Core Java Foundation – Day01

- Akshita Chanchlani

Trainer Introduction

- **Name**: Mrs.Akshita S.Chanchlani
- **Designation** : Technical Trainer
- **Education** :
  - PhD (Pursuing) : Computer Science and Engineering
  - Masters of Engineering (ME) in Information Technology
  - B.Tech in Computer Engineering, From VJTI Mumbai
- **Training Experience**
  - PreCAT Batches at Sunbeam
  - C, C++ , core java, python
- **Professional Experience**
  - **11+ years**
- **Email** : akshita.chanchlani@sunbeaminfo.com

Contents of Java Foundation Course

- JDK Installation Steps Explanation
- Java History and Standards
- Class & Object
- Constructor , Destructor, Getter, Setter
- Scanner class
- Static keyword, static field, static member function
- User defined functions
- Package concept
- Array
- Inheritance Introduction
- String
- Generics Introduction
- Essential Collection Framework
  - List (ArrayList)
  - Stack
  - Queue (LinkedList)
  - Map (HashMap)
  - Comparable (Sorting)
  - Comparator

Documentation

- **JDK download:**
  - https://adoptopenjdk.net/
- **Spring Tool Suite 3 download:**
  - https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3
- **Oracle Tutorial:**
  - https://docs.oracle.com/javase/tutorial/
- **Java 8 API Documentation:**
  - https://docs.oracle.com/javase/8/docs/api/
- **MySQL Connector:**
  - https://downloads.mysql.com/archives/c-j/
- **Core Java Tutorials:**
  1. http://tutorials.jenkov.com/java/index.html
  2. https://www.baeldung.com/java-tutorial
  3. https://www.journaldev.com/7153/core-java-tutorial

Java History

- **James Gosling, Mike Sheridan** and **Patrick Naughton** initiated the Java language project in June 1991.
- **Java** was originally **developed by James Gosling** at **Sun Microsystems** and **released in 1995.**
- The language was initially called **Oak** after an oak tree that stood outside Gosling's office.
- Sun Microsystems released the first public implementation as Java 1.0 in 1996.
- The Java programming language is a general-purpose, concurrent, class-based, object-oriented language.
- The Java programming language is a high-level language.
- The Java programming language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included.
- **It is intended to be a production language, not a research language.**
- The Java programming language is statically typed means type of variables is known at the compile time.
- It promised **Write Once, Run Anywhere** (**WORA**) functionality.

Version History

Version	Date
JDK Beta	1995
JDK1.0	January 23, 1996[39]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11	September 25, 2018[40]
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020[41]
Java SE 16	March 16, 2021

- The first version was released on January 23, 1996.

- The acquisition of Sun Microsystems by Oracle Corporation was completed on January 27, 2010

- As of September 2020, Java 8 and 11 are supported as Long Term Support (LTS) versions

- In September 2017, Mark Reinhold, chief Architect of the Java Platform, proposed to change the release train to "one feature release every six months".

- OpenJDK (Open Java Development Kit) is a free and open source implementation of the (Java SE). It is the result of an effort Sun Microsystems began in 2006.

- Java Language and Virtual Machine Specifications: https://docs.oracle.com/javase/specs/

Java Platforms

1. **Java SE**
   - Java Platform Standard Edition.
   - It is also called as Core Java.
   - For general purpose use on Desktop PC's, servers and similar devices.
2. **Java EE**
   - Java Platform Enterprise Edition.
   - It is also called as advanced Java / enterprise java / web java.
   - Java SE plus various API's which are useful client-server applications.
3. **Java ME**
   - Java Platform Micro Edition.
   - Specifies several different sets of libraries for devices with limited storage, display, and power capacities.
   - It is often used to develop applications for mobile devices, PDAs, TV set-top boxes and printers.
4. **Java Card**
   - A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.

SDK, JDK, JRE, JVM

- **SDK** = Development Tools + Documentation + Libraries + Runtime Environment.

- **JDK** = Java Development Tools + Java Docs + **rt.jar** + JVM.
  - JDK : Java Development Kit.
  - It is a software, that need to be install on developers machine.
  - We can download it from oracle official website.

- **JDK** = Java Development Tools + Java Docs + **JRE[rt.jar + JVM**].
  - JRE : Java Runtime Environment.
  - rt.jar and JVM are integrated part of JRE.
  - JRE is a software which comes with JDK. We can also download it separately.
  - To deploy application, we should install it on client's machine.

- rt.jar file contains core Java API in **compiled form**.

- **JVM** : An engine, which manages execution of Java application. (also called as Execution Engine)

C++ versus Java terminology

C++ Terminology	Java Terminology
Class	Class
Data Member	Field
Member Function	Method
Object	Instance
Pointer	Reference
Access Specifier	Access Modifier
Base Class	Super Class
Derived Class	Sub Class
Derived From	Extended From
Runtime Polymorphism	Dynamic Method Dispatch

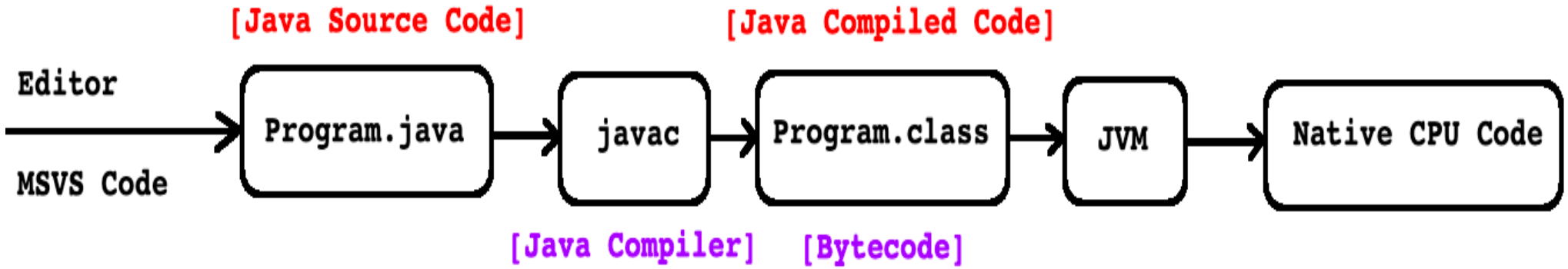
Simple Hello Application

```
//File Name : Program.java
class Program{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
Compilation=> javac Program.java  //Output : Program.class
Execution=>   java Program
```

```
System    :   Final class declared in java.lang package
out       :   public static final field of System class. Type of out is PrintStream
println   :   Non static method of java.io.PrintStream class
```

