Object Oriented Programming with Java 8
PG-DMC

Akshita Chanchlani

Wrapper class

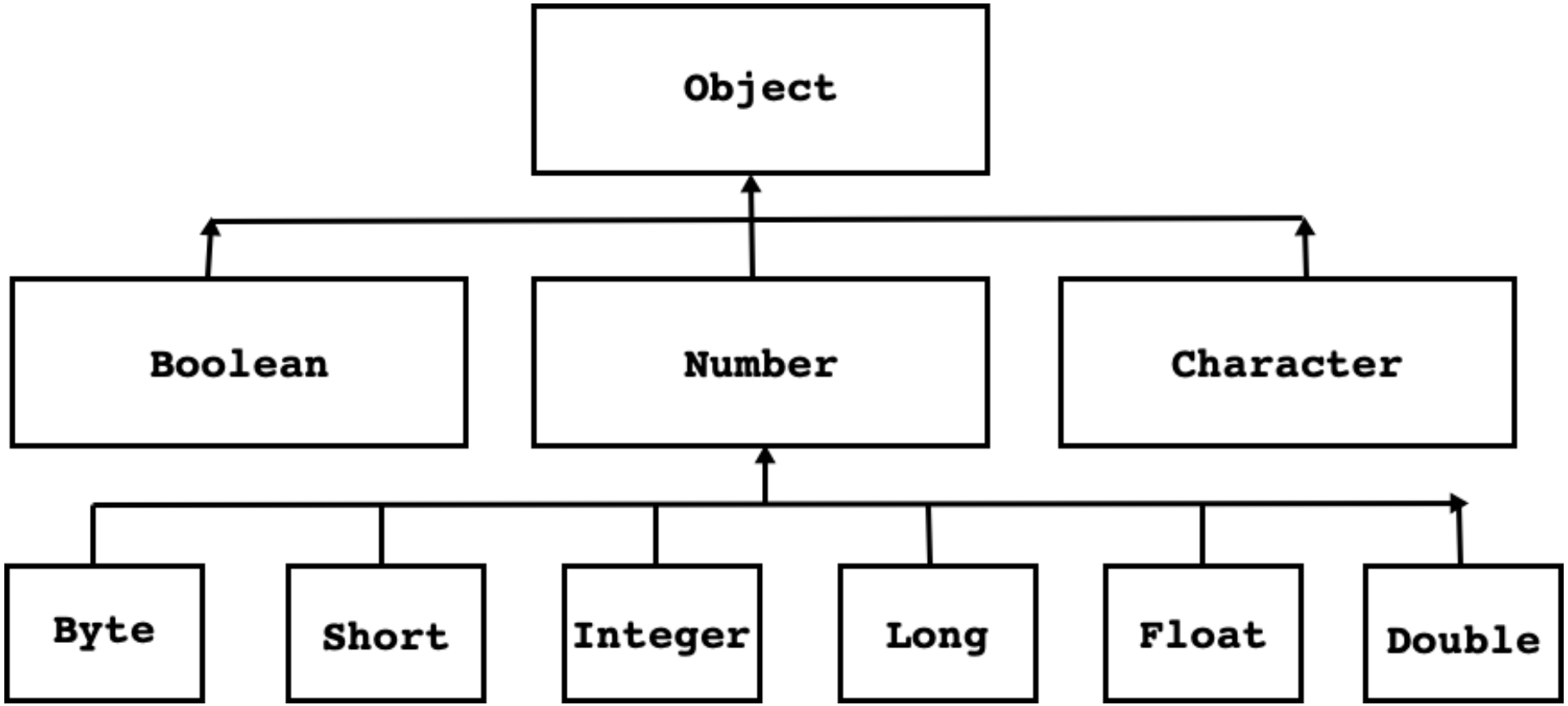
- In Java, primitive types are not classes. But for every primitive type, Java has defined a class. It is called wrapper class.

- All wrapper classes are final.

