefined";
            number = student_id;
        }

    int average()
        { // A method that sums the grades
            int sum = 0;
            for (int i = 0; i < grade.length; i++)
                sum += grade[i];

            return sum / grade.length;
        }
}

public class Students_3
{
    public static void main(String args[])
        {
            int i = 10;
            student table[];
            table = new student[i]; // The creation of the array;

            table[0] = new student(9988777); // creation of one object;
            table[0].name = "alpha";
            table[0].grade[0] = 70;
            table[0].grade[1] = 80;
            table[0].grade[2] = 75;
            table[0].grade[3] = 65;
            table[0].grade[4] = 85;

            System.out.print("The average of " + table[0].name);
            System.out.println(" was " + table[0].average());
        }
}
```

While a method is defined with :

- name

- return value : if return is not of type `void` then all paths of your method must include a `return` *xyz* statement.

- list of arguments

A constructor :

- Must have the same name as that of the class.

- Doesn't have a return value.

- May also have a list of arguments

Overloading methods and constructors :

```
class student
{
     String name;
     int number, GPA, grade[] = new int[5];

    student(int student_id)
        { // The constructor
            name = "undefined";
            number = student_id;
        }

    student(String student_name)
        {
            name = student_name;
        }

    int average()
        { // A method that sums the grades
            int sum = 0;
            for (int i = 0; i < grade.length; i++)
                sum += grade[i];

            return sum / grade.length;
        }

    void set_GPA()
        {
            GPA = average();
        }

    void set_GPA(int value)
        {
            GPA = value;
        }
}
```

References :

When an object is created with the *new* construct, we say that there exists a reference to it. In the example

```
...
student table[];
table = new student[i];
table[0] = new student(9988777);
...
```

all of the cells of the array `table[]` hold references to objects, however `table[0]`'s reference actually points to an item in memory that has been created. The other cells point to the special type `null`.

You may want to think of a reference as a *pointer* or a *handle* to the actual area in memory where the object is stored.

References are used mainly as :

- Qualifying names to access fields or to call methods.
- If of type `String`, with the + operator for concatenation.
- As the operand of the `instance of` operator.
- With the reference equality operators (== and ! =).

Null :

- Reserved word
- You may not declare a variable of type `null`.
- The `null` type can be casted to any reference type (arrays, class, . . .).
- It is the default value of an uninitialized reference.
- Any reference can be compared to `null` for equality or inequality.

If you are thinking of references as *pointers* to area in the memory of your computer, then `null` would indicate that your reference is not pointing to any area. In `C`, `C++` and a few other languages, this is called the *zero pointer*.

Java is considered a strongly typed language: the compiler knows the type of all variable at any position in the code. These can be grouped in three categories :

- Primitive
  Includes numeric types (like `int` or `double`) and the boolean type.

- Reference
  These include class references, interface references and array references.

- `null`
  The value assumed by references that have not been initialized (instantiated).

RAM is regained by the operating system whenever an object that was created by the `new` operator is not referenced by any variable. The concept of the language deallocating memory is called *garbage collection*.

JAVA comes with a set of objects that offer a multitude of functionalities. These classes are grouped together in *packages*. To tell the compiler that you intend to use these, you need to use the `import` operator. By default the package `java.lang.*` is always included.

For example, if I wanted to create an empty window for some user interface :

```
import java.awt.*;

public class My_Frame
{
    public static void main(String[] arg)
        {
            Frame display;

            display = new Frame("Mine");
            display.show();
        }
}
```