Java application flow of execution



Comments

- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program.
- Types of Comments:
  - **Implementation Comment**
    - Implementation comments are those found in C++, which are delimited by /*...*/, and //.
      1. Single-Line Comment
      2. Block Comment(also called as multiline comment)
  - **Documentation Comment**
    - Documentation comments (known as "doc comments") are Java-only, and are delimited by **/**...*/**.
    - Doc comments can be extracted to HTML files using the javadoc tool.

- Ref: https://www.oracle.com/java/technologies/javase/codeconventions-comments.html

Entry point method

- Syntax:
  1. **public static void main(String[] args)**
  2. **public static void main(String... args)**

- Java compiler do not check/invoke main method. JVM invoke main method.

- When we start execution of Java application then JVM starts execution of two threads:
  1. Main thread : responsible for invoking main method.
  2. Garbage Collector : responsible for deallocating memory of unused object.

- We can overload main method in Java.

- We can define main method per class. But only one main method can be considered as entry point method.

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

- Ref: https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html

Data Types

Sr.No.	Primitive Type	Size[In Bytes]	Default Value[For Field]	Wrapper Class
1	boolean	Isn't specified	FALSE	Boolean
2	byte	1	0	Byte
3	char	2	\u0000	Character
4	short	2	0	Short
5	int	4	0	Integer
6	float	4	0.0f	Float
7	double	8	0.0d	Double
8	long	8	0L	Long

Note : Char datatype supports UNICODE character set, so 2 bytes .

Wrapper class

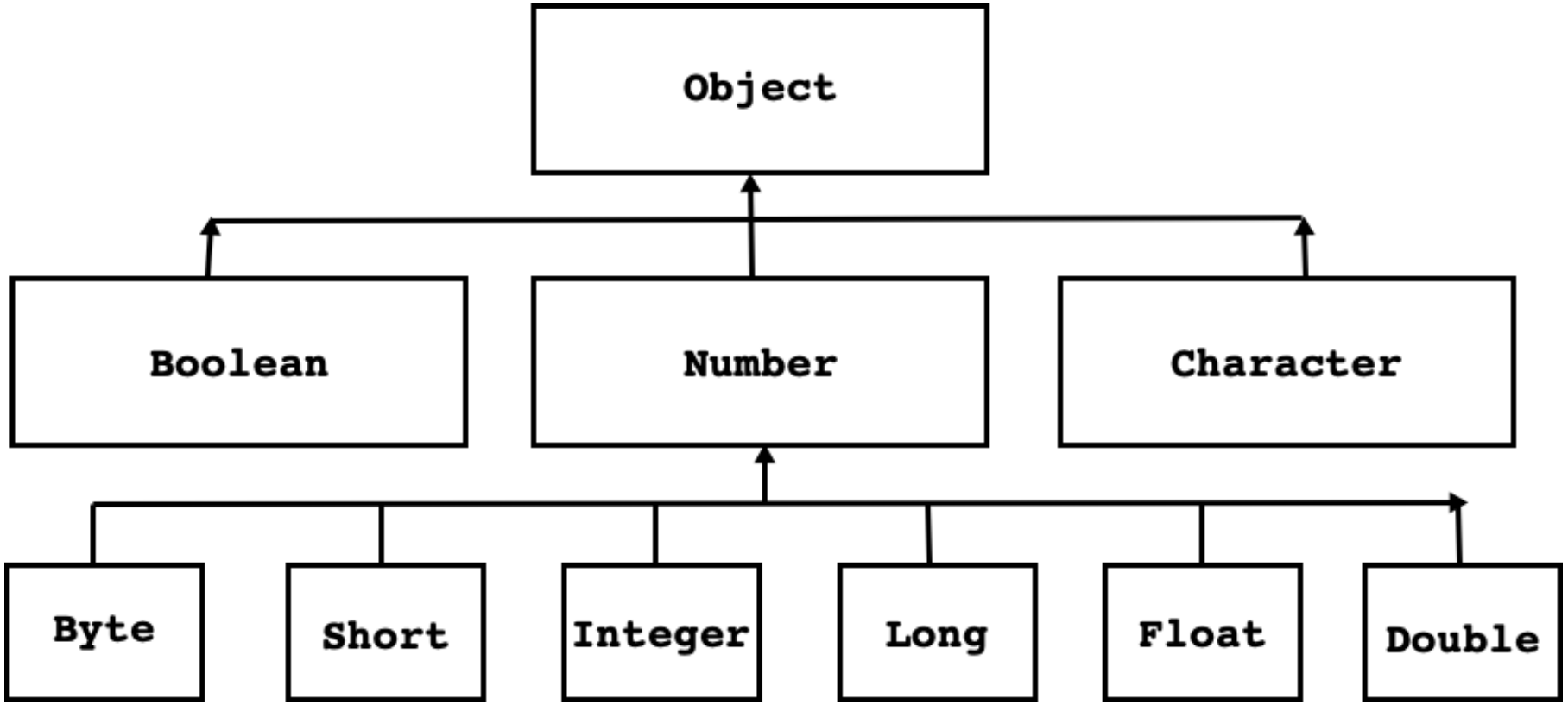
- In Java, primitive types are not classes. But for every primitive type, Java has defined a class. It is called wrapper class.