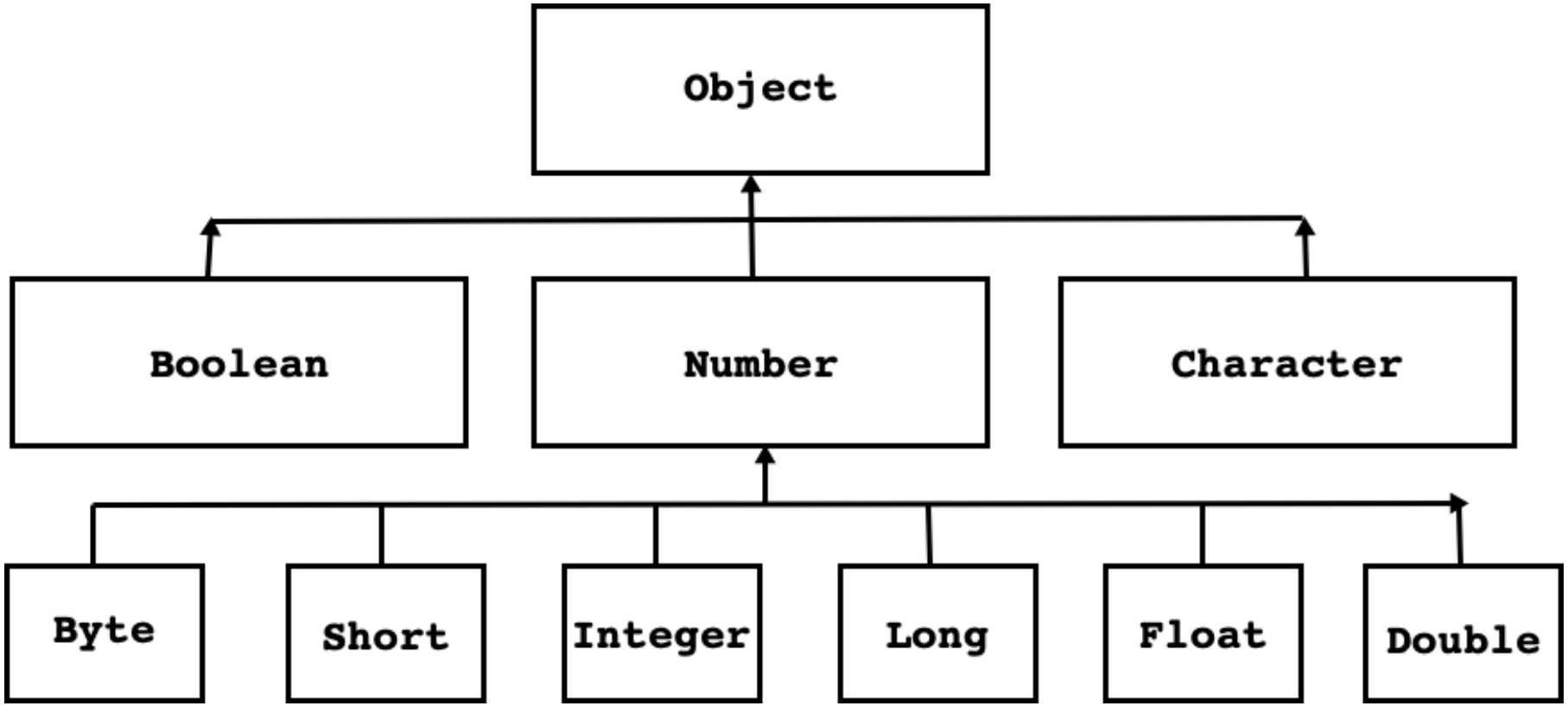
- All wrapper classes are declared in **java.lang** package.

- Uses of Wrapper class
  1. To parse string(i.e. to convert state of string into numeric type).
     example :
     ```
     int num = Integer.parseInt("123")
     float val = Float.parseFloat("125.34f");
     double d = Double.parseDouble("42.3d");
     ```
  1. To store value of primitive type into instance of generic class, type argument must be wrapper class.
     - **Stack<int> stk = new Stack<int>( ); //Not OK**
     - **Stack<Integer> stk = new Stack<Integer>( ); //OK**

