



# OOP using Java

Trainer: Mr. Rohan Paramane



# Language Basics

---

- Keywords
- Data Types
- Variables
- Operators
- Conditional Statements
- Loops
- Coding Conventions



# Keywords

<b>abstract</b>	<b>boolean</b>	<b>break</b>	<b>byte</b>	<b>case</b>	<b>catch</b>
<b>char</b>	<b>class</b>	<b>const</b>	<b>continue</b>	<b>default</b>	<b>do</b>
<b>double</b>	<b>else</b>	<b>extends</b>	<b>final</b>	<b>finally</b>	<b>float</b>
<b>for</b>	<b>goto</b>	<b>if</b>	<b>implements</b>	<b>import</b>	<b>instanceof</b>
<b>int</b>	<b>interface</b>	<b>long</b>	<b>native</b>	<b>new</b>	<b>package</b>
<b>private</b>	<b>protected</b>	<b>public</b>	<b>return</b>	<b>short</b>	<b>static</b>
<b>strictfp</b>	<b>super</b>	<b>switch</b>	<b>synchronized</b>	<b>this</b>	<b>throw</b>
<b>throws</b>	<b>transient</b>	<b>try</b>	<b>void</b>	<b>volatile</b>	<b>while</b>
<b>true</b>	<b>false</b>	<b>null</b>			



# Data Types

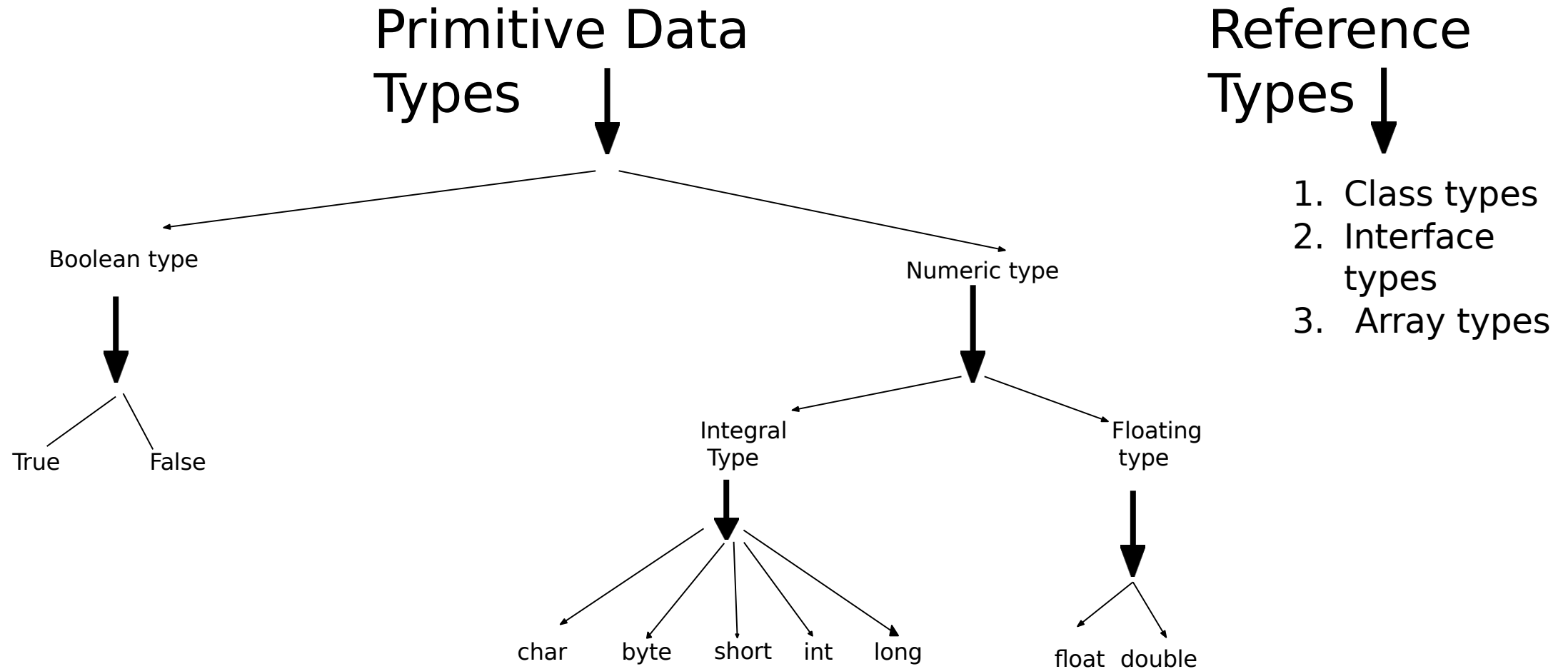
- Data type of any variable decide following things:
  1. Memory: How much memory is required to store the data.
  2. Nature: Which kind of data is allowed to store inside memory.
  3. Operation: Which operations are allowed to perform on the data stored in memory.
  4. Range: Set of values that we can store inside memory.
- The Java programming language is a statically typed language, which means that every variable and every expression has a type that is known at compile time.

