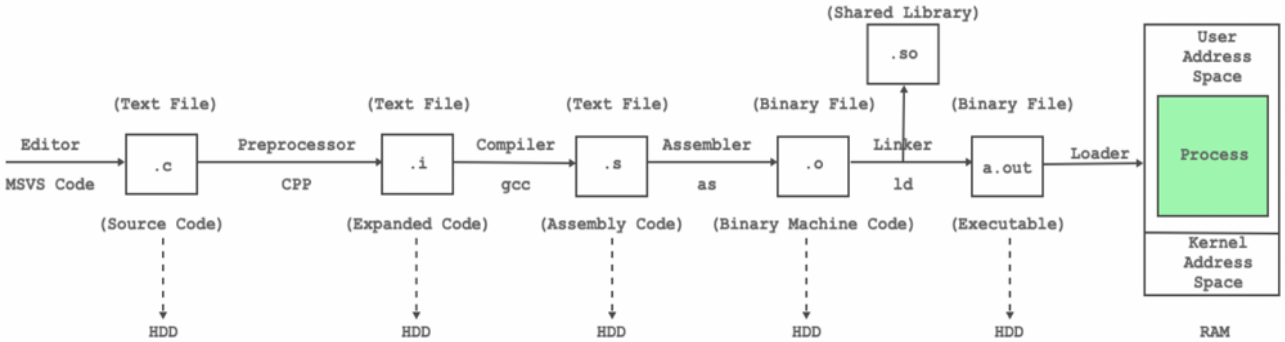
Session Overview

- Flow of execution in C/C++
- Machine Language Code
- Assembly Language Code
- High Level Language Code
- Programming Language
- Framework
- Technology
- Platform
- A short history of Java
- Java Introduction
- Conceptual Diagram of Java
- Java Editions/Development Platforms
- What is API?
- Java Version History
- Software Development Kit
- Java Development Kit
- Java Runtime Environment
- JDK Distributions
- src.zip vs rt.jar vs Java api docs
- ”Hello World!!”
- Java Application Execution Flow
- Overview of JVM Architecture

Flow of Execution in C/C++

```
// File Name: main.c

#include <stdio.h>
int main( void ) {
    printf("Hello, World!\n");
    return 0;
}
```

```
# Preprocessed file
gcc -E main.c -o main.i //main.i is a text file

# Assembly file
gcc -S main.c -o main.s //main.s is a text file

# Object file
gcc -c main.c -o main.o //main.o is a binary file. Use objdump/readelf/nm to view

# Executable
gcc main.c -o main.out //main.out is a binary file. Use objdump/readelf/nm to view
```



Machine Language Code

- Consider "Hello,World!" program using machine language.

```
sandeep@Sandeeps-MacBook-Air test % ls
main.c
sandeep@Sandeeps-MacBook-Air test %
sandeep@Sandeeps-MacBook-Air test % gcc --save-temps main.c
sandeep@Sandeeps-MacBook-Air test %
sandeep@Sandeeps-MacBook-Air test % ls
a.out   main.bc main.c  main.i  main.o  main.s
sandeep@Sandeeps-MacBook-Air test %
sandeep@Sandeeps-MacBook-Air test % objdump -s main.o
```

```
main.o: file format mach-o 64-bit x86-64
Contents of section __TEXT,__text:
 0000 554889e5 4883ec10 c745fc00 00000048  UH..H....E.....H
 0010 8d3d0f00 0000b000 e8000000 0031c048  .=...........1.H
 0020 83c4105d c3                          ...].
Contents of section __TEXT,__cstring:
 0025 48656c6c 6f20576f 726c6421 2100      Hello World!!.
Contents of section __LD,__compact_unwind:
 0038 00000000 00000000 25000000 00000001  ........%.......
 0048 00000000 00000000 00000000 00000000  ................
Contents of section __TEXT,__eh_frame:
 0058 14000000 00000000 017a5200 01781001  .........zR..x..
 0068 100c0708 90010000 24000000 1c000000  ........$.......
 0078 88ffffff ffffffff 25000000 00000000  ........%.......
 0088 00410e10 8602430d 06000000 00000000  .A....C.........
```

- **Pros**
  - Direct Execution
  - Performance
- **Cons**
  - Complexity
  - Portability
  - Maintenance
  - Development time

Assembly Language Code

- Consider "Hello,World!" program using assembly language.

```
    .section    __TEXT,__text,regular,pure_instructions
    .build_version macos, 12, 0 sdk_version 13, 1
    .globl  _main                           ## -- Begin function main
    .p2align    4, 0x90
_main:                                  ## @main
    .cfi_startproc
## %bb.0:
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    movl    $0, -4(%rbp)
    leaq    L_.str(%rip), %rdi
    movb    $0, %al
    callq   _printf
    xorl    %eax, %eax
    addq    $16, %rsp
    popq    %rbp
    retq
    .cfi_endproc
                                        ## -- End function
    .section    __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "Hello World!!"

.subsections_via_symbol
```

- **Pros**
  - Close to hardware
  - Performance
- **Cons**
  - Complexity
  - Portability
  - Maintenance
  - Development time

High Level Language Code

- Consider "Hello,World!" program using C language.

```c
#include<stdio.h>
int main( void ){
    printf("Hello World!!");
    return 0;
}
```

- **Pros**
  - Ease of Use
  - Faster Development
  - Portability
  - Error handling
- **Cons**
  - Performance
  - Memory Usage
  - Learning Curve

Programming Language

- Syntax
- Semantics
- Data Types
- Operator Set
- Built-in Features
- Use Case:
  - Console User Interface Application(CUI)
  - Graphical User Interface Application(GUI)
  - Shared libraries(.jar)
- ***We can create CUI,GUI & libraries using language but generally it is used to implement business logic.***
- Example: C, C++, Java, C#, Python, GO, Ruby etc.

Framework

- ***Set of ready-made libraries on the top of it we can develop application.***
- Why Frameworks?
  - Speeds up development
  - Reduces errors
  - Solve Specific problem
- Example
  - Logging Framework: Apache Log4j
  - Unit Testing Framework: Junit
  - MVC Based Web Application Framework: Apache Struts
  - Automatic Persistence Framework: Hibernate

Technology

- Development tools
- Deployment tools
- Framework
- Techniques
- ***In general, Technology help us to develop application/software.***
- Note: Every language can be considered as technology but every technology can not be considered as language.
- Example: Servlet/JSP, ASP.NET etc.

Platform

- **A *platform* is the hardware or software environment in which a program runs.**
- Most platforms can be described as a combination of the operating system and underlying hardware. Also called as ***Hardware Based Platform***.
- Example: Microsoft Windows, Linux, Solaris OS, and Mac OS.
- Platforms that do not require specific hardware but are built on top of existing operating systems and hardware are called as ***Software Only Platform***.
- Example: Java, Microsoft.NET etc.
- In general, software-only platform provides tools, API, Runtime environment.

