# Java

16/11/2023

## Applications

Desktop application - window based program
web application - dynamic webpages
Enterprice - Banking (JDBC)
mobile

Platform

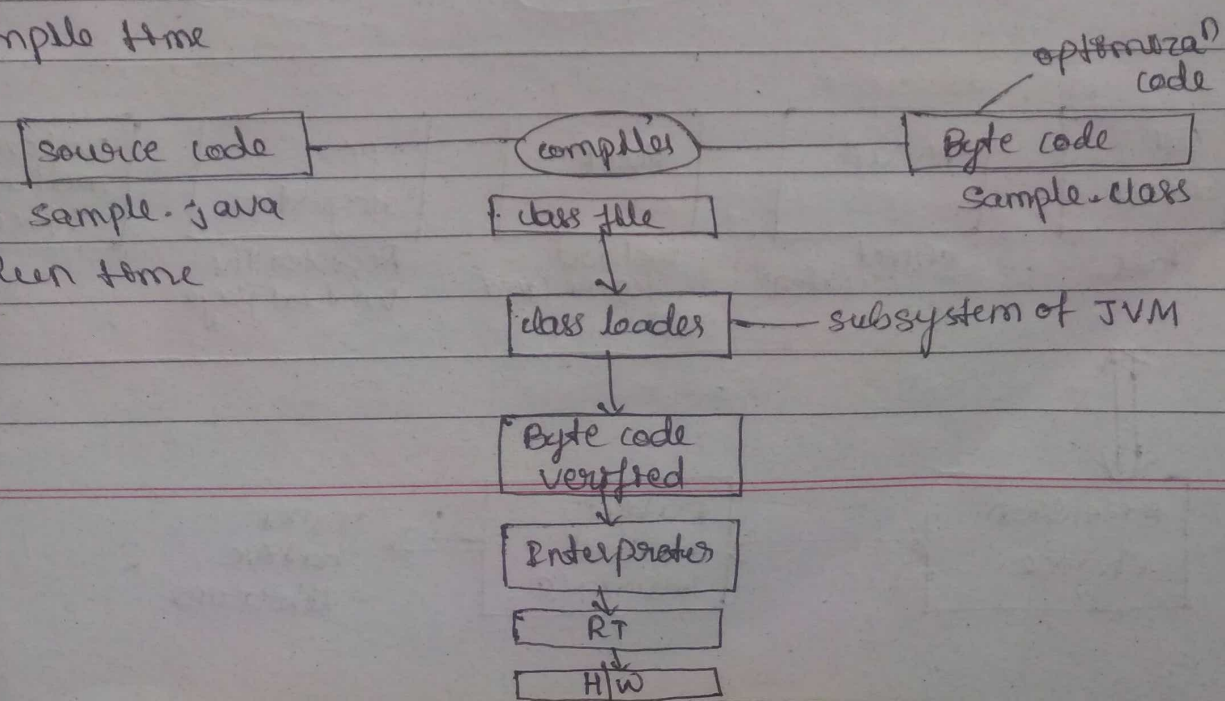
- Java SE
  structural
  (Java net sql)
- Java EE
  Enterprice
  (JDBC, servlet JSP)
  (Java Servlet page)
- Java ME
  → Dedicated to mobile application
- JFX
  Interface window based program
  (Application Swings)