## Types of data type:

1. Primitive type(also called as value type )
  - a. boolean type
  - b. Numeric type
    - i. Integral types (byte, char, short, int, long)
    - ii. Floating point types (float, double)
2. Non primitive type(also called as reference type)
  - a. Interface, Class, Type variable, Array



# Data Types



# Data Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits. Range is 1.40239846e-45f to 3.40282347e+38f
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits. Range is 4.94065645841246544e-324 to 1.79769313486231570e+308
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values



# Variables

- A **variable** is a name given memory location.
- That memory is associated to a data type and can be assigned a value.
- `int n; float f1; char ch; double d;`
- Rules of Variables
  - All variable names must begin with a letter of the alphabet, an underscore ( `_` ), or a dollar sign ( `$` ). Can't begin with a digit. The rest of the characters may be any of those previously mentioned plus the digits 0-9.
  - The convention is to always use a (lower case) letter of the alphabet. The dollar sign and the underscore are discouraged.
- `int n1;`
- `n1=21;               // assignment`
- `int val=50;           //initialization`
- `double d = 21.8;      // initialization`
- `d = n1;               // assignment`
- `float f1 = 16.13f;`



# Conditional Statements

1. if
2. else

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

⇒ A single statement.

```
if(expression){
    statement1;
    statement2;
}
```

⇒ 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
{
    statement1;
    statement2;
}
```

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





# Conditional statements

## 1. else-if

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



A single statement in the if, else-if and in the else block.

- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Usually, but not always, the last statement of a case is break.

## 2. switch

```
Switch(expression)
{
    case constant expr:
        statement(s);
        break;
    case constant expr:
        statement(s);
        break;
    case constant expr:
        statement(s);
        break;
    default :
        statement(s);
        break;
}
```



# 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 :

1. for
2. while
3. do ... while

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

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

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

o

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

- Is

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

equivalent to:



# break and continue statement

## 1. 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.

## 1. 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.



# Pascal Case Coding Conventions

- Example
  - System
  - StringBuilder
  - NullPointerException
  - IndexOutOfBoundsException
- In this case, including first word, first character of each word must in upper case.
- We should use this convention for:
  - Type Name( Interface, class, Enum, Annotation )
  - File Name



# Camel Case Coding Conventions

---

- Example
  - main
  - parseInt
  - showInputDialog
  - addNumberOfDays
- In this case, excluding first word, first character of each word must in upper case.
- We should use this convention for:
  - Method Parameter and Local variable
  - Field
  - Method
  - Reference



# Widening

- Process of converting value of variable of narrower type into wider type is called widening.
- E.g. Converting int to double
- In case of widening, there is no loss of data
- So , explicit type casting is optional.
- 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.

```
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);  
}
```



# Narrowing (Forced Conversion)

- Process of converting value of variable of wider type into narrower type is called narrowing.
- In case of narrowing, explicit type casting is mandatory.
- **Note : In case of narrowing and widening both variables are of primitive**

```
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 );  
}
```



# Class

- Consider following examples:
  1. day, month, year - related to - Date
  2. hour, minute, second - related to - Time
  3. red, green, blue - related to Color
  4. real, imag - related to - Complex
  5. xPosition, yPosition - related to Point
  6. number, type, balance - related to Account
  7. name, id, salary - related to Employee
- If we want to group related data elements together then we should use/define class in Java.

