VARIABLES AND DATA TYPES IN JAVA

A variable in Java can hold some data. We need to declare the type of a variable and the variable can only hold a compatible type of data. Declaring the type of a variable is like introducing the variable to your program, and you should only introduce once.

Declaration of a variable

The declaration of a variable begins with optional access modifier, the data type and is followed by the variable name and then a semicolon.

public **int myVar**;

- Here, **public** is the access modifier and **int** is the data type of the variable named **myVar**.
- Access modifiers tell which all code can access this member and access modifier public tell that this variable can be accessed from any other class.
- The **data type** is the type of value that a variable can hold. The data type may be a primitive data type or a reference data type.

Primitive type

1. The data type may be a primitive data type or a reference data type.
2. A **primitive data type** is a basic type like int, float, double etc. and they hold a literal directly.
3. There are eight primitive data types defined in Java: boolean, byte, char, short, int, long, float, double.

Example: Primitive type

```
int i=5;
```

- The primitive type int (primitive type for integer) tells that this variable can hold only integers.

Casting of types

1. Compatible types can be implicitely or explicitely casted to each other.
2. Except boolean, all other types are compatible with each other. A boolean can be assigned only a true or false or an expression that evaluate to true or false.

Example: Valid boolean assignments

```
boolean b1 = true;

boolean b2 = false;
```

boolean b3 = (3 < 5) ;

Example: Invalid boolean assignments
~~boolean b4 = 0;~~

~~boolean b2 = 1;~~

Reference type

1. When an object is created using new keyword, enough memory is allocated in memory to hold all its members. A **reference type** variable holds the address of an object.
   Example: Reference type
   Consider a class MyClass:

   We can create two objects of the class MyClass and assign it to the reference type as:

   public **MyClass myObject1** = new MyClass();
   public **MyClass myObject2** = new MyClass();

   Here MyClass is the reference type and myObject1 and myObject2 are reference type variables that refer to an object of type MyClass, returned by "new MyClass()".

Pass By value vs. Pass By Reference

1. When you copy variables or when you pass a variable to a method in Java, always, the value is passed: Java will pass whatever the variable holds.

2. In the case of a primitive, Java passes the value of the variable and in the case of a reference type variable, it passes the address (reference) that is holds.

   Example: Pass by value demonstration
   myObject1.val=5;

   myObject2 = myObject1;

   myObject2.val = 10;

   System.out.println(myObject1.val);

   Output will be:

   10

- This is because, after the statement 'myObject2 = myObject1;' both reference type variables point to the same object in memory.

- The object previously pointed by myObject2 doesn't have any reference to it and hence it will be eligible for garbage collection.

Null Literal and NullPointerException

Any reference type can be assigned with a special literal called null.

This means that the reference is not pointing to an actual object of its type.

If you try to invoke a method on a null object, you will get NullPointerException.

Assigning child object to a parent reference

A parent type (class or interface) can refer to any of its children's objects.

So below assignments are valid:

```
interface Abc{}

class Xyz{}

class Def implements Abc extends Xyz{}

...

Abc var1 = new Def();

Xyz var2 = new Def();
```

Class variables, Instance Variables and Local variables

Based on the scope in which the variable is declared, we can classify variables as Class variables, Instance Variables and Local variables.

Class variables (or static variables)

1. Class variables (or static variables) are declared within a class and outside all methods using the **static** keyword.
2. They belong to class and are common to all objects of that class.
3. They can be called using class name as MyClass.myStaticVar or using any of the object instances as myObjOfMyClass.myStaticVar.

Example: Static variable

```
class MyClass{

static int myVar;
```

}

Instance variables

1. Instance variables are non-static variables and are defined in a class outside any methods.
2. These variables are specific to each instance (object) of the class.
3. They can be called using any of the object instances as **myObjOfMyClass.myInsVar**.
   Example: Static variable

```
class MyClass {

int myVar;

}
```

Local variables

1. Local variables are local to methods or blocks in a class.
2. Local variables are declared the same way as instance variables, but are declared within a method or block.
3. A local variable must be initialized before you try to use it, else compiler will complain.
4. Their scope is limited to the block or method they are declared and die as soon as the block is exited.
5. Local variables are always thread safe as a different copy of local variable will be created per thread.
6. We can declare a local variable with the same name as an instance or static variable declared in a class and when accessed from within the local block (a method), local variable will take precedence over others.
7. We cannot redeclare a local variable again inside an inner block where that local variable is visible. This is because we have already introduced that local variable and there is no need to introduce it again.

Explicit and default initialization of variables

- **Default initial values** - A variable is given a default value according to its type when declared simply declared like **int myVariable;**. *Local variables are not initialized with default values.*
- **Explicit initialization** along with declaration as **int myVariable = 5;**
  You can initialize variables explicitely using initialization blocks or through constructors.

Final variables (or contstants)

- A variable declared using the final keyword will act as a constant and cannot be changed.
  - For a primitive variable, this means that the primitive value contained in the variable cannot be changed.
  - For a reference variable, this means that the reference variable cannot refer to any other object, but the referred object can still be modified.

Valid identifiers in Java

A **variable** name is also called an identifier and identifiers should follow below rules:

1. identifiers are case-sensitive
2. identifiers can only be composed of Letters, numbers, the underscore (_) and the dollar sign ($)
3. identifiers may only begin with a letter, the underscore or a dollar sign

Exercise to increase your understanding

1. Create a class MyClass with one instance variable myInsVar and one static variable myStaVar and initialize both to 0.

```
class MyClass{

int myInsVar=0;

static int myStaVar=0;

}
```

2. In the main method of the same class, create two objects of MyClass.

```
MyClass obj1 = new MyClass();

MyClass obj2= new MyClass();
```

3. Increment both the variables on the objects as below:

```
obj1.myInsVar++;

obj1.myStaVar++;

obj2.myInsVar++;

obj2.myStaVar++;
```

3. First calculate the output looking at the below code (without actually executing) and then execute to see if your findings were correct:

```
System.out.println(obj1.myInsVar);

System.out.println(obj1.myStaVar);

System.out.println(obj2.myInsVar);

System.out.println(obj2.myStaVar);
```

Did you get the output as you expected? If yes, you have understood. Else remember that myStaVar is static and is common for all objects and there is only one copy per class, but there is a copy of myInsVar per object. Now try to understand it again. Still not able to understand, please feel free to ask me.

Source : http://javajee.com/variables-and-data-types-in-java