## Internal working of program:

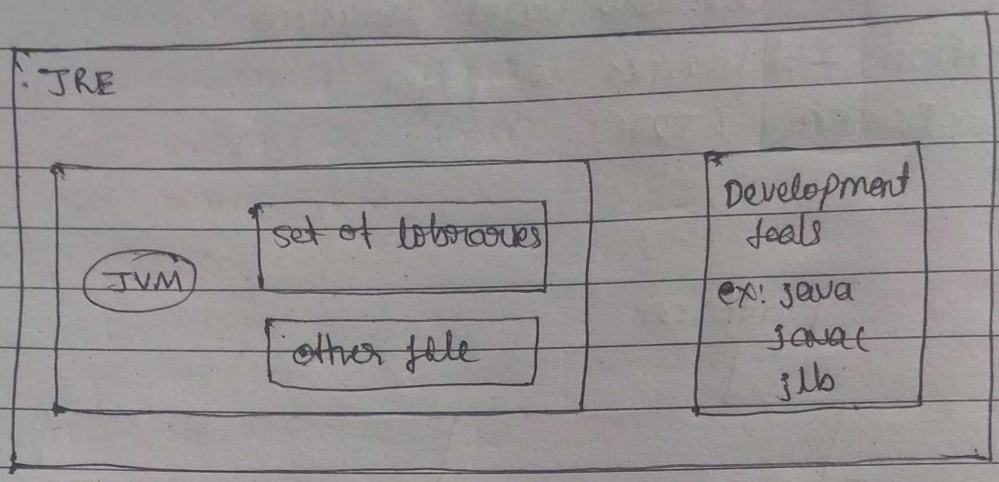
```
class sample{
    public static void main ( String args[])
        {
            System.out.println("Hi");
        }
}
```

