CHP-5
Java Basic Data Types

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.
There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Let us now look into detail about the eight primitive data types.

byte:

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100, byte b = -50

short:

- Short data type is a 16-bit signed two's complement integer.

- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767(inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example: short s= 10000, short r = -20000

int:

- int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.

 The default value is 0.
 Example: int a = 100000, int b = -200000

long:

 Long data type is a 64-bit signed two's complement integer.
 Minimum value is -9,223,372,036,854,775,808.(-2^63)
 Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
 This type is used when a wider range than int is needed.
 Default value is 0L.
 Example: int a = 100000L, int b = -200000L

float:

 Float data type is a single-precision 32-bit IEEE 754 floating point.
 Float is mainly used to save memory in large arrays of floating point numbers.
 Default value is 0.0f.
 Float data type is never used for precise values such as currency.
 Example: float f1 = 234.5f

double:

 double data type is a double-precision 64-bit IEEE 754 floating point.
 This data type is generally used as the default data type for decimal values, generally the default choice.
 Double data type should never be used for precise values such as currency.
 Default value is 0.0d.
 Example: double d1 = 123.4

boolean:

 boolean data type represents one bit of information.
 There are only two possible values: true and false.
 This data type is used for simple flags that track true/false conditions.
 Default value is false.
 Example: boolean one = true

char:

 char data type is a single 16-bit Unicode character.
 Minimum value is '\u0000' (or 0).
 Maximum value is '\uffff' (or 65,535 inclusive).
 Char data type is used to store any character.
 Example: char letterA ='A'

Reference Data Types:

 Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
 Class objects and various types of array variables come under reference data type.
 Default value of any reference variable is null.
 A reference variable can be used to refer to any object of the declared type or any compatible type.
 Example: Animal animal = new Animal("giraffe");

Java Literals:

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.
Literals can be assigned to any primitive type variable. For example:

```
byte a =68;
char a ='A'
```

byte, int, long, and short can be expressed in decimal(base 10),hexadecimal(base 16) or octal(base 8) number

systems as well.
Prefix 0 is used to indicate octal and prefix 0x indicates hexadecimal when using these number systems for literals.
For example:

```
int decimal=100;
int octal =0144;
int hexa =0x64;
```

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are:

```
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

String and char types of literals can contain any Unicode characters. For example:

```
char a ='\u0001';
String a ="\u0001";
```

Java language supports few special escape sequences for String and char literals as well. They are:

Notation	**Character represented**
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	Tab
\"	Double quote
\'	Single quote
\\	Backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

What is Next?

This chapter explained you various data types, next topic explains different variable types and their usage. This will give you a good understanding about how they can be used in the Java classes, interfaces, etc.

CHP-6
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. The basic form of a variable declaration is shown here:

```
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:

```
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 iis 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 this method only.

```
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 would produce the following result:

```
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 would produce the following error while compiling it:

```
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 and different class (when instance variables are given accessibility) 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 would produce the following result:

```
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 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 variables 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 would produce the following result:

```
Development average salary:1000
```

Note: If the variables are access 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 you Access Modifiers and Non Access Modifiers in detail.