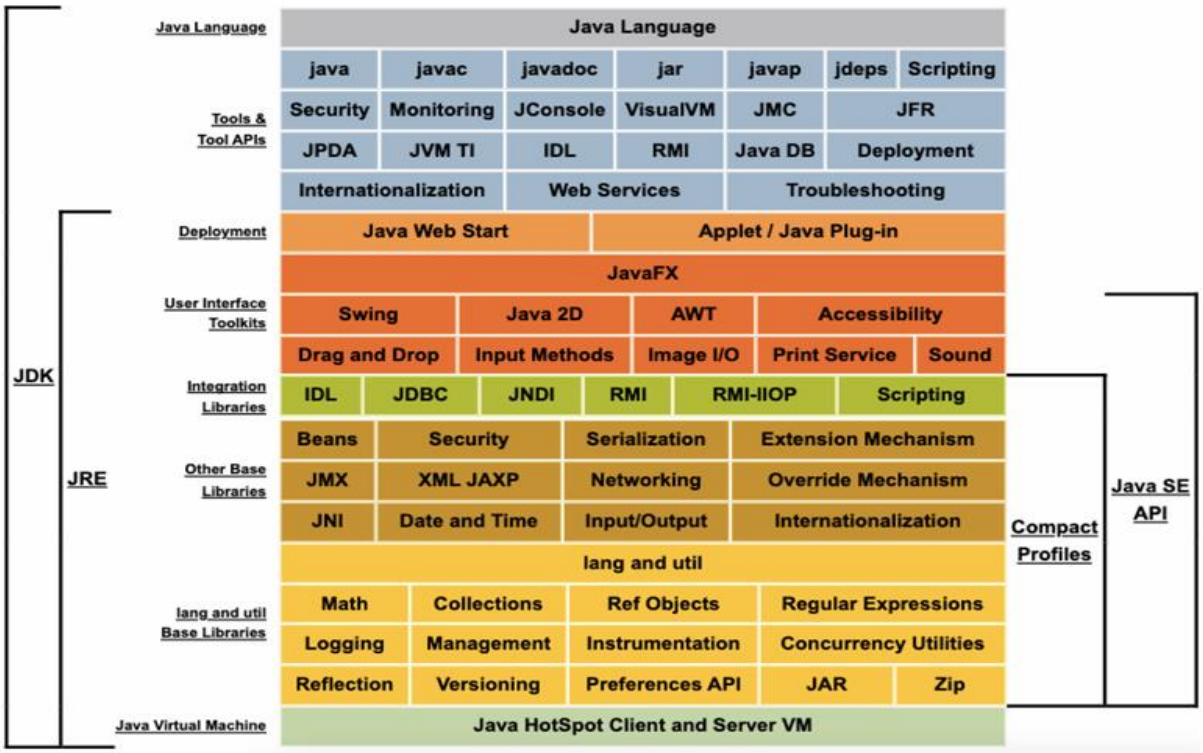
A Short History of Java

- **Birth**: 1991.
- **Origin**: Sun Microsystems.
- **Green Project**: To explore opportunities in the consumer electronics market.
- **Green Team**(Key Members): James Gosling, Patrick Naughton, Mike Sheridan.
- **From Oak to Java**: The original language, named Oak, was later renamed to Java.
- **Programming Paradigm**: Object Oriented.
- **The "*7" Device(1992)**: To showcase the technology potentials.
- **Failure of "*7"**: Time-Warner denied set-top box OS and video-on-demand technology for demo.
- **Breakthrough with the Web(1994)**: WebRunner (a Web browser), Applet.
- **First public implementation**: Java 1.0 in 1996.
- **Acquisition of Java**: Oracle Corporation acquired Sun Microsystems in 2010.
- **Slogan**: "Write Once, Run Anywhere".

Introduction

- Java is a product of **SUN/Oracle**.
- Java **language** is both, **technology** as well as **platform**.
- Java's standardization is managed through the **Java Community Process (JCP)**.
- Extension of Java source file **".java"**.
- Java is **object oriented** but also support **procedural** & **functional** programming paradigms.
- Java is **statically typed** language. It means type checking is done at compile time.
- Java is case sensitive language.
- Java, Kotlin, Scala, Groovy are some of the **JVM based** languages.

Conceptual Diagram of Java



Java Editions / Development Platforms

1. **Java Standard Edition(Java SE)**
   - It is also called as Core Java.
   - Key components are Java Application Programming Interface(API) and Java Virtual Machine(JVM)
   - Generally used for standalone applications(Desktop applications, command-line tools etc.)

2. **Java Enterprise Edition(Java EE)**
   - It is also called as Advanced Java / Web Java / Java EE / JEE. (Now Days Jakarta EE)
   - Key components are Servlets, JSP, Filter, JPA, JTA, JMS, EJB, JSF, Java Mail etc.
   - Generally it is used for web application and distributed application.

3. **Java Micro Edition(Java ME)**
   - Generally it is used to develop application for mobile phones, embedded systems and IoT devices.

4. **JavaFX**
   - Generally it is used for building rich, modern user interfaces for desktop applications.

5. **Java Card**
   - Generally it is used for developing application for smart card and secure IoT Devices.

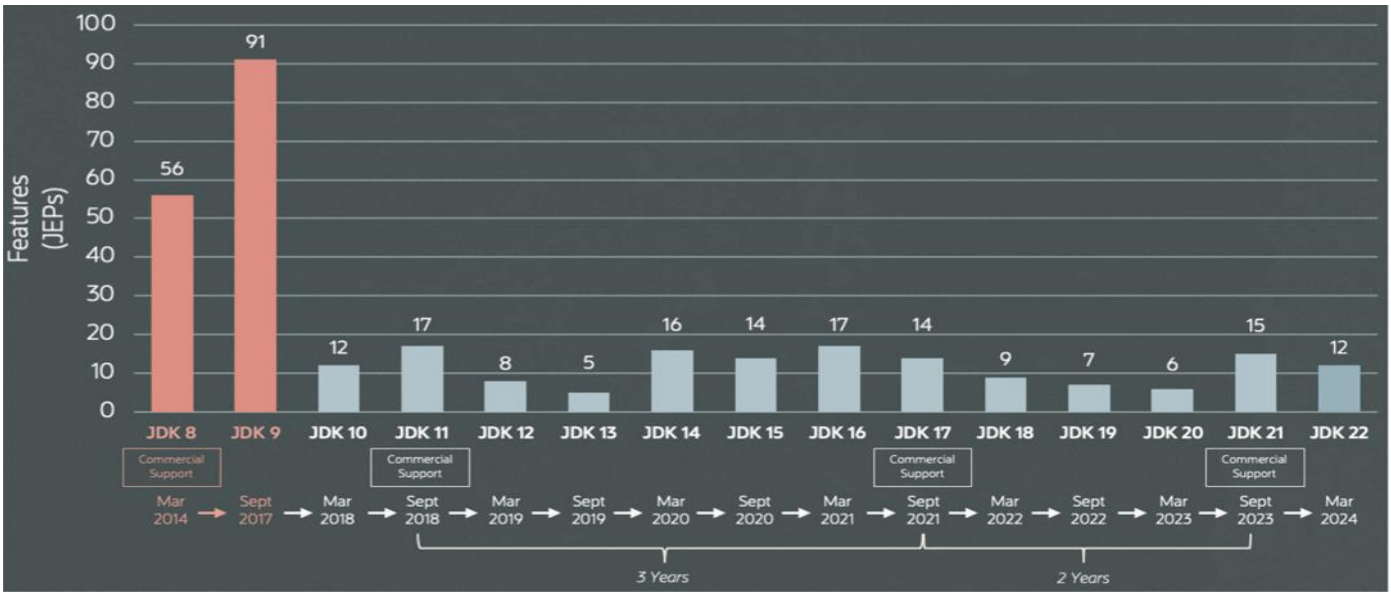
What is API?