Wrapper class



Wrapper class



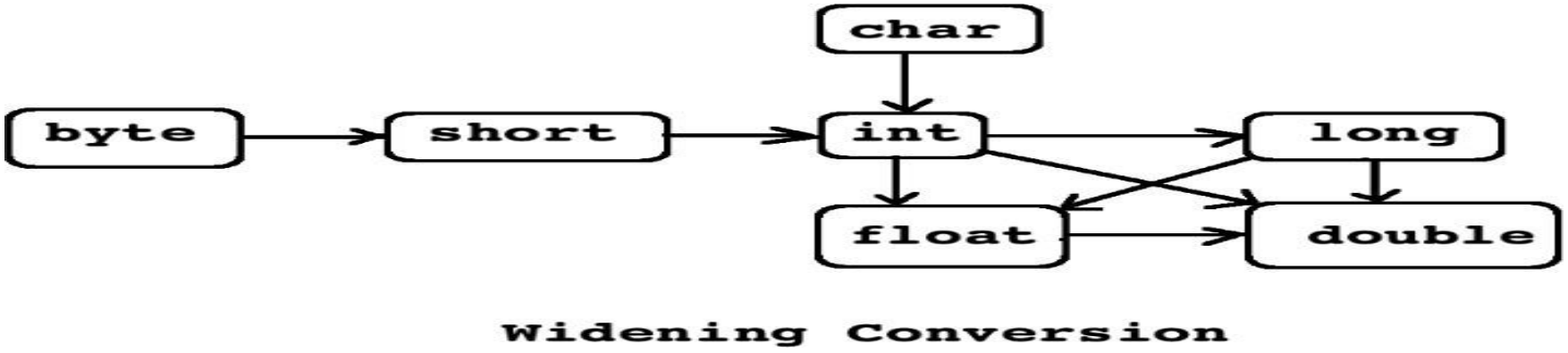
Widening

- Process of converting value of variable of narrower type into wider type is called widening.
- E.g. Converting int to double
-

```
public static void main(String[] args) {
    int num1 = 10;
    //double num2 = ( double )num1;    //Widening : OK
    double num2 = num1;    //Widening : OK
    System.out.println("Num2    :    "+num2);
}
```

- In case of widening, there is no loss of data
- So , explicit type casting is optional.

Widening



Widening Conversion

The range of values that can be represented by a float or double is much larger than the range that can be represented by a long. Although one might lose significant digits when converting from a long to a float, it is still a "widening" operation because the range is wider.

A widening conversion of an int or a long value to float, or of a long value to double, may result in loss of precision - that is, the result may lose some of the least significant bits of the value. In this case, the resulting floating-point value will be a correctly rounded version of the integer value, using IEEE 754 round-to-nearest mode.
Note that a double can exactly represent every possible int value.

long --->float ---is considered automatic type of conversion(since float data type can hold larger range of values than long data type)

Rules

- src & dest - must be compatible, typically dest data type must be able to store larger magnitude of values than that of src data type.

- 1. Any arithmetic operation involving byte,short --- automatically promoted to --int
- 2. int & long ---> long
- 3. long & float ---> float
- 4. byte,short.....& float & double----> double

Narrowing (Forced Conversion)

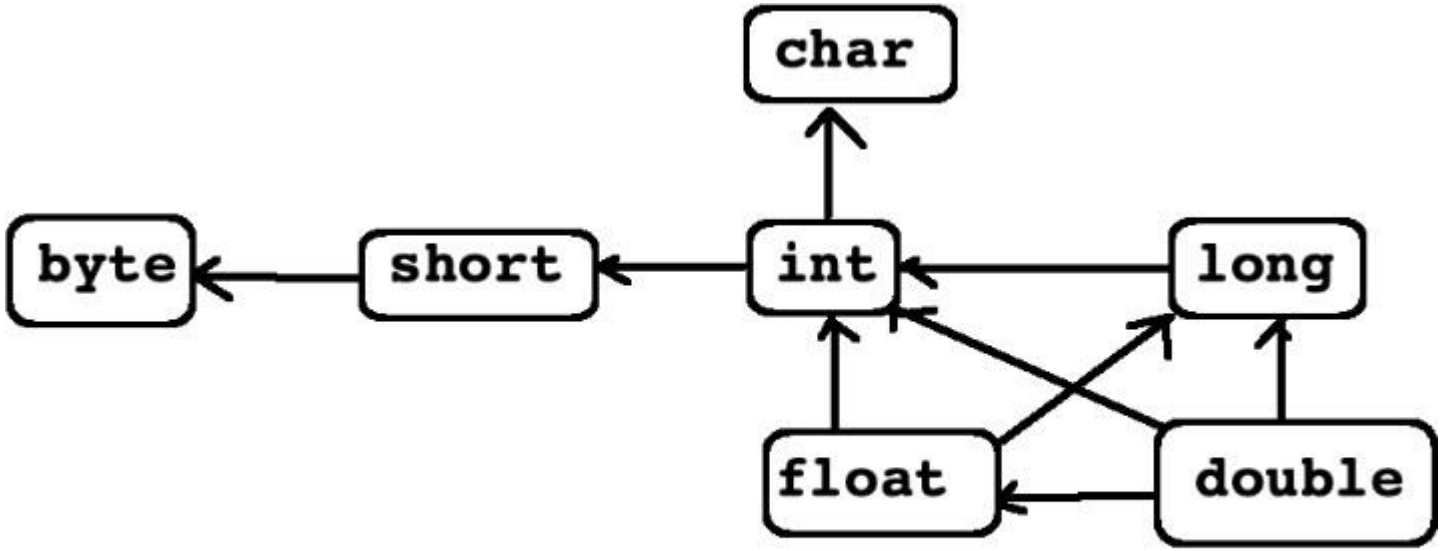
- Process of converting value of variable of wider type into narrower type is called narrowing.

```
public static void main(String[] args) {
    double num1 = 10.5;
    int num2 = ( int )num1;   //Narrowing : OK
    //int num2 = num1;   //Narrowing : NOT OK
    System.out.println("Num2     :    "+num2);
}
```