```
class Date{  
    int day;        //Field  
    int month;      //Field  
    int year;       //Field  
}
```

```
class Employee{  
    String name;    //Field  
    int id;         //Field  
    float salary;   //Field  
}
```





# Class

---

- Class is a non primitive/reference type in Java.
- A **class** is a user defined blueprint or prototype or template, from which objects are created.
- It is a logical entity
- It is a collection of fields(variables) and methods(Functions)
- Field
  - A variable declared inside class / class scope is called a field.
  - Field is also called as attribute or property.
- Method
  - A function implemented inside class/class scope is called as method.
  - Method is also called as operation, behavior or message.



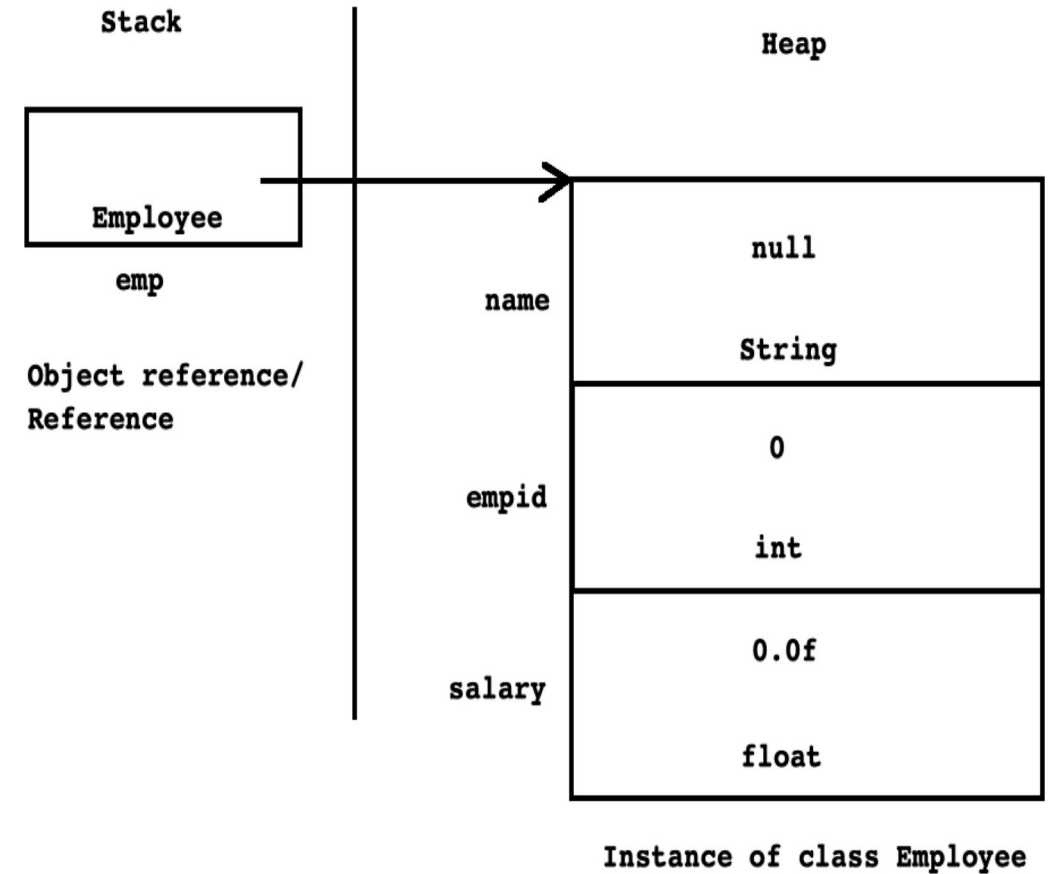
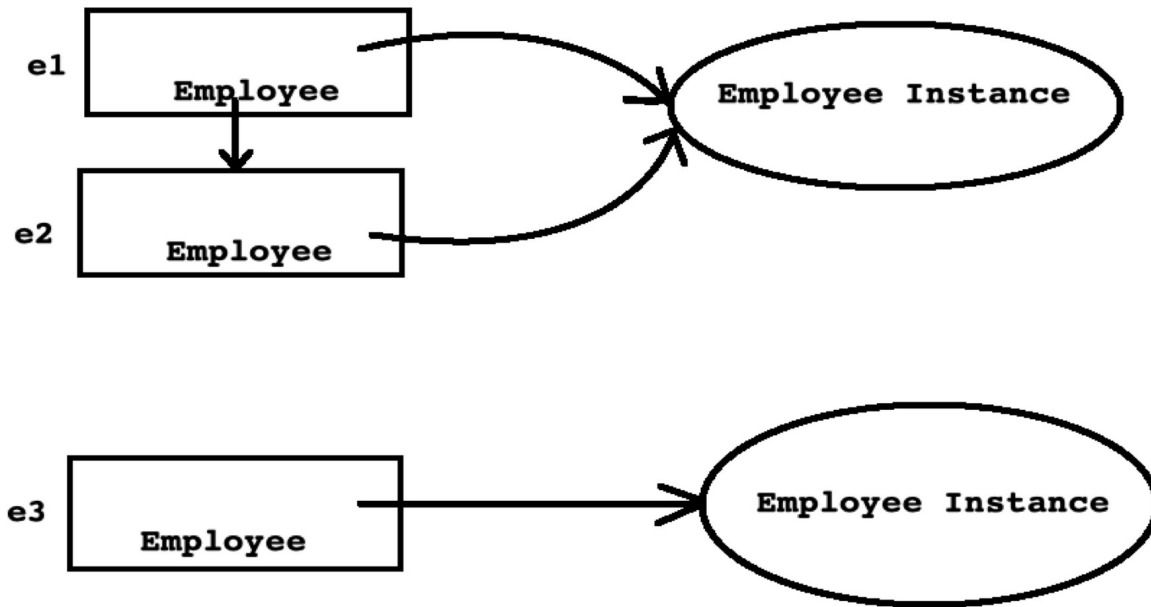
# Object

- It is a basic unit of Object Oriented Programming.
- It is a physical entity
- In Java, Object is also called as instance.
- An object consists of :
  - State : It is represented by attributes of an object. (properties / instance variables(non static) / fields)
  - Behavior : It is represented by methods.
  - Identity : It gives a unique identity to an object and enables one object to interact with other objects.