- 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



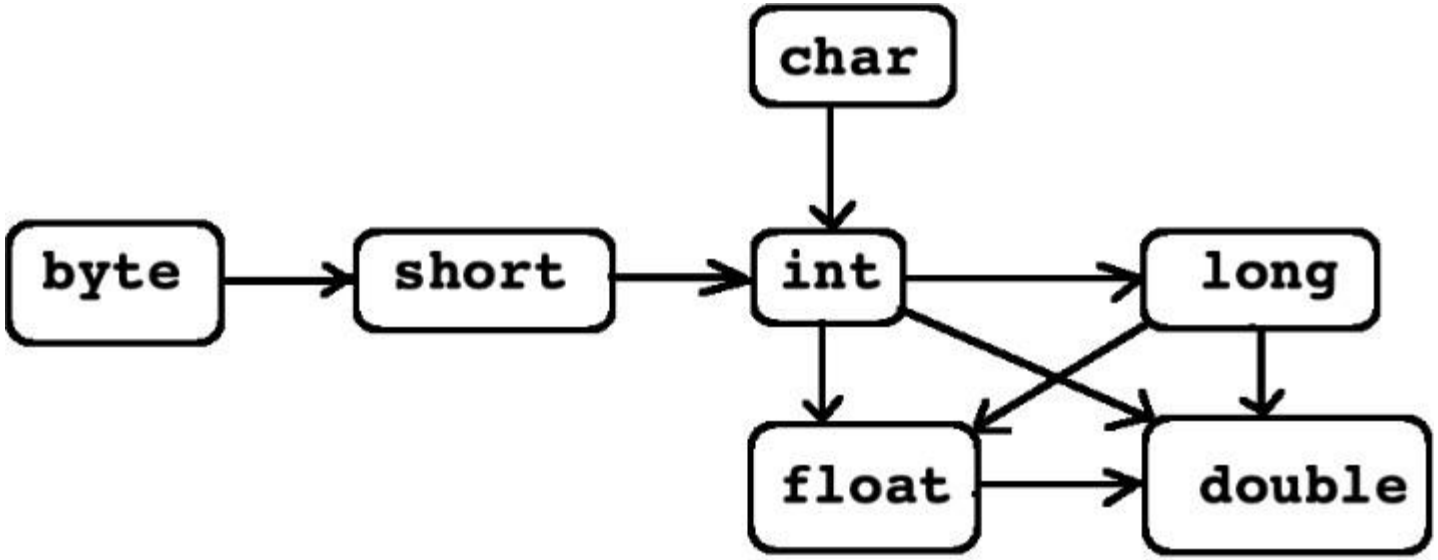
Widening

- 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

Narrowing

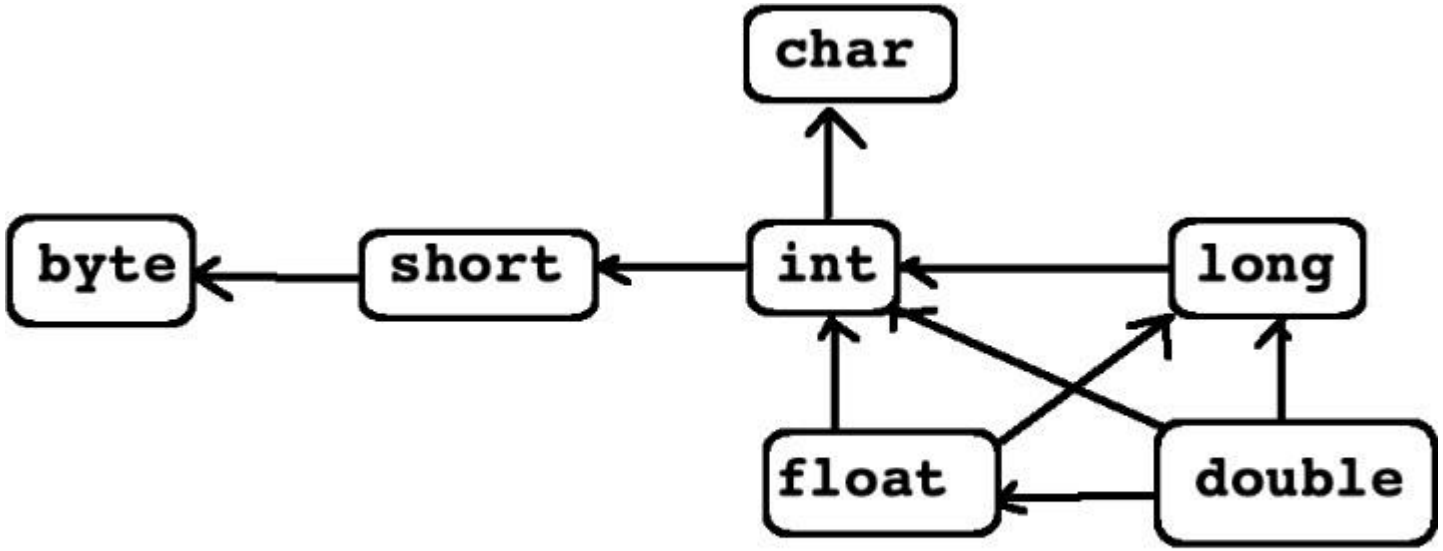
- Process of converting value of variable of wider type into narrower type is called narrowing.

```
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.

Boxing

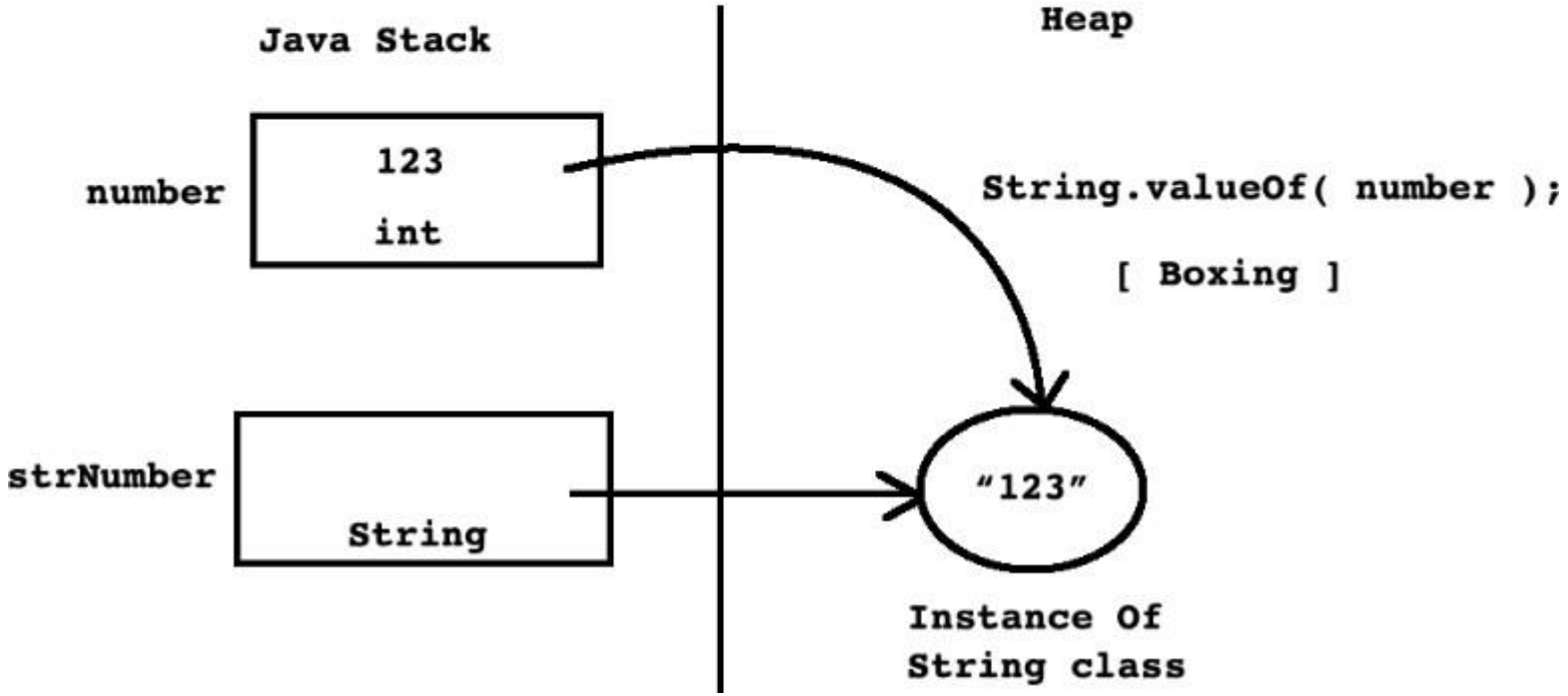
- Process of converting value of variable of primitive type into non primitive type is called **boxing**.