Java Version History

Java SE Version	Release Year	Approximate Number of Major Features Added
Java SE 1.0	1996	10+
Java SE 1.1	1997	15+
Java SE 1.2	1998	25+
Java SE 1.3	2000	20+
Java SE 1.4	2002	30+
Java SE 5.0	2004	40+
Java SE 6	2006	50+
Java SE 7	2011	60+

Reference : https://javaalmanac.io/features/

Java Version History



Software Development Kit(SDK)

- To develop software, a developer must install the SDK on their machine.
- SDK = Dev Tools + API Docs + Supporting Libraries + Execution Environment.
- Software Development Kit typically includes:
    1. **Development Tools**
        - Tools to compile, build, test and debug application.
    2. **Documentation**
        - References that explain how to use SDK.
    3. **Libraries**
        - Pre-written code which minimizes developer efforts.
    4. **Execution environment**
        - A platform where we can deploy and test the application.
    5. **Code Samples**
        - Example projects that demonstrates how to implement certain features using the SDK.

Java Development Kit(JDK)

- To develop Java software, a developer must install the JDK on their machine.
- But what is JDK?
    - Java SDK = Java Dev Tools + Java API Docs + Java Supporting Lib. + Java Execution Environment.
    - JDK = Java Dev Tools + Java API Docs + rt.jar + Java Virtual Machine.
    - JDK = Java Dev Tools + Java API Docs + JRE(rt.jar + JVM).
- Java Development Kit typically includes:
    1. **Java Development Tools**
        - javac, java, javap, jstack, jdb etc.
    2. **Java API Documentation**
        - https://docs.oracle.com/javase/8/docs/api/ (online help)
        - https://www.oracle.com/in/java/technologies/javase-jdk8-doc-downloads.html (For Download)
    3. **Java API Libraries**
        - rt.jar
    4. **Execution environment**
        - Java virtual machine
    5. **Code Samples**
        - src.zip

Java Runtime Environment(JRE)

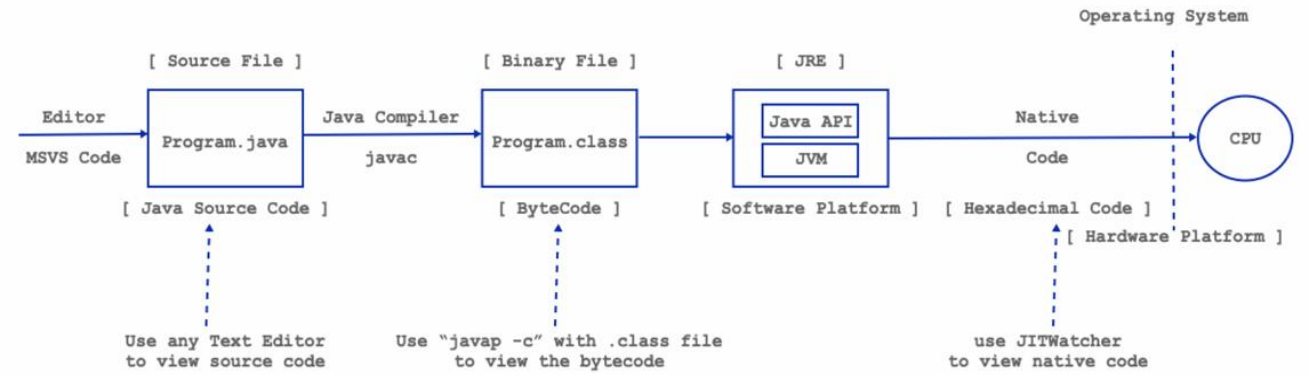
- To execute/run Java application on developer's machine / client's machine, we require Java Runtime Environment(JRE).
- The JRE comes with the JDK by default, so developers do not need to download it separately.
- On a client's machine, we must first download and install the JRE.
- Components of Java Runtime Environment:
    1. Java Class Library(rt.jar)
    2. Java Virtual Machine(JVM)

JDK Distributions

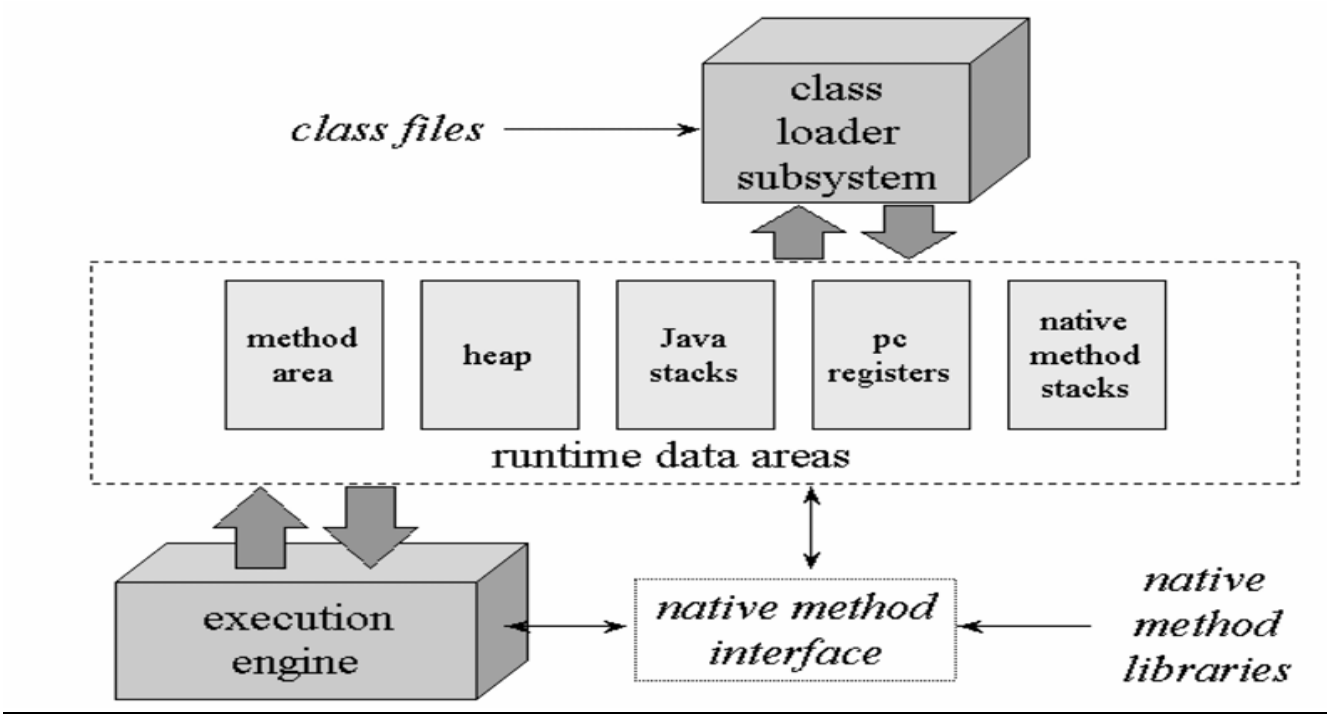
Sr.No.	Distribution	Provider
1	Oracle JDK	Oracle Corporation
2	Graal VM	Oracle Corporation
3	OpenJDK	Oracle Corporation
4	Adoptium Eclipse Temurin	Eclipse Foundation
5	Azul Zulu	Azul Systems
6	Azul Zing	Azul Systems
7	Liberica JDK	BellSoft
8	IBM Semeru Runtime	IBM
9	Amazon Corretto	AWS
10	Microsoft Build of OpenJDK	Microsoft
11	Alibaba Dragonwell	Alibaba
12	Red Hat OpenJDK	Red Hat
13	SapMachine	SAP