- In case of narrowing, explicit type casting is mandatory.

Note : In case of narrowing and widening both variables are of primitive

Narrowing



Narrowing Conversion.

eg ---

double ---> int

float --> long

double ---> float

Boxing

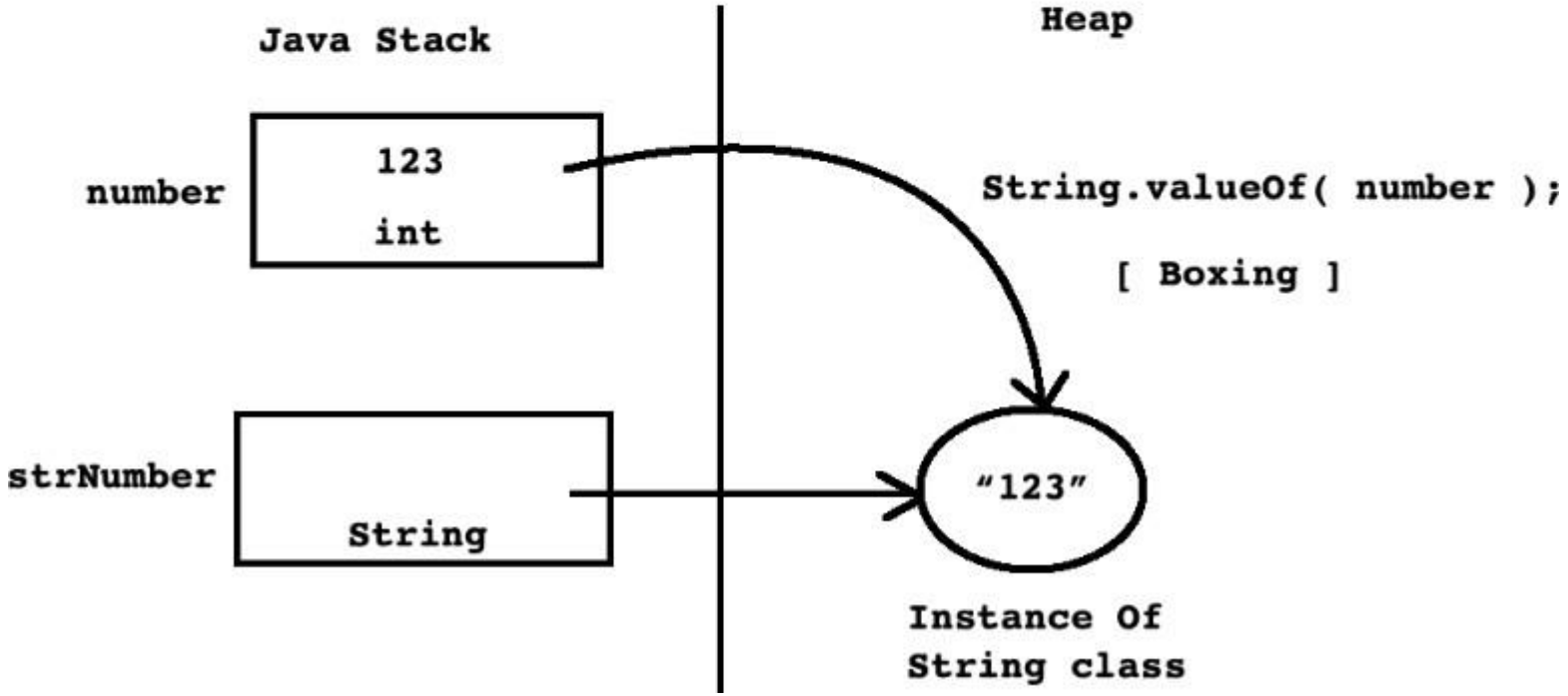
- Process of converting value of variable of primitive type into non primitive type is called **boxing**.

```
public static void main(String[] args) {
    int number = 123;
    //String str = Integer.toString( number );    //Boxing : OK
    String str = String.valueOf(number);          //Boxing : OK
    System.out.println("Str :   "+str);
}
```

