Variables and Data Types

At this point we know Java is a strongly typed language. That is, we must give a type to each variable we declare in our code. At the same time, when we declare variables in some part of the code, these will be visible to other parts of the code depending on where each variable was declared. This “visibility” is called the “scope” of a variable.

The scope of a variable dependents mostly on the location of the variable declaration. For instance, a “local” variable might be a variable defined inside a method whereas an “instance” variable or a “class” variable is defined outside methods. Local variables are the most common, across languages and are defined inside functions, control structures, etc.

Local variables

In Java, local variables are those defined inside a *code block*. In Java, code blocks are defined using curly braces « { and } ». Therefor, a variable declared between an opening brace « { » and a closing brace « } » will only be visible inside these. For example, let’s suppose we have a class Person, that has among its attributes the date of birth. This way, we’ll have persons (objects of type Person) and each one will have its own date of birth. Each Person should have an operation that emits the name, which answers the question “what’s your name?”. At the same time, the Person might have another similar operation, like “tellAge” which answers the questions “how old are you?”. Thus, I could define those operations like so:

```
public  class Person {

  private String name;

  private Date dateOfBirth;

  public void tellName() {

    String message = “I am ” + name;

    System.out.println(message);

  }
  public void tellAge() {

    Date today = new Date(); // new Date() gives the current date

    // just an example, in java we cannot just "subtract" date

    int age = today - this.dateOfBirth;

    System.out.println("I'm " + age);

  }

 }
```

In this code, the variable "message" will never be accessible inside the method "tellAge", neither the "today" variable would be accessible inside the "tellName" method. This is because "message" **is a local variable corresponding to the code block of the "tellName" method**. The same happens with the "today" and "age" variables. In Java, variables defined inside a code block are local to that lock and cannot be accessed outside that block. A variable defined between braces lives and dies inside that block only. Lets see the following code:

```
public class Test {

  public static void main(String[] args) {

    int outer = 1;

    if(outer == 1) {

        int local = 3 + outer; // outer

        System.out.println(local); // imprime 4

    }

    // local is not accessible here. Compile error.

    System.out.println(local);

  }

}
```

This is just an example, as this if block does not have any practical use (outer is always equal to one) but helps to make the point of local variables. What we *can* do is access a variable defined *outside* a code block. Something like this:

```
public class Test {

  public static void main(String[] args) {

    int outer = 3;

    for(int i=0; i < 5; i++){

        int local = i + outer; // outer is accessible here

        System.out.println(local);

    }

    // outer is still accessible here.

    System.out.println(local); // 3 4 5 6 7

  }

}
```

Instance variables

Instance variables are variables that an object has access to. Previously, we defined a class Person, with two attributes: age and name. This means that each time we instantiate the class Person and obtain new objects of type Person, each one will have its own value for each attribute. Thus, each object (each person) will be able to tell its own age and its own name to the world through the tellName and tellAge operations. Instance variables are accessible anywhere on the code inside the class (and possible outside the class but only through the object, more on that later). That is the case of the “name” variable, which we can use directly inside the method tellName. Although the variable (in this case call an attribute) “name” is defined at the top we can use it directly (in fact, it is the same case as the “outer” variable example, is it outside the tellName method block). If we look closely at dateOfBirth in the above code, we used the keyword **this**. There, **this** does not have any other effect than to highlight the fact that we are using the *attribute* and we know at a glance that it is not a local variable but a variable that belongs to the object, and *instance variable*.

If we had declared the instance variable as **public** (a practice that defeats the purpose of encapsulation) we could access that variable from the outside, although only *through* the object. For example:

```
public class Test {

  public static void main(String[] args) {

    Person p = new Person();

    p.name = “Mike”; //should not be used like this

  }

}
```

Class Variables

Class variables constitute a subject with a little bit more complexity and should be analyzed in more depth, but for now, let’s just say that they hold values that go beyond objects or instances. That is, they hold the same value across all instances of the same class. The typical use is to hold constant, global values. The important thing here is the keyword static, which tells the variable (attribute) to be part of the class (and not the objects produced from that class). For example:

```
public class Person {

  public static int AMOUNT_OF_HEARTS = 1;
}
```

Although a somewhat silly example, we all know that a person has only one heart. That is a *constant* among all people. So we could do something like this:

```
public class Test {

  public static void main(String[] args) {

    Person p = new Person(); //p holds a Person object

    Person p2= new Persona();//p2 holds *another* Person object

    System.out.println(p.AMOUNT_OF_HEARTS)// prints “1”

    System.out.println(p2.AMOUNT_OF_HEARTS)// prints “1” as well

    //in fact, AMOUNT_OF_HEARTS is a class variable or class attribute

    //therefor we can (and should) access is through the class:

    System.out.println(Person.AMOUNT_OF_HEARTS)// prints “1”

  }

}
```

In the case of instance variables and class variables, we can somewhat “break” the rule of the braces, but they still apply inside the code of the class. However we can only access those variables *through* the object or the class, respectively. These variables are, in fact, inside a code block on their own, the code block for the class.

Naming

We already know that each variable declaration must have its data type defined. All **local variable names in Java, as a convention, start with lowercase**. And then, each word we add to the name, starts with uppercase. For example:

String userEmail = “some@email.com”.

It is worth noting that methods follow the same convention, typically called “Pascal Case” since it was first introduced with the Pascal programming language. Classes however, start with uppercase and each word added uses uppercase as well (i.g. SomeTestClass). This is called “came case”.

Class variables, in turn, are all named in uppercase, and each word added to the name is separated using an underscore. This is referred to as snake case, in this case “upper snake case. For example: static int NUMBER_OF_WHEELS;