```
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.

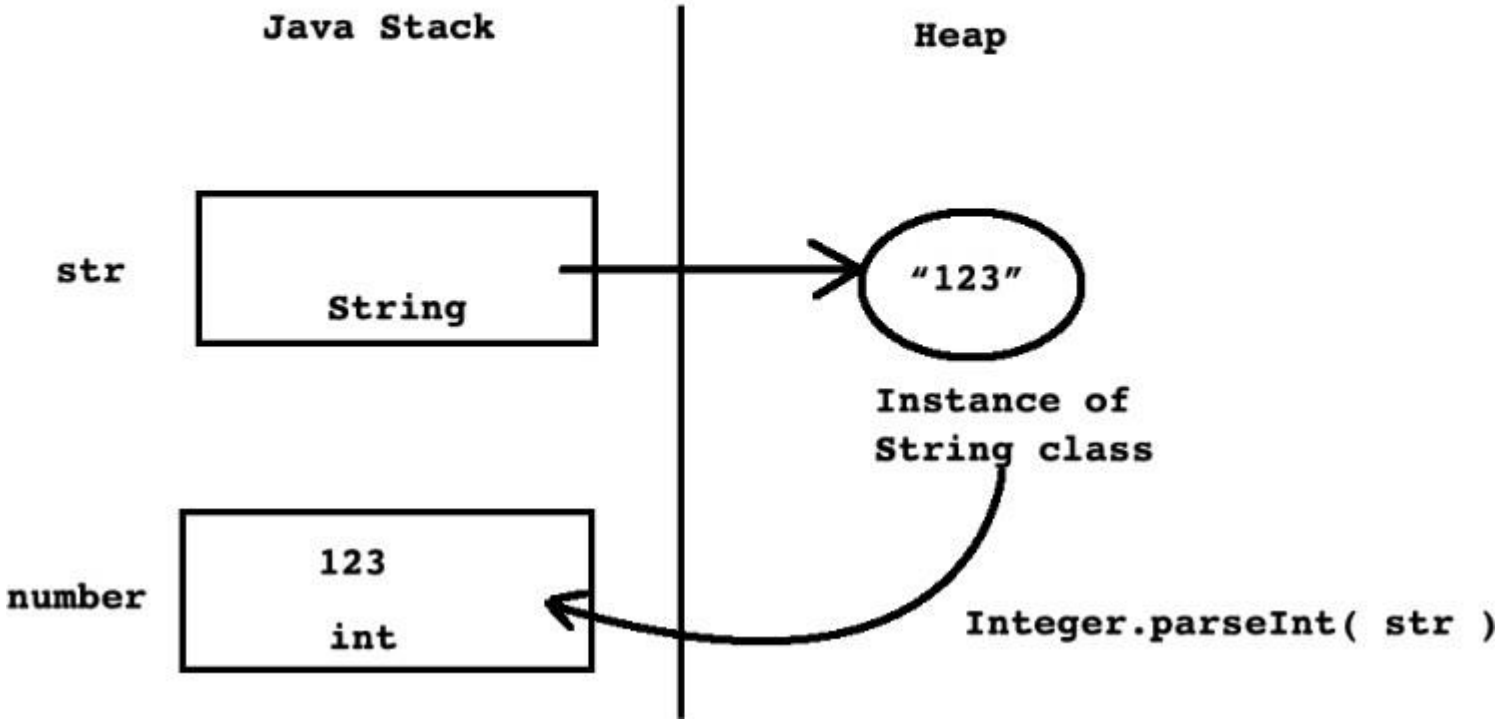
```
public static void main(String[] args) {
    String str = "123";
    int number = Integer.parseInt(str); //UnBoxing
    System.out.println("Number  :   "+number);
}
```

