

Java - Variable Types

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration –

```
data type variable [ = value][, variable [ = value] ...] ;
```

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java –

Example

```
int a, b, c;      // Declares three ints, a, b, and c.
int a = 10, b = 10; // Example of initialization
byte B = 22;      // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';     // the char variable a is initialized with value 'a'
```

This chapter will explain various variable types available in Java Language. There are three kinds of variables in Java –

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

```
public class Test {  
    public void pupAge() {  
        int age = 0;  
        age = age + 7;  
        System.out.println("Puppy age is : " + age);  
    }  
  
    public static void main(String args[]) {  
        Test test = new Test();  
        test.pupAge();  
    }  
}
```

This will produce the following result –

Output

Puppy age is: 7

Example

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test {  
    public void pupAge() {  
        int age;  
        age = age + 7;  
        System.out.println("Puppy age is : " + age);  
    }  
  
    public static void main(String args[]) {  
        Test test = new Test();  
        test.pupAge();  
    }  
}
```

This will produce the following error while compiling it –

Output

Test.java:4:variable number might not have been initialized

age = age + 7;

^

1 error

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Example

```
import java.io.*;
public class Employee {

    // this instance variable is visible for any child class.
    public String name;

    // salary variable is visible in Employee class only.
    private double salary;
```

```

// The name variable is assigned in the constructor.
public Employee (String empName) {
    name = empName;
}

// The salary variable is assigned a value.
public void setSalary(double empSal) {
    salary = empSal;
}

// This method prints the employee details.
public void printEmp() {
    System.out.println("name : " + name );
    System.out.println("salary : " + salary);
}

public static void main(String args[]) {
    Employee empOne = new Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
}
}

```

This will produce the following result –

Output

```

name : Ransika
salary :1000.0

```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.

- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

```
import java.io.*;
public class Employee {

    // salary variable is a private static variable
    private static double salary;

    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development ";

    public static void main(String args[]) {
        salary = 1000;
        System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

This will produce the following result –

Output

Development average salary:1000

Note – If the variables are accessed from an outside class, the constant should be accessed as Employee.DEPARTMENT

What is Next?

You already have used access modifiers (public & private) in this chapter. The next chapter will explain Access Modifiers and Non-Access Modifiers in detail.

References

- www.tutorialspoint.com/java