    - eg : Emp id / Student PRN / Invoice No
- Creating an object
  - The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory.



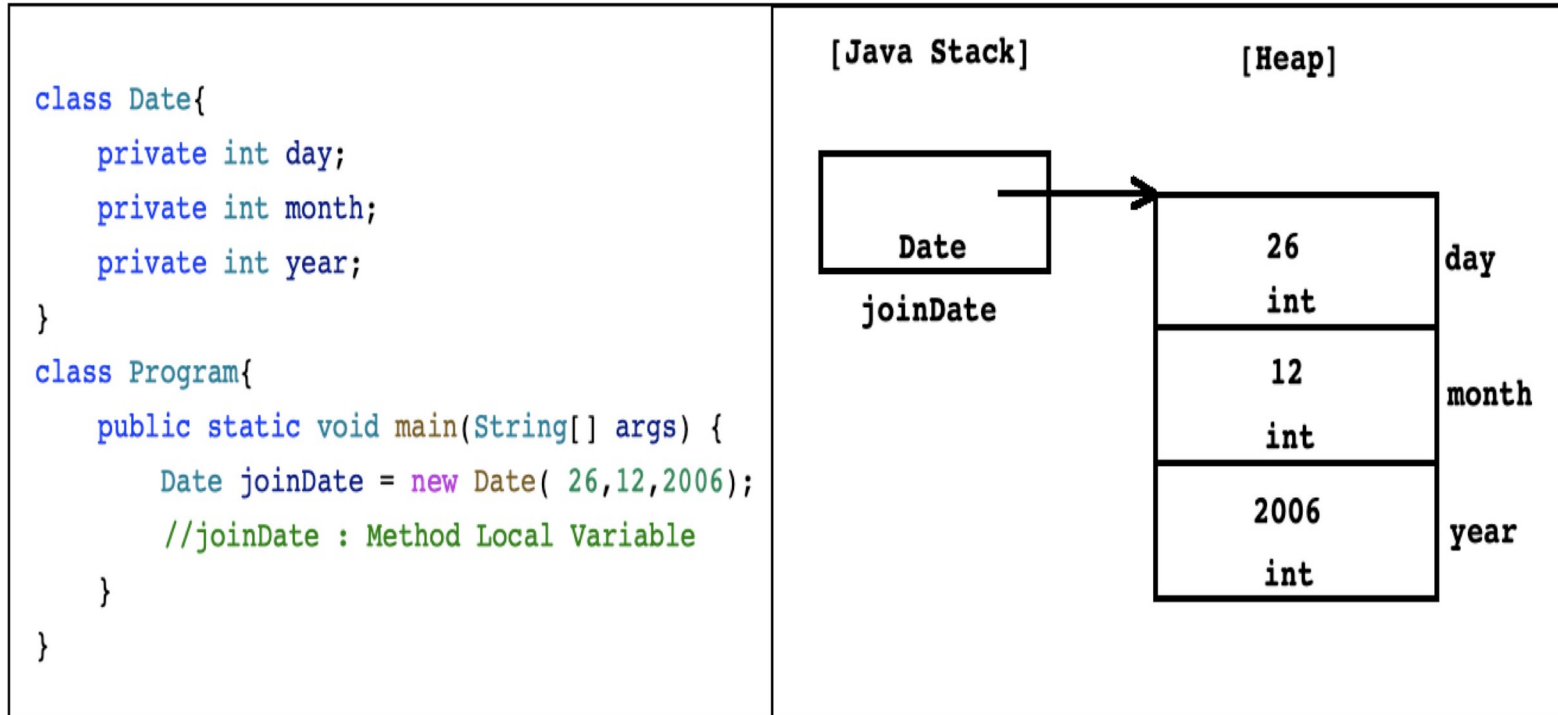
# Instantiation

- Process of creating instance/object from a class is called as instantiation.
- For eg –
  - `Employee e1 = new Employee();`
  - `Employee e2 = e1;`
  - `Employee e3 = new Employee();`



# Reference

- Local reference variable get space on Java Stack.



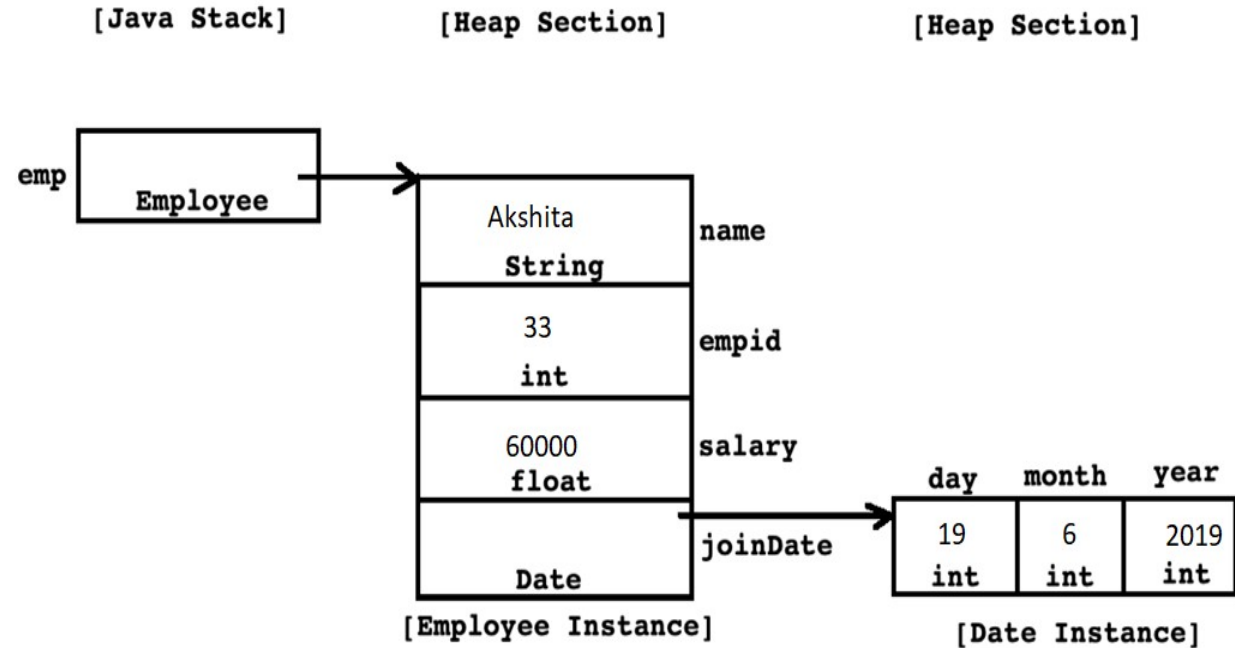
- In above code joinDate is method local reference variable hence it gets space on Java Stack.



# Reference

- Class Scope reference variable get space on Java heap.