CT - compile time

RT - Run time



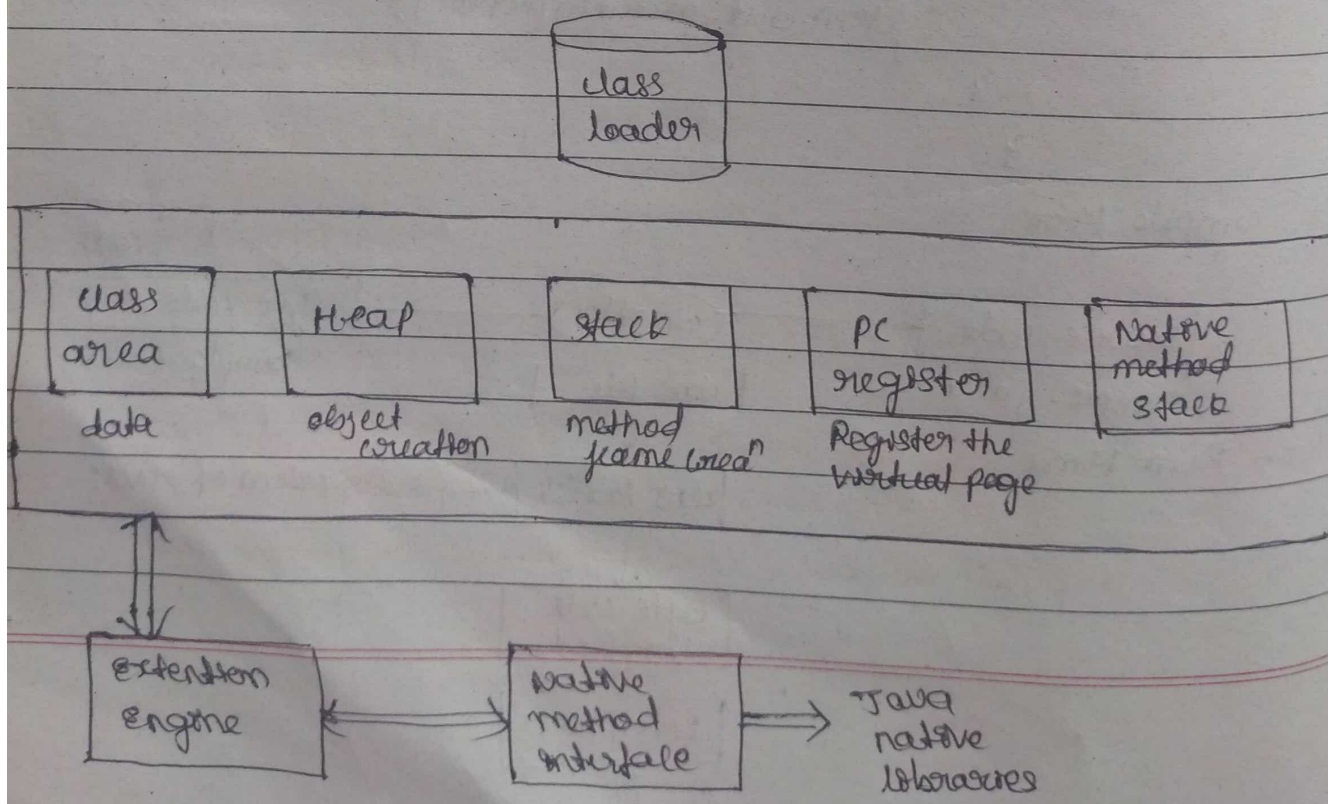
difference btw JDK, JRE, JVM




JDK = JRE + Development tools
JRE = set of libraries

loads the code
verifies the code
provide RT
execution code

Archetecture of JVM:



class loader

- Bootstrap class loader
- Extension class loader
- System | Application class loader

Extention engine

- virtual preprocessor
- Interpreter
- JIT — Just Intime compiler ↑ses the performance

21/11/23

String:

```
char[] ch = {'J','a','v','a'};
String s = new String(ch);

String s = "Java";
```

String implements
- Serializable
- comparable
- char sequence

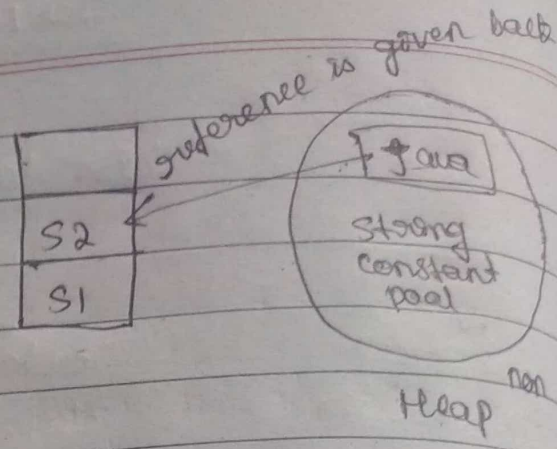
char sequence
- String
- String buffer
- String builder

String
- Immutable
  - String
- mutable
  - String buffer & builder

Creation of String object:

Creation of String
- ① By literals
- ② By new keyword

① class program {
    psvm() {
        String S1 = "java";
        String S2 = "java";
        sop(S1);
    }
}




② class program {
    psvm() {
        char[] ch = {'H', 'i'};
        String S1 = new String(ch);
        sop(S1);
    }
}

<u>Immutable String:</u>

classloader
threadsafe

psvm() {
    String s = "wel";
    s = s.concat("come");
    sop(s);
}

<u>output</u>: welcome

<u>String comparison</u>:
↳ equals() → method
↳ == → operator  checking the reference object
↳ compareTo() → method  - compare two strings in dictionary or lexicographically

equals() :

syntax:

```
                                class
public boolean equals(object another)  class
public boolean equalsIgnoreCase(String another)
```

```
char pgm{
  psvm(){
    String s1 = "java"                    sop(s1.equals(s2)); //T
    String s2 = "java";                   sop(s1.equals(s3)); //F
    String s3 = new String(java);         sop(s1.equals(s4)); //F
    String s4 = "JAVA";                   sop(s1.equalsIgnoreCase(s4)); //T
  }
}
```

ex2:

```
class program{
  psvm(){
    String s1 = "java";
    String s2 = "java";
    String s3 = new String(s2);
    sop(s1 = s2);   //T
    sop(s1 = s3);   //T
  }
}
```

22/11/2023

compareTo()

ex:

```
class program{
  psvm(){
    String s1 = "java";
    String s2 = "java";
    String s3 = "pgm";
    sop(s1.compareTo(s2));   //0
    sop(s1.compareTo(s3));   //1
    sop(s3.compareTo(s1));   //-1
  }
}
```

s1 = s2 : 0
s1 > s2  1
s1 < s2  -1

## String concatenation:

- +
- concat()