src.zip vs rt.jar vs Java api docs

Java Application Execution Flow

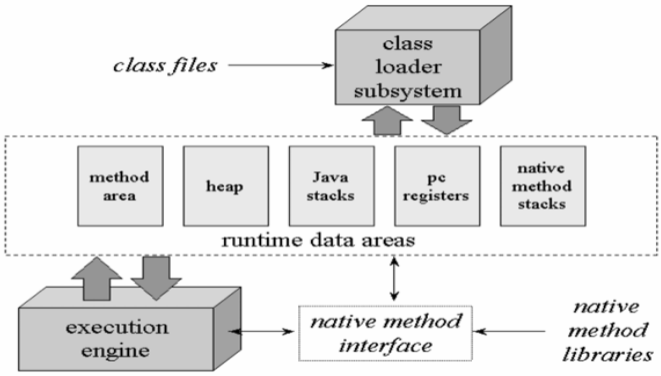


Overview of JVM Architecture



Components of JVM

- **Class loader subsystem**
    1. Bootstrap class loader
    2. Extension class loader
    3. System class loader
    4. Custom class loader
- **Runtime data areas**
    1. Method area
    2. Heap
    3. Java Stacks
    4. PC Register
    5. Native method stacks
- **Execution engine**
    1. Interpreter
    2. Just In Time Compiler
    3. Garbage Collector



Java Buzzwords

- **Java Buzzwords are Java language marketing words.**

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

Java Modifier

- A keyword which is used to change the behavior of variable, field, method, class etc.
- There are 12 modifiers in Java:

1. private	:	Access Modifier
2. protected	:	Access Modifier
3. public	:	Access Modifier
4. static	:	Non Access Modifier
5. final	:	Non Access Modifier
6. abstract	:	Non Access Modifier
7. interface	:	Non Access Modifier
8. transient	:	Non Access Modifier
9. synchronized	:	Non Access Modifier
10. volatile	:	Non Access Modifier
11. strictfp	:	Non Access Modifier
12. native	:	Non Access Modifier

Access Modifier

- Modifier which is used to control visibility of the members of the class/enum/interface.
- 4 Access modifiers in Java
  1. private
  2. package level private(also called as default)
  3. protected
  4. public

	Same Package			Different Package	
Access Modifier	Same Class	Sub Class	Non Sub Class	Sub Class	Non Sub Class
private	A	NA	NA	NA	NA
package level private	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

Java Virtual Machine Threads

```
sandeep@Sandeeps-MacBook-Air Day_1.1 % cat Program.java
class Program{
    public static void main( String[] args )throws Exception{
        System.out.println("Press any key to continue...");
        System.in.read( );
    }
}
sandeep@Sandeeps-MacBook-Air Day_1.1 %
sandeep@Sandeeps-MacBook-Air Day_1.1 % jcmd
44201 sun.tools.jcmd.JCmd
44173 Program
sandeep@Sandeeps-MacBook-Air Day_1.1 %
sandeep@Sandeeps-MacBook-Air Day_1.1 % jstack 44173
```

Java Virtual Machine Threads

1. Attach Listener
    - A daemon thread responsible for handling the attachment of debuggers.
2. Service Thread
    - A daemon thread that handles JVM service tasks
3. C2 CompilerThread0
4. C2 CompilerThread1
5. C1 CompilerThread2
    - These are compiler threads responsible for compiling Java bytecode to native code.
6. Signal Dispatcher
    - A daemon thread that handles OS signals.
7. Main
    - main application thread.
8. VM Thread
    - A JVM thread used for internal JVM tasks.
9. GC task thread#0 (ParallelGC)
10. GC task thread#1 (ParallelGC)
11. GC task thread#2 (ParallelGC)
12. GC task thread#3 (ParallelGC)
    - Threads dedicated to garbage collection

Entry Point Method

- "main" method is considered as entry point method in java.

- Legal main method signatures
    1. public static void main(string[] args)
    2. public static void main(string args[ ])
    3. public static void main(string. . . args)

- Java compiler do not call main method. With the help of main thread JVM invoke main method in Java.

- If we try to execute class which do not contain main method then we get below error:
    - **Error: Main method not found in class Program, please define the main method as:**
      public static void main(String[] args)

- We can overload main method in Java.

Meaning of System.out.println

```java
package java.lang;
import java.io.*;

public final class System {
  public final static InputStream in = null;
  public final static PrintStream out = null;
  public final static PrintStream err = null;

  public static void exit(int status) {
      Runtime.getRuntime().exit(status);
  }

  public static void gc() {
      Runtime.getRuntime().gc();
  }
}
```

```java
package java.io;

public class PrintStream {

    public void print(String s) {
        // Method to print a string without a newline
    }

    // Other overloaded print(...) methods

    public void println(String s) {
        // Method to print a string followed by a newline
    }

    // Other overloaded println(...) methods

    public PrintStream printf(String format, Object... args) {
        // Method to print formatted strings
        return this;
    }

    // Other overloaded printf() methods
}
```

Data Types

- Java is statically as well as strongly typed language.

- Data Type of any variable describes following properties:
    - **Memory Allocation:** How much memory is required to store the data?
    - **Type of Data:** What kind of data is allowed to store inside variable?
    - **Operation:** Which operations can be performed on the data stored in memory?
    - **Range of data:** Range of values that can be stored in a variable?

- Classification of data types:
    1. **Primitive Data Types**(also called as value types)
    2. **Non Primitive Data Types**(also called as reference types)

Classification of Data Types

- Primitive Data Types
    1. boolean
    2. byte
    3. char
    4. short
    5. int
    6. float
    7. double
    8. Long

- Variable of above type can contain only value. Hence such type is also called as value type.

- Non Primitive Data Types
    1. Interface
    2. Class
    3. Array
    4. Enum

- Variable of above type can contain only object reference/reference. Hence such type is also called as reference type.