- 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

Java Buzzwords

1. Simple
2. Object Oriented
3. Architecture Neutral
4. Portable
5. Robust
6. Multithreaded
7. Dynamic
8. Secure
9. High Performance
10. Distributed

Simple

- Java is **simple** programming language.
  - **Syntax of Java is simpler than syntax of C/C++ hence it is considered as simple**

    - Ø No need of header files and macros.
    - Ø We can not define anything global
    - Ø Do not support structure and union. operator overloading.
    - Ø Do not support copy constructor and assignment operator function constructor member initializer list and default argument constant data member and constant member function.
      Delete operator, destructor , friend function, friend class.
      Multiple class (Multiple inheritance)

    - Ø `No diamond problem and virtual base class.`

    - Ø `Do not support pointer and pointer arithmetic.`

  - **Size of software(JDK), that is required to develop Java application is small hence Java is considered as simple programming language.**

Object Oriented

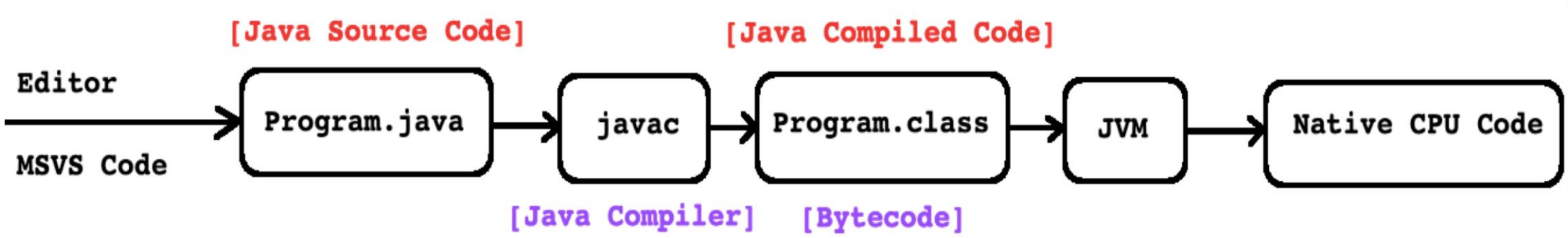
- Java is **object oriented** programming language.
  - Java Supports all the major and minor pillars of oops hence it is considered as object oriented programming language.

  - **Major pillars of oops.**
    1. Abstraction
    2. Encapsulation
    3. Modularity
    4. Hierarchy

  - **Minor pillars of oops.**
    1. Typing / Polymorphism
    2. Concurrency
    3. Persistence.

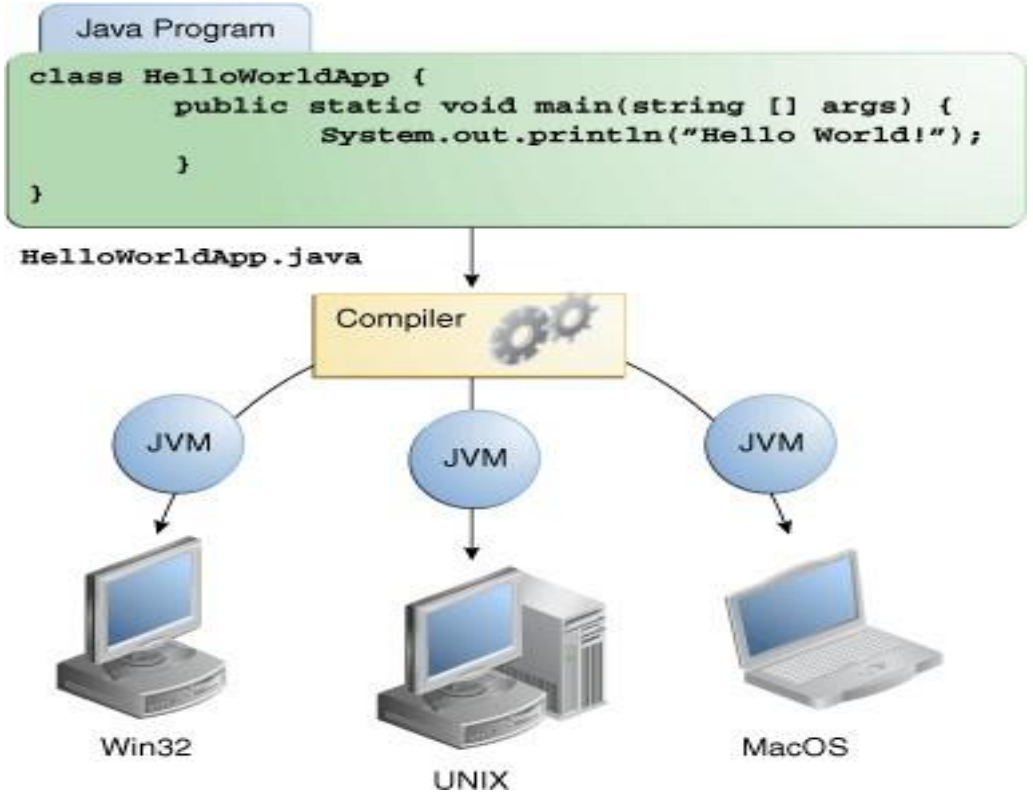
Architecture Neutral

- Java is object **architecture neutral** programming language.
  - Java technology is designed to support applications that will be deployed into heterogeneous network environments. In such environments, applications must be capable of executing on a variety of hardware architectures. Within this variety of hardware platforms, applications must execute on the top of a variety of operating systems. To accommodate the diversity of operating environments, the Java Compilerproduct generates *bytecodes*--an *architecture neutral* intermediate format designed to transport code efficiently to multiple hardware and software platforms.