  - int n1=10; float f=3.5f; double d1=3.45
  - String str1=String.valueOf(n1);
  - String str2=String.valueOf(f);
  - String str3=String.valueOf(d1);

Boxing

```
int number = 123;
String strNumber = String.valueOf( number ); //Boxing
```



Unboxing

- Process of converting value of variable of non primitive type into primitive type is called unboxing.

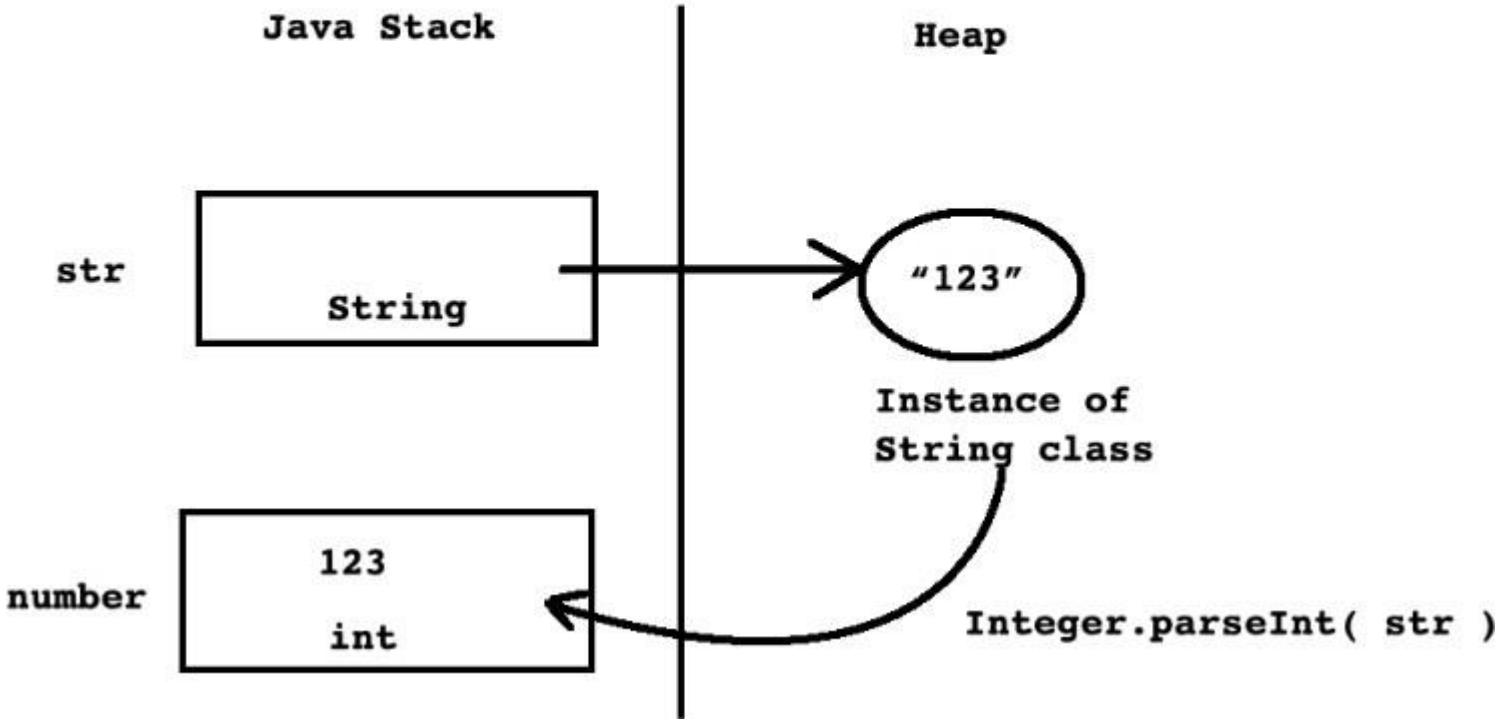
```
public static void main(String[] args) {
    String str = "123";
    int number = Integer.parseInt(str); //UnBoxing
    System.out.println("Number  :   "+number);
}
```