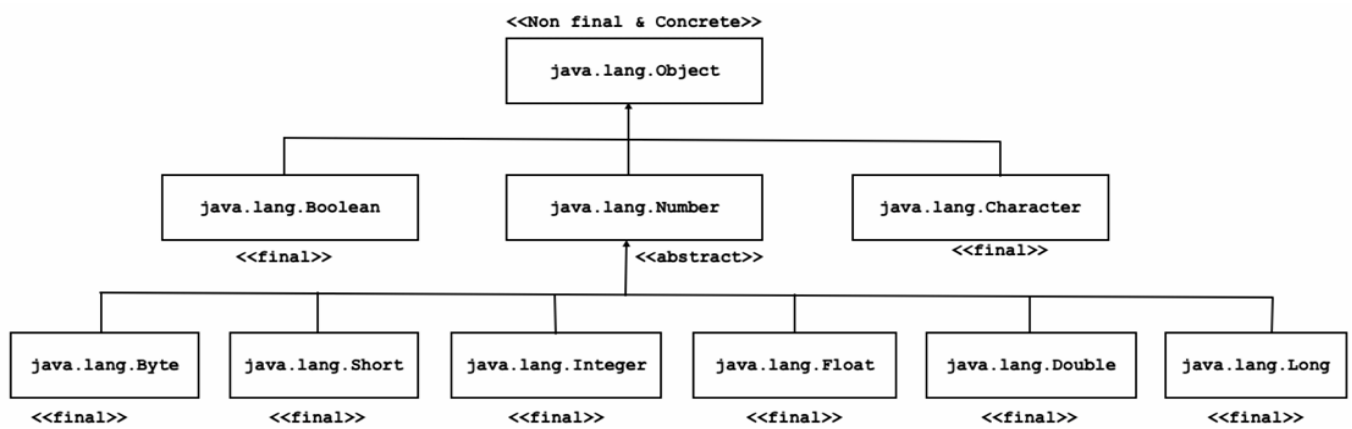
Primitive Data Type - Reference: https://docs.oracle.com/javase%2Ftutorial%2F/java/n utsandbolts/datatypes.html

Non Primitive Data Type - https://docs.oracle.com/javase/tutorial/reflect/clas s/index.html

Primitive Data Types

Sr.No.	Primitive Type	Size(In Bytes)	Default Value(For Fields)	Wrapper Class
1	boolean	Not defined	false	java.lang.Boolean
2	byte	1	0	java.lang.Byte
3	char	2	'\u0000'	java.lang.Character
4	short	2	0	java.lang.Short
5	int	4	0	java.lang.Integer
6	float	4	0.0f	java.lang.Float
7	double	8	0.0d	java.lang.Double
8	long	8	0L	java.lang.Long

Wrapper Class Hierarchy



Overview of String

- String is not a primitive / built-in type in Java.
- String is a final class declared in java.lang package.
- Since String is a class, it is considered as non primitive type/reference type.
- We can create Instance of String using new operator as well as without new operator.

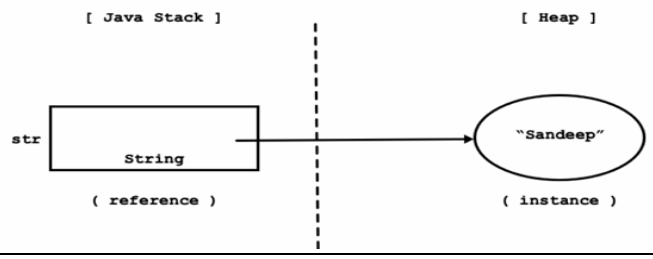
- Example 1:
  - String str = new String("Sandeep"); //OK
    - **str** is called as object reference / simply reference.
    - **new String("Sandeep")** is called as instance.

- Example 2:
  - String str = "Sandeep"; //OK
    - **str** is called as object reference / simply reference.
    - **"Sandeep"** is called as String constant/literal.

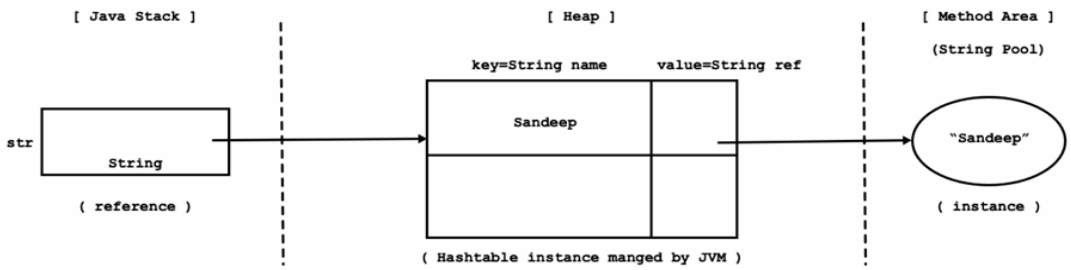
Memory Representation

Example 1:

- String str = new String("Sandeep"); //OK
  - **str** is called as object reference / simply reference.
  - **new String("Sandeep")** is called as instance.

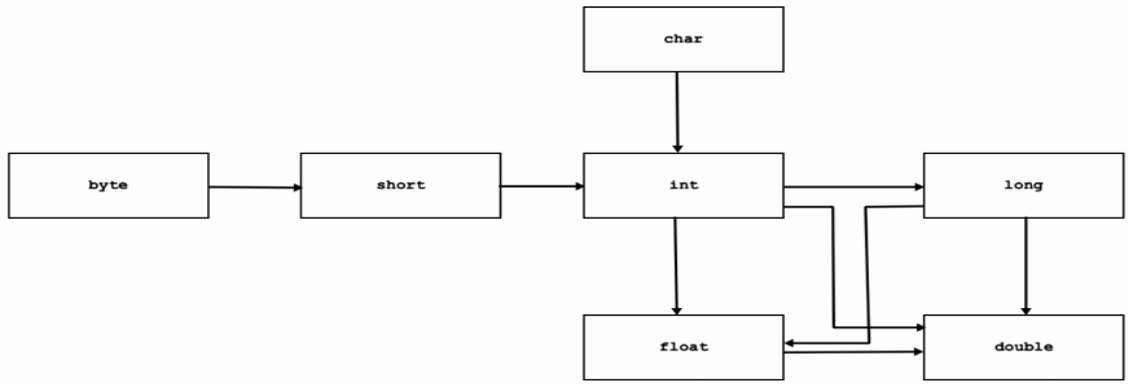


- Example 2:
  - String str = "Sandeep"; //OK
    - **str** is called as object reference / simply reference.
    - **"Sandeep"** is called as String constant/literal.

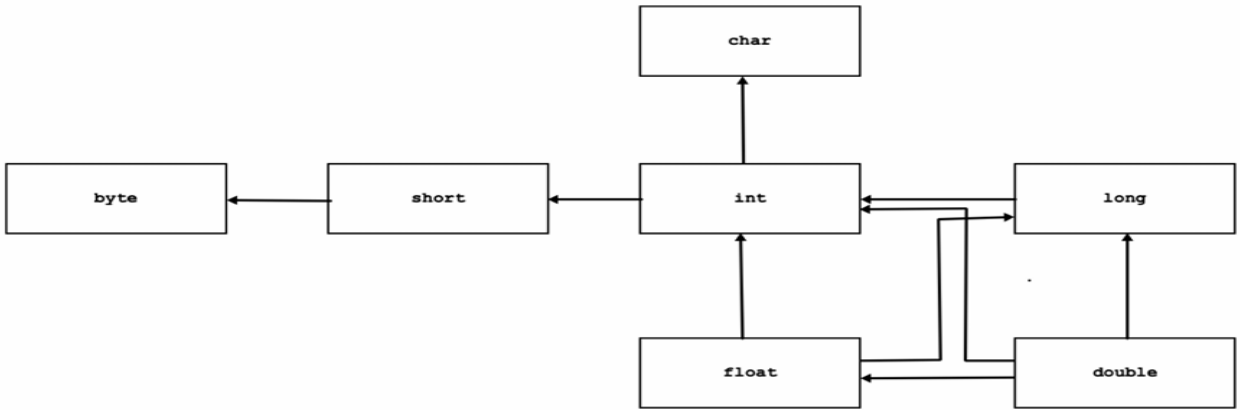


Widening Conversion

- **Process of converting value of variable of narrower type into wider type is called as widening.**