Portable

- Java is **portable** programming language.
  - Architecture neutrality is just one part of a truly *portable* system .



Portable

- Java is **portable** programming language.
  - Java technology takes portability a stage further by being strict in its definition of the basic language.
  - Java technology puts a stake in the ground and specifies the sizes of its basic types and the behavior of its data arithmetic operators.
  - Your programs are the same on every platform--there are no data type incompatibilities across hardware and software architectures.

Sr.No.	Primitive Type	Size	Default Value For Field
1	boolean	Isn't Defined	FALSE
2	byte	1 Byte	0
3	char	2 Bytes	\u0000'
4	short	2 Bytes	0
5	int	4 Bytes	0
6	float	4 Bytes	0.0f
7	double	8 Bytes	0.0d
8	long	8 Bytes	0L

Robust

- Java is **robust** programming language.
  - The Java programming language is designed for creating highly *reliable* software. It provides extensive compile-time checking, followed by a second level of run- time checking.
  - Java is robust because of following features:

1. *Architecture Neutral*.
   - Ø Java developer is free from developing H/W or OS specific coding.
2. *Object orientation*.
   - Ø Reusability reduces developer's effort.
3. *Automatic memory management*.
   - Ø Developer need not to worry about memory leakage / program crashes.
4. *Exception handling*.
   - Ø Java compiler helps developer to provide try-catch block.

Multithreaded

- Java is **multithreaded** programming language.
  - When we start execution of Java application then JVM starts execution threads hence Java is considered as multithreaded.
    1. Main thread
       - Ø It is user thread / non daemon thread.
       - Ø It is responsible for invoking main method.
       - Ø Its default priority is 5(Thread.NORM_PRIORITY).
    2. Garbage Collector / Finalizer
       - Ø It is daemon thread / background thread.
       - Ø It is responsible for releasing / deallocating memory of unused objects.
       - Ø Its default priority is 8(Thread.NORM_PRIORITY + 3).
  - The Java platform supports multithreading at the language level with the addition of sophisticated synchronization primitives: the language library provides the Thread class, and the run-time system provides monitor and condition lock primitives. At the library level, moreover, Java technology's high-level system libraries have been written to be thread safe: the functionality provided by the libraries is available without conflict to multiple concurrent threads of execution.

Dynamic

- Java is **dynamic** programming language.
  - While the Java Compiler is strict in its compile-time static checking, the language and run-time system are *dynamic* in their linking stages. Classes are linked only as needed. New code modules can be linked in on demand from a variety of sources, even from sources across a network.
  - Java is designed to adapt to an evolving environment.
  - Libraries can freely add new methods and instance variables without any effect on their clients.
  - In Java finding out runtime type information is straightforward.
  - In Java, all the methods are by default virtual.

Secure

- Java is **secure** programming language.
  - Java is intended to be used in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems.
  - From the beginning, Java was designed to make certain kinds of attacks impossible, among them:
    1. Overrunning the runtime stack—a common attack of worms and viruses
    2. Corrupting memory outside its own process space
    3. Reading or writing files without permission

High Performance

- Java

  is **high performance** programming language.

  - The Java platform achieves superior performance by adopting a scheme by which the interpreter can run at full speed without needing to check the run-time environment.
  - The *automatic garbage collector* runs as a low-priority background thread, ensuring a high probability that memory is available when required, leading to better performance.
  - Applications requiring large amounts of compute power can be designed such that compute-intensive sections can be rewritten in native machine code as required and interfaced with the Java platform.
  - In general, users perceive that interactive applications respond quickly even though they're interpreted.

Distributed

- Java is **distributed** programming language.
- Java has an extensive library of routines for coping with protocols like HTTP , TCP/IP and FTP.
- Java applications can open and access objects across the Net via URL with the same ease as when accessing a local file system.

o Nowadays, one takes this for granted, but in 1995, connecting to a web server from a C++ or Visual Basic program was a major undertaking.

How to access members of package?

Package : p1

```
public class Complex{
  //TODO
}
```

```
public class Program{
  public static void main( String[] args ){
    p1.Complex c1 = new p1.Complex( );
  }
}
```

1

```
import p1.Complex;
public class Program{
  public static void main( String[] args ){
    Complex c1 = new Complex( );
  }
}
```

2

User Input Using Console class

- Console is class declared in java.io package.
- console( ) is a static method of System class which returns reference of java.io.Console class
  - public static Console console( );
- **public String readLine( )** is a method of java.io.Console class.

```
java.io.Console console = System.console( );
String name = console.readLine( );
int empid = Integer.parseInt( console.readLine( ) );
float salary = Float.parseFloat( console.readLine( ) );
```

```
import java.io.Console;
Console console = System.console( );
String name = console.readLine( );
int empid = Integer.parseInt( console.readLine( ) );
float salary = Float.parseFloat( console.readLine( ) );
```