- If string does not contain parseable numeric value then **parseXXX( )** method throws **NumberFormatException.**

```
String str = "12c";
int number = Integer.parseInt(str); //UnBoxing : NumberFormatException
```

Unboxing

```
String str = "123";
int number = Integer.parseInt( str ); //UnBoxing
```



Note : In case of boxing and unboxing one variable is primitive and other Is not primitive

Command line argument

```
class Program{
    public static void main( String[] args ){
        int num1      = Integer.parseInt(args[0]);
        float num2    = Float.parseFloat(args[1]);
        double num3   = Double.parseDouble(args[2]);
        double result = num1 + num2 + num3;
        System.out.println("Result : "+result);
    }
}
```

+ User input from terminal:
    - java Program 10 20.3f 35.2d (Press enter key)

Operators

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators
- Shift Operators

Operators Cont..

- **Arithmetic Operators**
  - They are used to perform simple arithmetic operations on primitive data types.

  * : Multiplication

  / : Division

  % : Modulo

  + : Addition

  – : Subtraction
- **Unary Operators**
  - Unary operators need only one operand. They are used to increment, decrement or negate a value.
  - Unary minus, used for negating the values.
  - eg : int a=20; int b=-a;
  - ++ :Increment operator, used for incrementing the value by 1. There are two varieties of increment operator.
  - Post-Increment : Value is first used for computing the result and then incremented.
  - Pre-Increment : Value is incremented first and then result is computed.
  - -- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operator.
  - Post-decrement : Value is first used for computing the result and then decremented.
  - Pre-Decrement : Value is decremented first and then result is computed.
  - ! : Logical not operator, used for inverting a boolean value.
  - eg : boolean jobDone=true; boolean flag=!jobDone; System.out.println(flag);

Operators Cont..

- **Assignment Operators**
  - ‘=’ Assignment operator is used to assign a value to any variable. It has a right to left associativity.
  - Eg. int val = 500;
  - assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement.
  - +=, for adding left operand with right operand and then assigning it to variable on the left.
  - -=, for subtracting left operand with right operand and then assigning it to variable on the left.
  - *=, for multiplying left operand with right operand and then assigning it to variable on the left.
  - /=, for dividing left operand with right operand and then assigning it to variable on the left.
  - %=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.
- **Relational Operators**
  - These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are used in looping statements and conditional if else statements.
  - ==, Equal to : returns true if left hand side is equal to right hand side.
  - !=, Not Equal to : returns true if left hand side is not equal to right hand side.
  - <, less than : returns true if left hand side is less than right hand side.
  - <=, less than or equal to : returns true if left hand side is less than or equal to right hand side.
  - >, Greater than : returns true if left hand side is greater than right hand side.
  - >=, Greater than or equal to: returns true if left hand side is greater than or equal to right hand side.

Operator Cont…

- **Logical Operators :**
  - These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics.
    &&, Logical AND : returns true when both conditions are true.
    ||, Logical OR : returns true if at least one condition is true.
  - eg : int data=100;
    int data2=50;
    if(data > 60 && data2 < 100)
    System.out.println("test performed...");
    else
    System.out.println("test not performed...");

- **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.
  - General format is :
    condition ? if true : if false
    The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.
    eg : int data=100;
    System.out.println(data>100?"Yes":"No");

Operators Cont..

- **Bitwise Operators :**
  - These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.
  - &, Bitwise AND operator: returns bit by bit AND of input values.
  - |, Bitwise OR operator: returns bit by bit OR of input values.
  - ^, Bitwise XOR operator: returns bit by bit XOR of input values.
  - ~, Bitwise Complement Operator: This is a unary operator which returns the one's compliment representation of the input value, i.e. with all bits inversed.
  - Eg : String binary[] = { "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
    "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111" };

```
int a = 3; // 0 + 2 + 1 or 0011 in binary
int b = 6; // 4 + 2 + 0 or 0110 in binary
int c = a | b;
int d = a & b;
int e = a ^ b;
System.out.println("      a = " + binary[a]);
System.out.println("      b = " + binary[b]);
System.out.println("    a|b = " + binary[c]);
System.out.println("    a&b = " + binary[d]);
System.out.println("    a^b = " + binary[e]);      }
```

Operators Cont…

- **Shift Operators :**
  - These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

  - <<, Left shift operator: shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

eg :

```
int a = 25;
System.out.println(a<<4); //25 * 16 = 400
a=-25;
System.out.println(a<<4);//-25 * 16 = -400
```

  - Signed right shift operator : The signed right shift operator '>>' uses the sign bit to fill the trailing positions. For example, if the number is positive then 0 will be used to fill the trailing positions and if the number is negative then 1 will be used to fill the trailing positions.