Narrowing Conversion



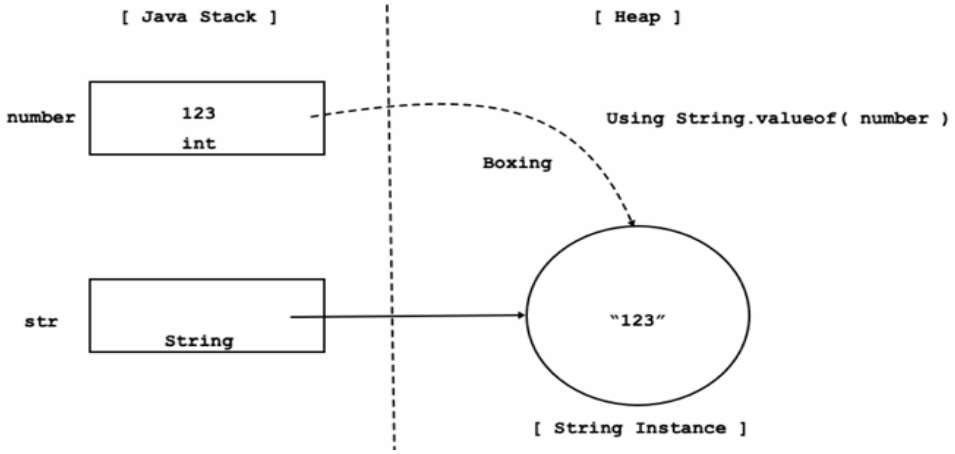
Boxing

- Using toString() method of Wrapper class or String.valueOf() method, we can convert value of any primitive type into String.

```
1. String s1 = Boolean.toString( true ); or String s1 = String.valueOf( true );
2. String s2 = Character.toString( 'A' ); or String s2 = String.valueOf( 'A' );
3. String s3 = Integer.toString( 123 ); or String s3 = String.valueOf( 123 );
4. String s4 = Float.toString( 123.45f ); or String s4 = String.valueOf( 123.45f );
5. String s5 = Double.toString( 123.45d ); or String s5 = String.valueOf( 123.45d );
```

- Boxing is the process of converting value of variable of primitive type into non primitive type.

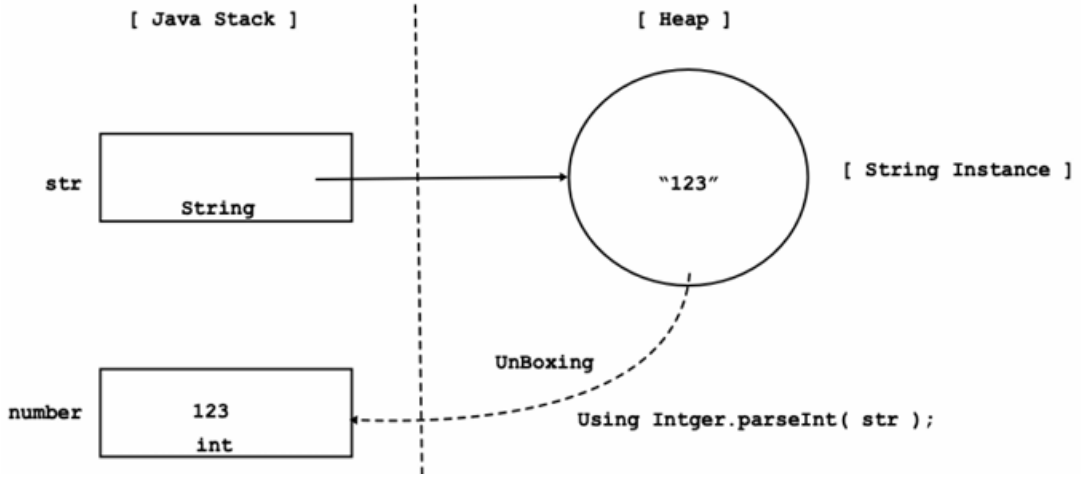
```
int number = 123;
String str = String.valueOf( number ); //Boxing
```



Unboxing

- **Using parseXXX () method of Wrapper class, we can convert state of String into primitive value.**
  1. boolean b = Boolean.parseBoolean("true");
  2. int i = Integer.parseInt("123");
  3. float f = Float.parseFloat("123.45f");
  4. double d = Double.parseDouble("123.45d");
  5. int number = Integer.parseInt("1A2b3C");//NumberFormatException
- **Unboxing is the process of converting value of variable of non primitive type into primitive type.**

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



Command line Arguments

Program.java

Program.java > Program > main(String[])

```java
class Program{
  public static void main(String[] args){
    System.out.println( args[ 0 ] );
  }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program Hello
Hello
sandeep@Sandeeps-MacBook-Air Day_1 %
```

➢

Program.java

Program.java > Program > main(String[])

```java
class Program{
  public static void main(String[] args){
    System.out.println( args[ 0 ] );
    System.out.println( args[ 1 ] );
  }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20
10
20
```

➢

Program.java 2

Program.java > Program

```java
class Program{
  public static void main(String[] args){
    int num1 = args[ 0 ];
    int num2 = args[ 1 ];
    int result = num1 + num2;
    System.out.println("Result::"+result);
  }
}
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
Program.java:3: error: incompatible types: String cannot be converted to int
    int num1 = args[ 0 ];
               ^
Program.java:4: error: incompatible types: String cannot be converted to int
    int num2 = args[ 1 ];
               ^
2 errors
```

Program.java

```java
class Program{
  public static void main(String[] args){
    int num1 = Integer.parseInt( args[ 0 ] );
    int num2 = Integer.parseInt( args[ 1 ] );
    int result = num1 + num2;
    System.out.println("Result::"+result);
  }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
        at Program.main(Program.java:3)
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20
Result::30
sandeep@Sandeeps-MacBook-Air Day_1 %
```

➢

Program.java

```java
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[ 1 ] );
    double result = num1 + num2 + num3;
    System.out.println("Result::"+result);
  }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20.1f 30.2d
Result::50.2000003814697
```

Stream Associated with Console

- C Programming Language
  1. stdin
  2. stdout
  3. stderr
- C++ programming language
  1. cin
  2. cout
  3. cerr
  4. clog
- Java programming language
  1. System.in
  2. System.out
  3. System.err

- System.in represents keyboard.
- System.out represents Monitor
- System.err represents Monitor. Intended for error messages.

User Input using Scanner

- **Scanner is a final class which is declared in java.util package.**
- **Instantiation:**

➢ **Scanner sc = new Scanner(System.in);**

- **Methods:**

  ✓ **public String nextLine();**
  ✓ **public int nextInt();**
  ✓ **public float nextFloat();**
  ✓ **public double nextDouble();**

Example:

```
Scanner sc = new Scanner( System.in );

String name = sc.nextLine();

int empid = sc.intInt();

float salary = sc.nextFloat();
```

How to get System Date?