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( );
```

Class

- class is a non primitive/reference type in Java.
- If we want create object/instance of a class then it is mandatory to use new operator.
- If we create instance using new operator then it gets space on heap section.
- Only fields of the get space once per instance according to order of their declaration inside class.

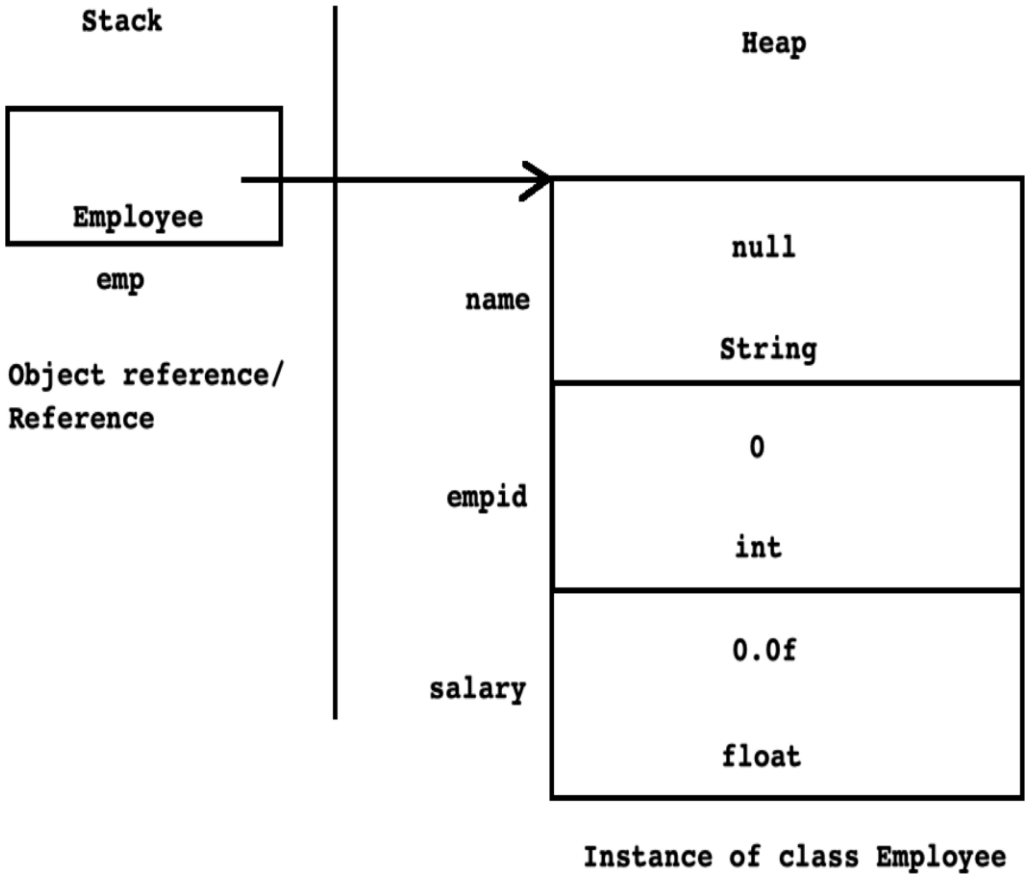
Class

- Field
  - Ø A variable declared inside class / class scope is called a field.
  - Ø Field is also called as attribute or property.
- Method
  - Ø A function implemented inside class/class scope is called as method.
  - Ø Method is also called as operation, behavior or message.
- Class
  - Ø Class is a collection of fields and methods.
  - Ø Class can contain
    1. Nested Type
    2. Field
    3. Constructor
    4. Method
  - **Instance**
    - Ø In Java, Object is also called as instance.

Instantiation

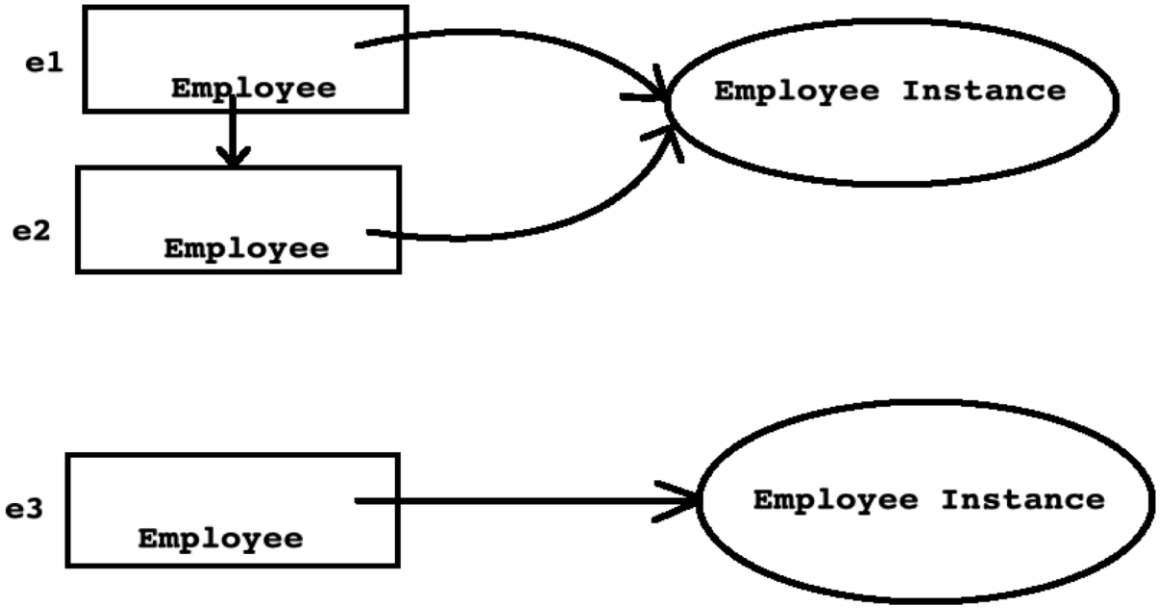
- Process of creating instance/object from a class is called as instantiation.
- In C programming language
  - Ø Syntax : struct StructureName identifier_name; struct
  - Ø Employee emp;
- In C++ programming language
  - Ø Syntax : [class] ClassName identifier_name;
  - Ø Employee emp;
- In Java programming language
  - Ø Syntax : ClassName identifier_name = new ClassName( );
  - Ø Employee emp = new Employee();
- **Every instance on heap section is anonymous.**

Instantiation



For eg :