**Note:**
after a string literal all the + will be treated as string concatenation

### ① + operator

```
class program{
  psvm(){
    String s = "java"+ "pgm";
      sop(s);
  }
}
```

Imp

```
String s = (new StringBuilder()).append("java").append
                                  ("pgm").toString();
```

for mutable string

appending the values

to convert string o

Ex2:

```
psvm(){
    String s = 30+30+"java"+30+20;
    sop(s);
}
```

output: 60java3020

### ② concat()

```
class program{
  psvm(){
    String s1 = "java";
    String s2 = "program";
    sop(s1.concat(s2));
  }
}
```

other ways of string concatenation can be done using
① String builder class
② format method
③ join()
④ joiner()

↳ StringBuilders class applicable to mutable string

```
class pgm {
    psvm() {
        StringBuilders s1 = new StringBuilders("java");
        StringBuilders s2 = new StringBuilders("pgm");
        StringBuilders s3 = s1.append(s2);
            sop(s3.toString());
    }
}
```

② Format method

String.format()

```
class pgm {
    psvm() {
        String s1 = new String("java");
        String s2 = new String("pgm");
        String s3 = String.format("%s%s", s1, s2);
        sop(s3.toString());
    }
}
```

## ③ join() :

String . join()

```
class pegm1
  psvm(){
    String s1 = new String("java");
    String s2 = new String("pegm");
    String s3 = String.join("", s1 s2);
      sop(s3.toString());
  }
}
```

## ④ joiner()

String . joiner()

```
class pegm2
  psvm(){
    String.joiner s1 = new String.joiner(",");
              s1.add("wel");
              s1.add("come");
            sop(s1.toString());
    }
  }
```

23/11/2023

## String buffer & String builder : — predefined classes

↳ class like string
↳ mutable string

① StringBuffer()
② StringBuffer(int capacity)
③ StringBuffer(String s)

① StringBuffer() : Create empty string buffer with initial capacity of 16

② Create an empty string buffer with initial capacity as ~~the~~ a length.

③ Creation of ~~String~~ buffer with respected to specified string.

methods provided by string buffer

① append()
② replace()
③ insert()
④ delete()
⑤ reverse()
⑥ capacity()
⑦ ensure capacity()

### append():

```
class program
psvm() {
    StringBuffer s = new StringBuffer("java");
    ~~Stri~~ sop(s.append("java program"));
    }
}
```

output: java java program

2] Insert()

Syntax:

Insert (int offset, String s)

```
class program{
  psvm(){
    String buffer s = New StringBuffer("Hello");
    sop (s.Insert(1, "java");
  }
}
```

output: HjavaHello

3] replace()

Syntax: replace(int startIndex, int endIndex, String s)

```
class program{
  psvm(){
    String buffer s = New StringBuffer("Hello");
    sop( s. replace(1, H, "java");
  }
}
```

output: HJava0

If we give the index value more than the given string StringIndexOutofBound Exception occurs.

4] delete()

```
class program{
  psvm(){
  StringBuffer sb = new StringBuffer("Hello");
  sb. delete (1,2);
  System.out.println(sb);
}
}
```

output: llo

## 5) reverse()

```
class program{
  psvm(){
      StringBuffer sb = new StringBuffer("Hellow");
      sb.reverse();
      sop(sb)
  }
}
```

output: wollet

## 6) capacity()

```
class program{
  psvm(){
      StringBuffer sb=new StringBuffer();
      sop(sb.capacity());
      sb.append("Hello");
      sop(sb.capacity());
      sb.append("Java is my favorite language");
      sop(sb.capacity());
      sb.append("Java is my favorite language");
      sop(sb.capacity());
  }
}
```

output:

16
16
34
70

capacity() :