- `Import java.util.Date`

```
Date dt = new Date( );
int day = dt.getDate( );
int month = dt.getMonth() + 1;
int year = dt.getYear(); + 1900;
```

- `Import java.util.Calendar`

```
Calendar c = Calendar.getInstance();
int day = c.get(Calendar. DAY_OF_MONTH);
int month = c.get(Calendar.MONTH) + 1;
int year = c.get(Calendar.YEAR);
```

✓ **Import java.time.LocalDate**

```
LocalDate ldt = LocalDate.now();

int day = ldt.getDayOfMonth( );

int month = ldt.getMonthValue();

int year = ldt.getYear();
```

How to solve problem using Object Oriented Paradigm?

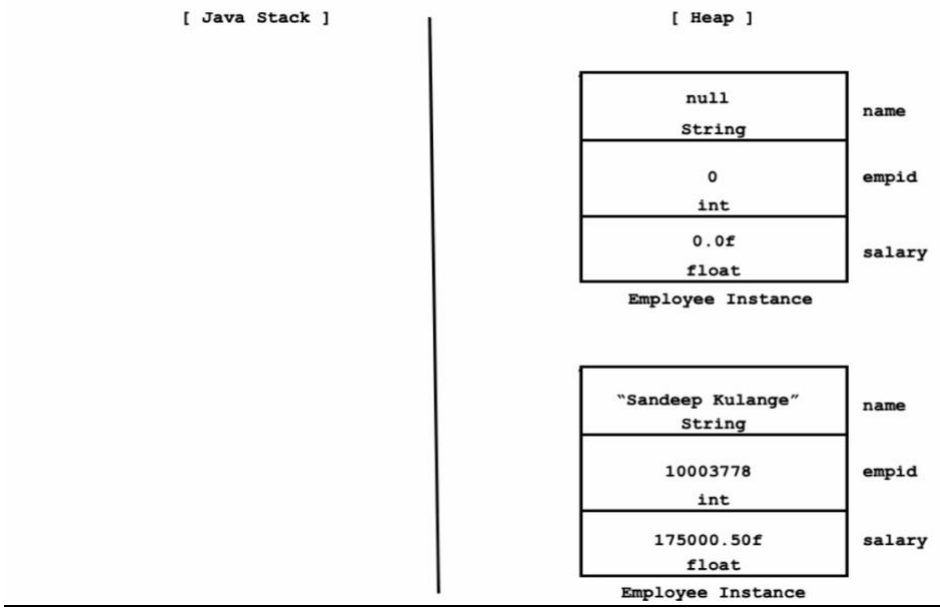
- **Problem Statement:** Write a program to accept and print employee record.
- First analyse problem statement and group related data element(s) together.
    1. To group related data elements together define class.

```
class Sample{
  DataType varName1;          //Non static field
  static DataType varName2;   //Static field
}
```

```
class Employee{
    String name;
    int empid;
    float salary;
}
```

- Non static field is called as instance variable.
- Static field is called as class level variable

- **Problem Statement:** Write a program to accept and print employee record.
- To store value inside name, empid and salary, it must get space inside memory.
- Since name, empid and salary are non static field declared inside Employee class, it will get space after creating object/instance of the class.
- new is operator in java, which is used to create instance of class on heap section of JVM. Consider below code:
    - `new Employee( ); //Instance of class Employee`
    - `new Employee( "Sandeep Kulange", 10003778, 175000.50f); //Instance of class Employe`

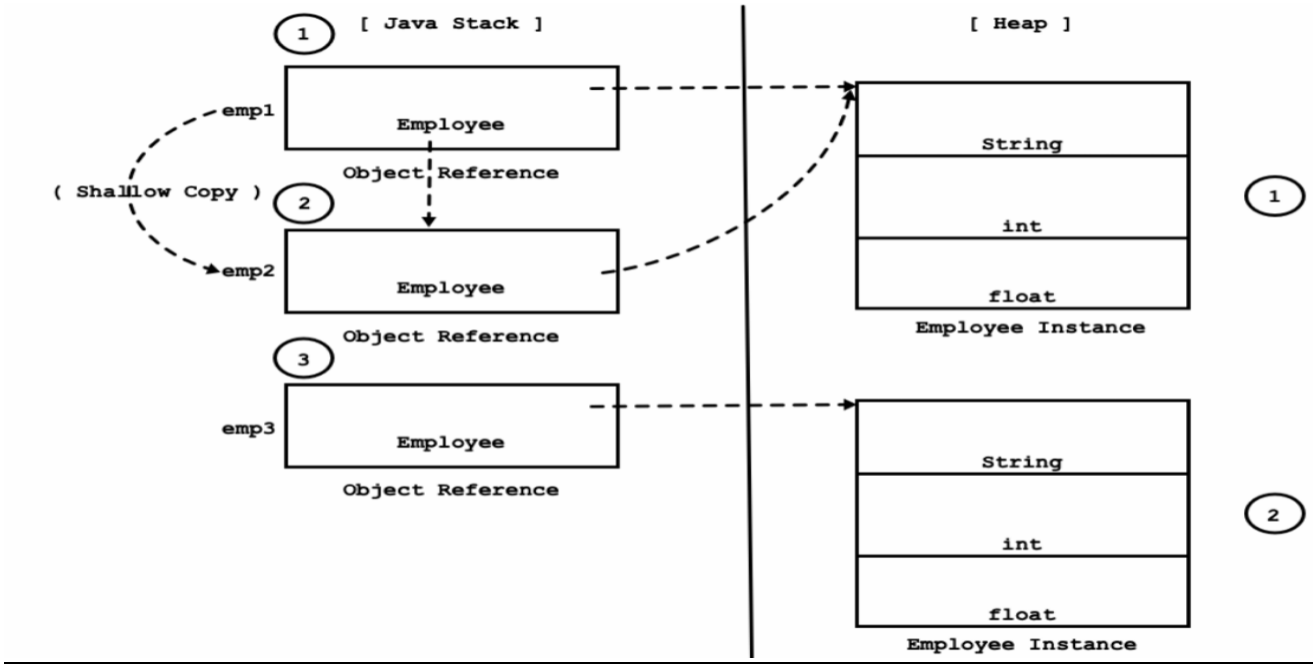


- **Problem Statement:** Write a program to accept and print employee record.
- If we want to perform some operations on data stored inside instance then we should create reference of it.
- In Java, reference is also called as object reference.
- We can declare reference as a method local variable / field of the class.
- How to declare local reference variable:

```
Employee emp1;
emp = new Employee( ); //OK
Employee emp2 = new Employee( "Sandeep Kulange", 10003778, 175000.50f); //OK
```

- For more clarity, consider below example. Identify how many instances and how many references?

```
Employee emp1 = new Employee( );
Employee emp2 = emp1;
Employee emp3 = new Employee( );
```



- **Problem Statement:** Write a program to accept and print employee record.
- To process(acceept/print) state of the instance, we should call method on it.
- Consider below code:

```
Employee emp = new Employee( );
emp.acceptRecord( ); //acceptRecord() method is called on emp
emp.printRecord( ); //printRecord() method is called on emp
```

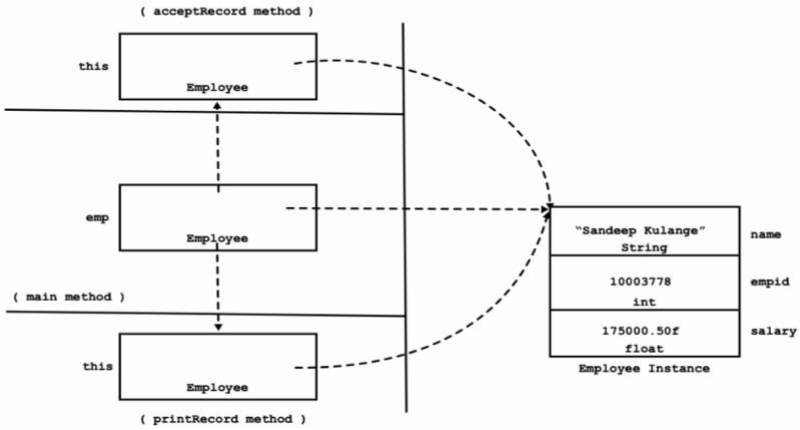
- Process of calling method on instance(actually object reference) is called as message passing.