```
class Employee{  
    private String name;  
    private int empid;  
    private float salary;  
    private Date joinDate; //joinDate : Field  
    public Employee( String name, int empid, float salary, Date joinDate ){  
        this.name = name;  
        this.empid = empid;  
        this.salary = salary;  
        this.joinDate = joinDate;  
    }  
}
```



- In above code, `emp` is method local reference variable hence it gets space on Java Stack. But `joinDate` is field of `Employee` class hence it will get space inside instance on Heap.

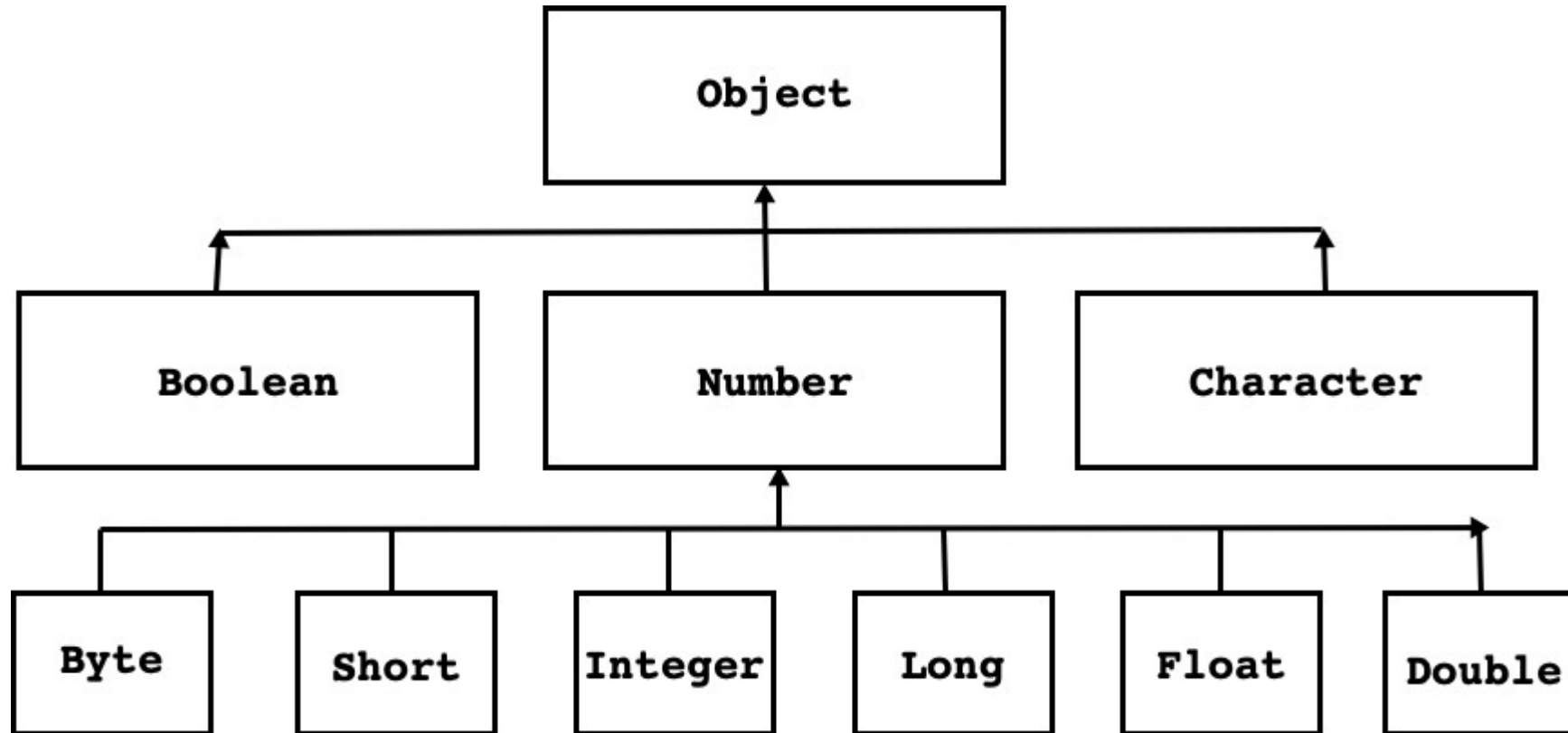


# Wrapper class

- 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 and are declared in java.lang package.