StringBuilder class provide a function capacity which return the current capacity of the buffer. the default value of default capacity value is 16 if no. of characters is more than current capacity it is used by ((old capacity value*2)+2)

7] ensure capacity function()

```
class program{
  psvm(){
    StringBuffer sb = new StringBuffer();
    sop(sb.capacity());                    // default 16
    sb.append("Hello");
    sop(sb.capacity());  // now 16
    sb.append("Java is my favourite language");
    sop(sb.capacity());  // now ((16*2)+2) = 34
    sb.ensureCapacity(10);
    sop(sb.capacity());  // now 34
    sb.ensureCapacity(50);  // now (34*2)+2
    sop(sb.capacity());  // now 70
  }
}
```

## String builder

### append():

1) class program{
    psvm(){
        StringBuilder s = New StringBuilder("Hello");
        sop(s.append("java");
    }
}

### 2) Insert():

class program{
    psvm(){
        StringBuilder s = new StringBuilder("Hello");
        sop(s.Insert(1,"java");
    }
}

### 3) replace():

class program{
    psvm(){
        StringBuilder s = new StringBuilder("Hello");
        sop(s.replace(1,4,"java");
    }
}

### 4) delete():

class program{
    psvm(){
        StringBuilder s = New StringBuilder("Hello");
        s.delete(1,2);
        sop(s);
    }
}

5) reverse():

```
class program
  psvm(){
    StringBuilder sb = new StringBuilder("Hello");
    sb.reverse();
    sop(sb);
  }
}
```

6) capacity()

```
class program
  psvm(){
    StringBuilder sb = new StringBuilder();
    sop(sb.capacity());
    sb.append("Hello");
    sop(sb.capacity());
    sb.append("java is my favorite language");
    sop(sb.capacity());
    sb.append("java is my favorite language");
    sop(sb.capacity());
  }
}
```

7) ensure capacity()

```
class program
  psvm(){
    StringBuilder sb = new StringBuilder();
    sop(sb.capacity());          // default 16
    sb.append("Hello");
    sop(sb.capacity());          // Now 16
    sb.append("java is my favorite language");
    sop(sb.capacity());          // Now (16*2)+2 = 34
    sb.ensureCapacity(10);
    sop(sb.capacity());          // Now 34
    sb.ensureCapacity(50);       // Now (34*2)+2
    sop(sb.capacity());          // Now 70
  }
}
```

String class provide a funⁿ where it converts

Value of () → primitive value into an respective string

```
public static String valueof (boolean b)
public static String valueof (int i)
                             (char[] ch)
                             (double d)
                             (float f)
                             (Long l)
                             (object o)
```

```
class program2
  psvm()
    boolean b1 = true;
    byte b2 = 11;
    short sh = 12;
    int i = 13;
    long l = 14L;
    float f = 15.5f;
    double d = 16.5d;
    char chr[] = {'j','a','v','a'};
    program1 obj = new program1();
    String s1 = String.valueof(b1);
    String s2 = String.valueof(b2);
    String s3 = String.valueof(sh);
    String s4 = String.valueof(i);
    String s5 = String.valueof(l);
    String s6 = String.valueof(f);
    String s7 = String.valueof(d);
    String s8 = String.valueof(chr);
    String s9 = String.valueof(obj);
    System.out.println(s1);
    :
    System.out.println(s9);
  }
}
```

output:

```
true
11
12
13
14
15.5
16.5
java
valueof.program1
@5e26
```

compiler internally override the string class

```
class student1{
    int rollno;
    string name;
    string city;
    student1 (int rollno, string name, string city){
      this.rollno = rollno;
      this.name = name;
      this.city = city;
    }
    public string to string(){ //overriding the to string() method
    return rollno+" "+name+" "+city;
    }
    public static void main (string args[]){
        student1 s1 = new student1 (101, "Raj", "lucknow");
        student s2 = new student1 (102, "vijay", "ghaziabad");
        sop(s1);   // compiler writes here s1.to string()
        sop(s2);   // compiler writes here s2.to string()
}
```

output:

101 Raj lucknow
102 vijay ghaziabad

28/11/23

toLowerCase()
toUpperCase()

```
public class Upper {
    psvm() {
        String s1 = " hello Java Program";
        String s1upper = s1.toUpperCase();
        sop( s1upper);
    }
}
```

output: HELLO JAVA PROGRAM

isEmpty()

syntax:

```
public boolean isEmpty();
```

```
public class empty {
    psvm() {
        String s1 = " ";
        String s2 = " java";
        sop (s1.isEmpty());
        sop (s2.isEmpty());
```

output:

true
false

## substring()

**Syntax**

```
public String subString(int startIndex)

public String subString(int startIndex, int endIndex)
```

```
public class sub {
    psvm() {
        String s1 = "JavaProgramm";
        String substr = s1.substring(0);
        System.out.println(substr);
        String substr2 = s1.substring(5,10);   // inclusion & exclusion
        SOP(substr2);
        String substr3 = s1.substring(5,15);
    }
}
```

**output:**

```
JavaProgramm
rogra
Error: index size out of bound exception.
```

## trim()

```
public class trim {
    psvm() {
        String s1 = " hello string ";
        SOP(s1+" javapoint");
        SOP(s1.trim() +" javatpoint");
    }
}
```

removes starting & end sp

**output:** hello string javatpoint

hello string javatpoint

trim() is used to eleminate leading spaces.
it checks unicode values before & after the string.

unicode: universal international standard character encoding

note: the trim() does not remove middle space

replace()

```
public class replace1{
    psvm(){
      String s1 = "Java programming language";
      String replaceString = s1.replace("Java", "python");
      sop(replaceString);
    }
}
```

output: Python programming language

startWith()

syntax:

```
public boolean startsWith(String Prefix);
```

checks the string with given prefix and returns true if string starts with give prefix or else returns false.

```
pc StartWith
psvm(){
  String s1 = "java Programm";
  sop(s1.startsWith("j"));
  sop(s1.startsWith("ja"));
  sop(s1.startsWith("String"));
}
}
```

1/12/2023

| String | String Buffer |
| --- | --- |
| * Immutable | * mutable |
| * low execun & more space | * fast execun & less space |
| * string constant pool | * Heap area |

| StringBuilder | StringBuffer |
| --- | --- |
| * less efficient | * more efficient |
| * synchronized i.e. thread safe | * non synchronized thread |

# RMI - Remote method Invocation

Remote obj

stub — client side gateway — outgoing request

skeleton — server side gateway — incoming request

establish the connec^n btw JVM

write the parameters

wait sometime

read the return value

steps:

- implementation for interface
- compile
  - classes & client create stub & skeleton object
- rmi ~~registry tool~~ registry tool
- start client server.

The naming class provides methods to get & store the rem

① public static java.rmi.Remote lookup(java.lang.String) throw
java.rmi.NotBoundException,
java.net.MalformedURLException,
java.rmi.RemoteException;

It return the reference of the remote object

② public static void bind(java.lang.String, java.rmi.Remote) throws.
java.rmi.AlreadyBoundException,
java.net.MalformedURLException,
java.rmi.RemoteException;

It binds the remote object with the given name.

③ public static void unbind(java.lang.String, java.rmi.Remote) throws.
java.rmi.RemoteException,
java.net.MalformedURLException;

It destroys the rem object which is bound with the given name

④ public static void rebind(java.lang.String, java.rmi.Remote) throws :
java.rmi.RemoteException,
java.net.MalformedURLException;

It binds the remo object to the new name

① import java.rmi.Remote;
public interface AddServerInt { extends Remote {
int add(int x, int y) throws RemoteException;

import java.rmi.*;
public class AddServerImpl extends UnicastRemoteObject
implement
public AddServerImpl() throws RemoteException {}
public int add