# OOP-Unit-2 Notes

## Variable

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location.

### Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

#### *1) Local Variable*

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

#### *2) Instance Variable*

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

#### *3) Static variable*

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

## Example to understand the types of variables in java

```
public class A

{

   static int m=500;//static variable

   void method()

   {

      int n=10;//local variable

   }

   public static void main(String args[])

   {

      int data=40;//instance variable

   }

}//end of class
```

## Variables Dynamic Initialization in Java

If any variable is not assigned with value at compile-time and assigned at run time is called dynamic initialization of a variable. Basically, this is achieved through constructors, setter methods, normal methods and builtin api methods which returns a value or object.

## Scope and lifetime of variables in Java

### Instance Variables

A variable which is declared inside a class and outside all the methods and blocks is an instance variable. The general scope of an instance variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory.

### Class Variables

A variable which is declared inside a class, outside all the blocks and is marked static is known as a class variable. The general scope of a class variable is throughout the class and the lifetime of a class variable is until the end of the program or as long as the class is loaded in memory.

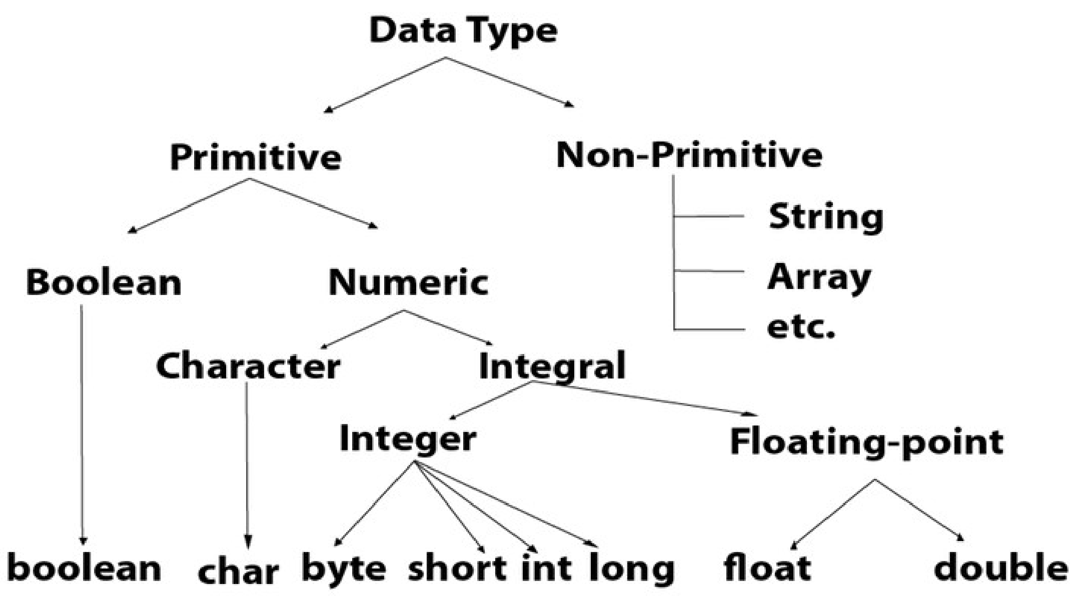
## Local Variables

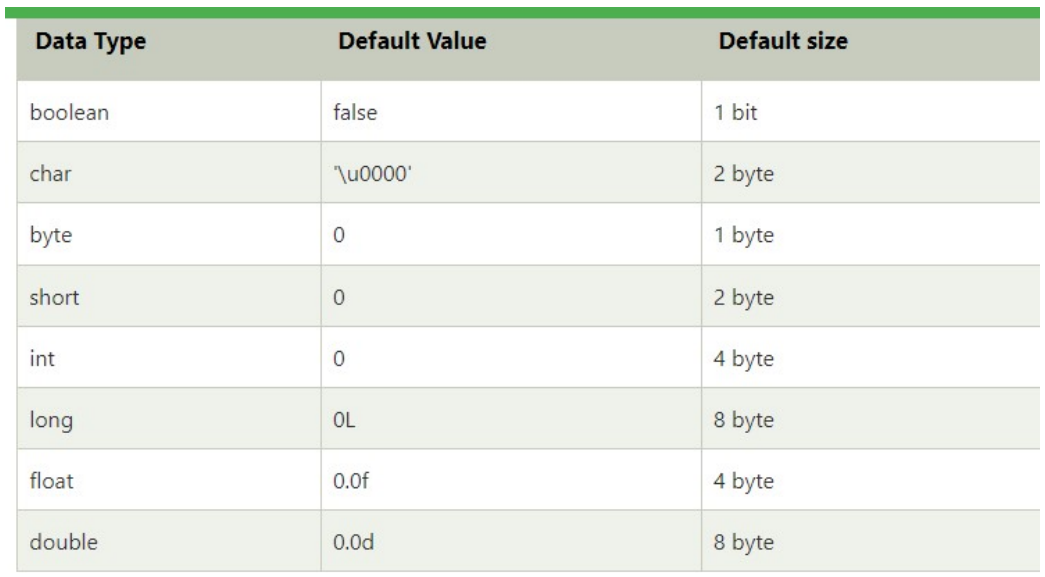
All other variables which are not instance and class variables are treated as local variables including the parameters in a method. Scope of a local variable is within the block in which it is declared and the lifetime of a local variable is until the control leaves the block in which it is declared.

## Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.





| Data Type | Default Value | Default size |
|---|---|---|
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

## Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:**

```
Boolean one = false
```

## Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:**

```
byte a = 10, byte b = -20
```

## Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:**

```
short s = 10000, short r = -5000
```

## Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**Example:**

```
int a = 100000, int b = -200000
```

## Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

**Example:**

1. `long a = 100000L, long b = -200000L`

## Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:** **float** f1 = 234.5f

## Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:**

**double** d1 = 12.3

## Char Data Type

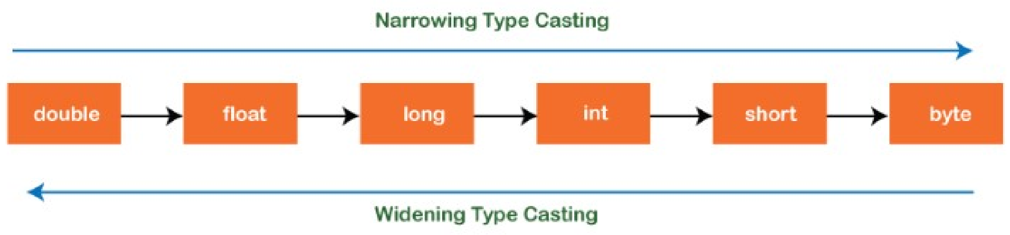
The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example:**

**char** letterA = 'A'

## Type Conversion & Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.



## Type casting

Convert a value from one data type to another data type is known as **type casting**.

### Types of Type Casting

There are two types of type casting:

- Widening Type Casting
- Narrowing Type Casting

## Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

**byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

### WideningTypeCastingExample.java

```
public class WideningTypeCastingExample

{

public static void main(String[] args)

{

int x = 7;

//automatically converts the integer type into long type

long y = x;

//automatically converts the long type into float type

float z = y;

System.out.println("Before conversion, int value "+x);

System.out.println("After conversion, long value "+y);

System.out.println("After conversion, float value "+z);

}  }
```

In the above example, we have taken a variable x and converted it into a long type. After that, the long type is converted into the float type.

## Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

**double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

**NarrowingTypeCastingExample.java**

```
public class NarrowingTypeCastingExample  {

public static void main(String args[])  {

double d = 166.66;

//converting double data type into long data type

long l = (long)d;

//converting long data type into int data type

int i = (int)l;

System.out.println("Before conversion: "+d);

//fractional part lost

System.out.println("After conversion into long type: "+l);

//fractional part lost

System.out.println("After conversion into int type: "+i);  }}
```

# Operators in Java

**Operator** in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

## Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

### Java Unary Operator Example: ++ and --

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

## Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

## Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

### Java Left Shift Operator Example

```
public class OperatorExample{

public static void main(String args[]){

System.out.println(10<<2);//10*2^2=10*4=40

System.out.println(10<<3);//10*2^3=10*8=80

System.out.println(20<<2);//20*2^2=20*4=80

System.out.println(15<<4);//15*2^4=15*16=240

}}
```

## Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

### Java Right Shift Operator Example

```
public OperatorExample{

public static void main(String args[]){

System.out.println(10>>2);//10/2^2=10/4=2

System.out.println(20>>2);//20/2^2=20/4=5

System.out.println(20>>3);//20/2^3=20/8=2

}}
```

## Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

## Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

## Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

## Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left. (=, +=, -=, *=, /=, %= )