- Uses of Wrapper class
  - 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");`
- 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



# Boxing

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

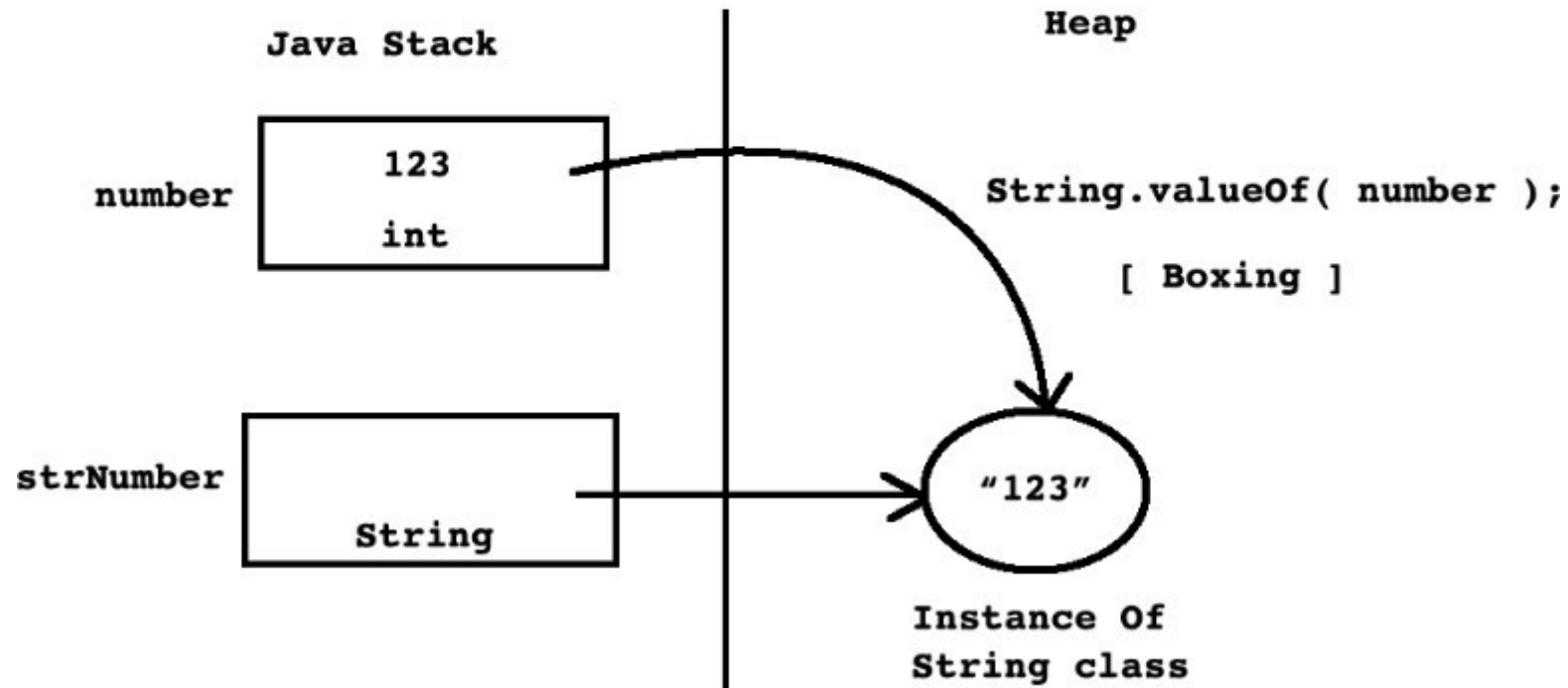
```
public static void main(String[] args) {  
    int number = 123;  
    //String str = Integer.toString( number );    //Boxing : OK  
    String str = String.valueOf(number);          //Boxing : OK  
    System.out.println("Str : "+str);  
}
```





# 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.

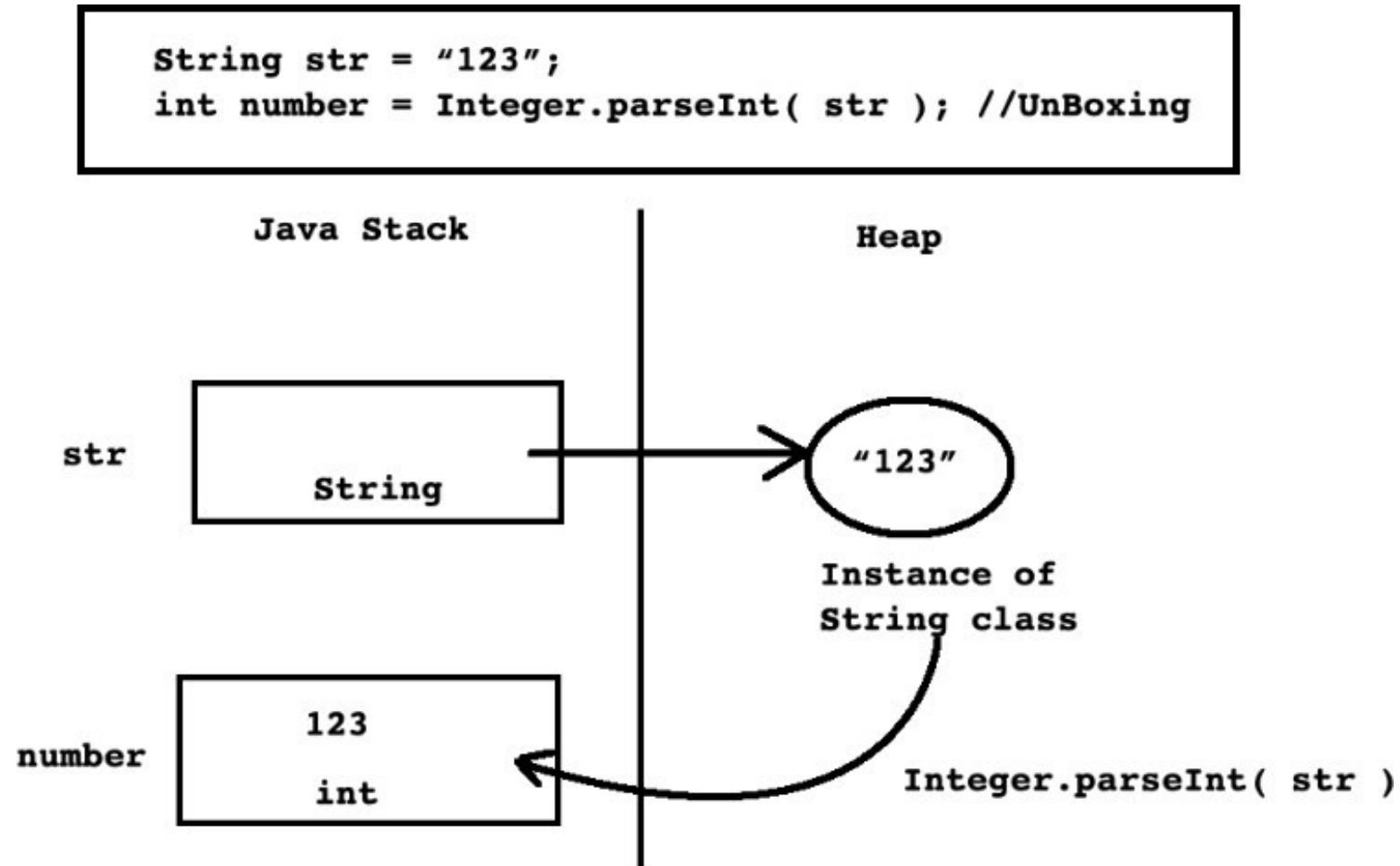
```
public static void main(String[] args) {  
    String str = "123";  
    int number = Integer.parseInt(str); //UnBoxing  
    System.out.println("Number : "+number);  
}
```

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

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



# UnBoxing



# 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)



# Scanner

- A class (java.util.Scanner) that represents text based parser.
- It can parse text data from any source
- 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 use Scanner?

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





Thank you!

Rohan Paramane

[rohan.paramane@sunbeaminfo.com](mailto:rohan.paramane@sunbeaminfo.com)