```
1. Employee e1 = new Employee();
2. Employee e2 = e1;
3. Employee e3 = new Employee();
```



this reference

- If we call non static method on instance(actually object reference) then compiler implicitly pass, reference of current/calling instance as a argument to the method implicitly. To store reference of current/calling instance, compiler implicitly declare one reference as a parameter inside method. It is called this reference.

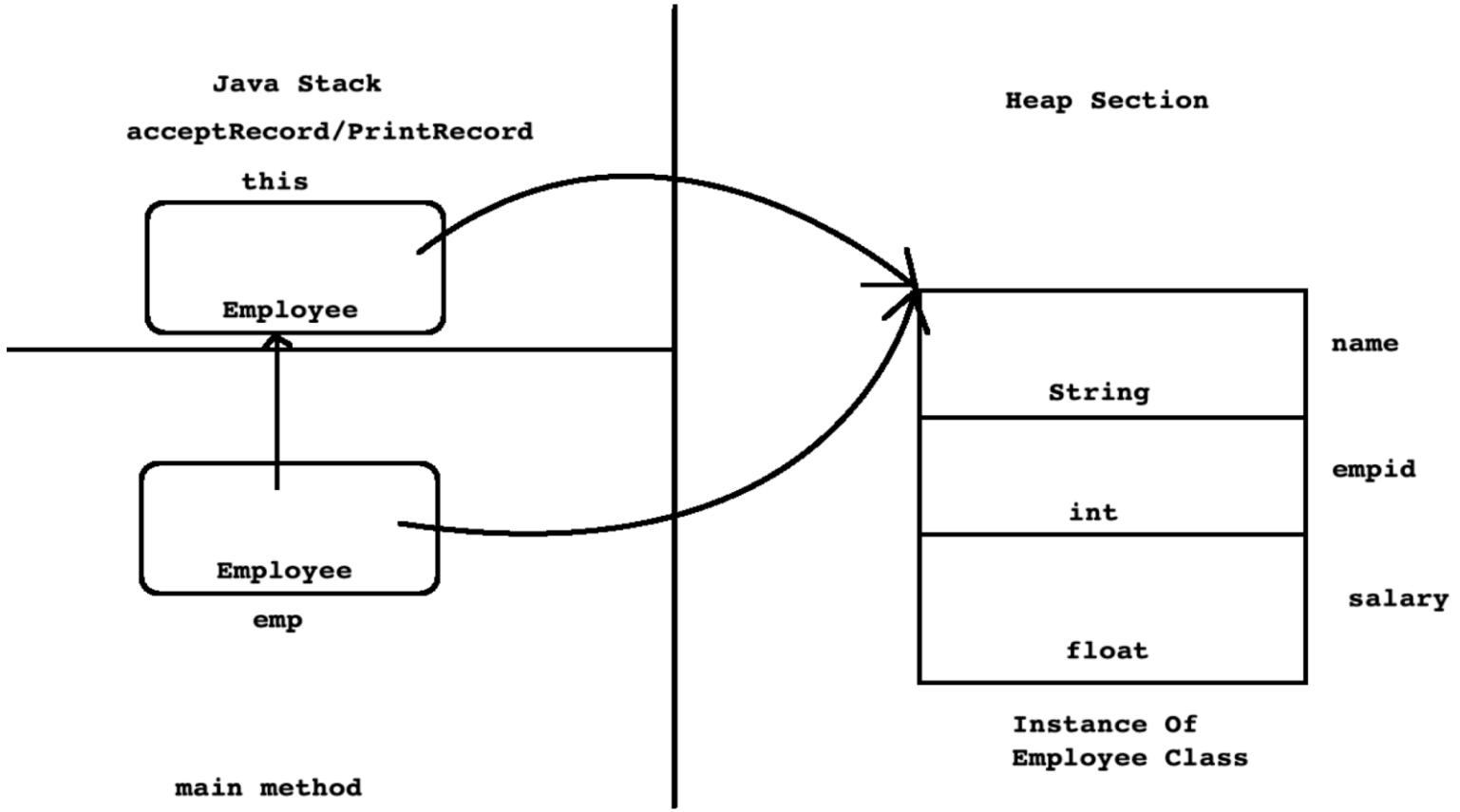
- **this is a keyword** in Java which is designed to store reference of current/calling instance.

- **Using this reference, non static fields and non static methods are communicating with each other. Hence this reference is considered as a link/connection between them.**

- Definition
  - Ø **“this" is implicit reference variable that is available in every non static method of class which is used to store reference of current/calling instance.**

- Inside method, to access members of same class, use this keyword is optional

this reference



this reference

- If name of local variable/parameter and name of field is same then preference is always given to the local variable.

```
class Employee{
    private String name;
    private int empid;
    private float salary;
    public void initEmployee(String name, int empid, float salary ){
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
}
```

Constructor

- If we want to initialize instance then we should define constructor inside class.
- Constructor look like method but it is not considered as method.
- It is special because:
  - Its name is same as class name.
  - It doesn't have any return type.
  - It is designed to be called implicitly
  - It is called once per instance.
- We can not call constructor on instance explicitly

  Employee emp = new Employee();
  emp.Employee( );//Not Ok

- **Types of constructor:**
  1. Parameterless constructor
  2. Parameterized constructor
  3. Default constructor .

Parameterless Constructor

- If we define constructor without parameter then it is called as parameterless constructor.
- It is also called as zero argument / user defined default constructor.
- If we create instance without passing argument then parameterless constructor gets called.

```
public Employee( ){
    //TODO
}
```

```
Employee emp = new Employee( ); //Here on instance parameterless ctor will call.
```

Parameterized Constructor

- If we define constructor with parameter then it is called as parameterized constructor.
- If we create instance by passing argument then parameterized constructor gets called.

```
public Employee( String name, int empid, float salary ){
    //TODO
}
```

```
Employee emp = new Employee( "ABC",123, 8000 ); //Here on instance parameterized ctor will call.
```

Default Constructor

- If we do not define any constructor inside class then compiler generate one constructor for the class by default. It is called default constructor.

- Compiler generated default constructor is parameterless.
- Compiler never generate default parameterized constructor. In other words, if we want to create instance by passing arguments then we must define parameterized constructor inside class.

Constructor Chaining

- We can call constructor from another constructor. It is called constructor chaining.
- For constructor chaining, we should use this statement.
- this statement must be first statement inside constructor body.
- Using constructor chaining, we can reduce developers effort.

```
class Employee{
    //TODO : Field declaration
    public Employee( ){
        this( "None", 0, 8500 );    //Constructor Chaining
    }
    public Employee( String name, int empid, float salary ){
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
}
```

Thank you.