Example Shift Operations

- Assume if a = 60 and b = -60; now in binary format, they will be as follows −

- a = 0000 0000 0000 0000 0000 0000 0011 1100
- b = 1111 1111 1111 1111 1111 1111 1100 0100
- In Java, negative numbers are stored as 2's complement.

- Thus a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010

- Unsigned right shift operator
- The unsigned right shift operator '>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

- Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements: Conditional Stmt: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return

if Statement – different syntax options

```
if
(expression)
   statement;
```

⇨ A single statement.

```
if
(expression)
   {
    statements;
   }
```

⇨ A block of statements.

```
if
(expression)
   statement;
else
statement;
```

⇨ Single statement in the if and a single statement in the else.

```
if
(expression)
      statement;
else
{
   statements;
}
```

⇨ A single statement in the if and a block of statements in the else.

Conditional Operator

- The operator “ ? : ” is the only operator that takes three operands, each of which is an expression.
- The value of the whole expression equals the value of expr2 if expr1 is true, or equals the value of expr3 if expr1 is false.
- Syntax:

```
expr1 ? expr2 : expr3
```

switch Statement

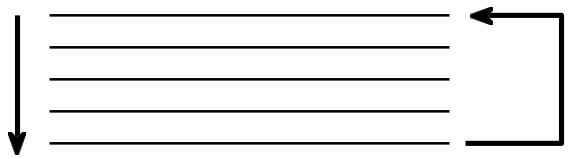
- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Syntax:

```
switch(expression)
{
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    default :
            statement(s);
            [break;]
}
```

- Usually, but not always, the last statement of a case is break.
- default case is optional.

Loops

- Loops break the serial execution of the program.
- A group of statements is executed a number of times.



There are three kinds of loops :

- **`while`**
- **`for`**
- **`do … while`**

While – Loop

- Syntax:

```
while (expression)
   Statement;
```

or

```
while (expression)
{Statements;}
```

- The loop continues to iterate as long as the value of expression is true (expression differs from zero).
- Expression is evaluated each time before the loop body is executed.
- The braces { } are used to group declarations and statements together into a compound statement or block, so they are syntactically equivalent to a single statement.

for - Loop

- Syntax:

```
for (expr1 ; expr2 ; expr3)
     statement;
```

or

```
for (expr1 ; expr2 ; expr3)
{
    statements;
}
```

- Is equivalent to:

```
expr1;
while (expr2)
{
    {statements;}
    expr3;
}
```

do while Loop

- Syntax:

```
do
{
    Statements;
}while (expression);
```

- The condition expression for looping is evaluated only after the loop body had executed.

break Statement

- We have seen how to use the break statement within the switch statement.

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

continue Statement

- In some situations, you might want to skip to the next iteration of a loop without finishing the current iteration.
- The **`continue`** statement allows you to do that.
- When encountered, **`continue`** skips over the remaining statements of the loop, but **`continues`** to the next iteration of the loop.

What is Scanner ?

- A class (java.util.Scanner) that represents text based parser(has inherent small ~ 1K buffer)
- It can parse text data from any source --Console input,Text file , socket, string

e.g. Scanner input = new Scanner(System.in);

```
System.out.print("Enter your name: ");
String name = input.next ();
System.out.println(“Your name is " + name);
input.close();
```

User Input Using Scanner class.

- Scanner is a final class declared in java.util package.
- Methods of Scanner class:

```
1. public String nextLine()
2. public int nextInt()
3. public float nextFloat()
4. public double nextDouble()
```

- How to user Scanner?

```
Scanner sc = new Scanner(System.in);
String name = sc.nextLine( );
int empid = sc.nextInt( );
float salary = sc.nextFloat( );
```

Thank you.

akshita.Chanchlani@sunbeaminfo.com