- **Problem Statement:** Write a program to accept and print employee record.
- To call method on instance, first we must define method inside class.
- Function defined inside class is called method. It can be static or non static.
  - Non static methods are designed to call on instance. Hence it is called as instance method.
  - We can call static methods on instance but it is designed to call on type(class/interface) hence it is called as class level method.

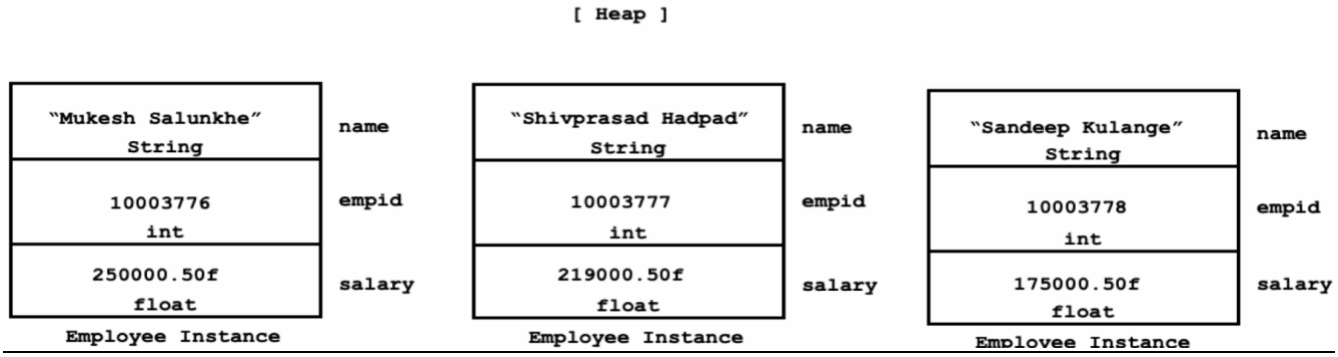
```
class Employee{
    private String name;      //Field
    private int empid;        //Field
    private float salary;     //Field
    void acceptRecord( ){     //Method
        //TODO
    }
    void printRecord( ){      //Method
        //TODO
    }
}
```

```
class Program{
    public static void main( String[] args ){
        Employee emp = new Employee( );
        emp.acceptRecord( );
        emp.printRecord( );
    }
}
```

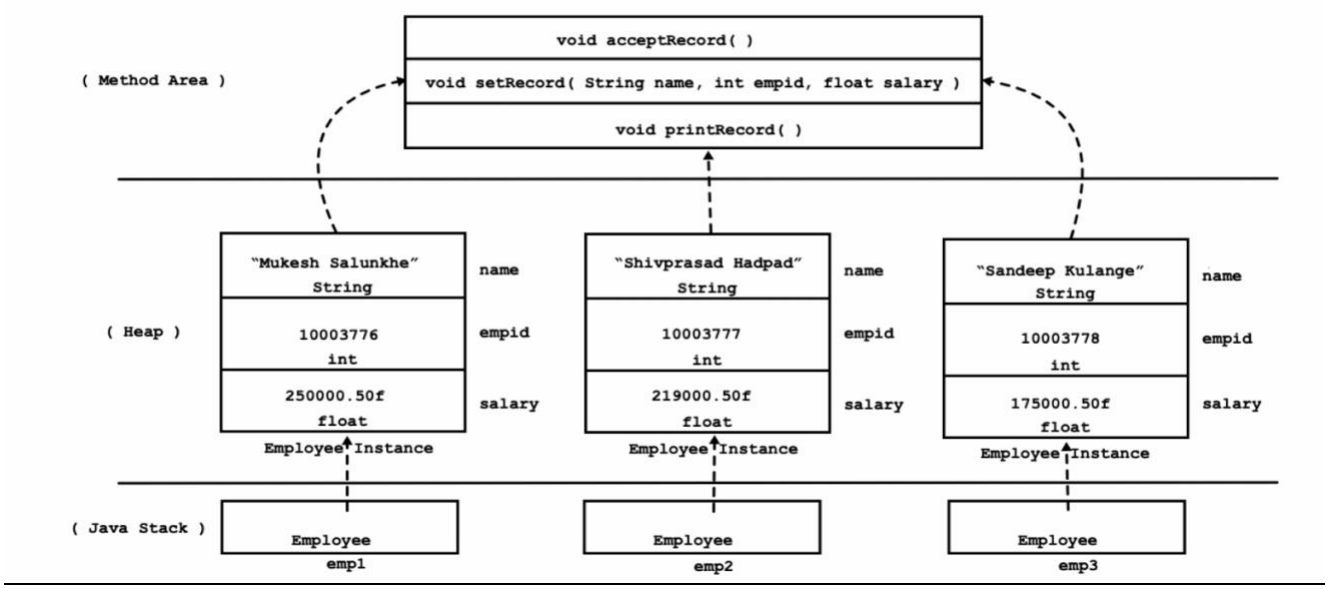
- **Problem Statement:** Write a program to accept and print employee record.
- But now question is how to access non static field/instance variable of the instance inside method?
  - If we call method on instance(actually object reference) then compiler implicitly pass reference of the instance as a argument the method. To catch value of the argument, compiler implicitly declare one parameter is called as this reference.



- Instance variable get space once per instance, according to order of non static fields declared inside class.



- Method do not get space inside instance. All the instances of same class share its behaviour. This sharing is done by passing reference of the instance to the method.



Value Type versus Reference Type

Sr.No.	Value Type	Reference Type
1	Primitive type is also called as value type.	Non primitive type is also called as reference type.
2	boolean, byte, char, short, int, float, double, long are primitive / value types in Java.	Interface, class, enum and array are non primitive / reference types in Java.
3	Variable of value type contains value.	Variable of reference type contains reference.
4	In case of initialization / assignment value gets copied.	In case of initialization / assignment reference gets copied.
5	For the field, default value of primitive /value type variable is zero.	For the field, default value of non primitive /refrence type variable is null.
6	Variable of primitive / value type do not contain null value.	Variable of non primitive / reference type can contain null value
7	To create variable of primitive / value type new operator is not required.	To create instance of non primitive type / reference type new operator is required.
8	Variable of value type get space on Java Stack	Instance of reference type get space on Heap.