Core Java Programming

Presented By:

Corporate Trainer

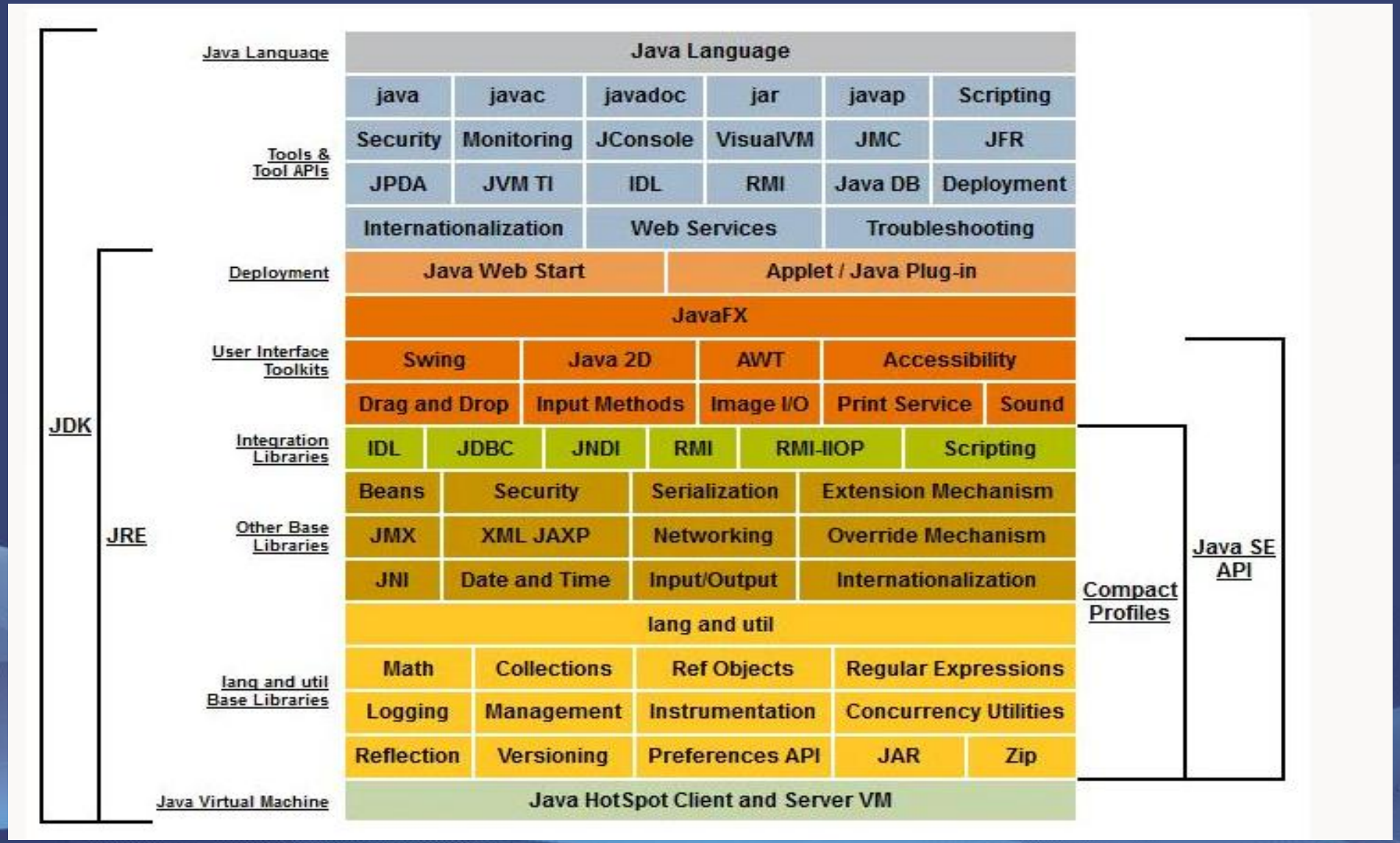
Mallikarjuna G D

gdmallikarjuna@gmail.com

CONTENTS

- ✓ JAVA STANDARD EDITION
- ✓ JAVA API HIERARCHY
- ✓ OOPS CONCEPTS
- ✓ CLASS, OBJECTS, CONSTRUCTORS
- ✓ INHERITANCE
- ✓ ACCESS MODIFIERS
- ✓ METHOD OVERLOADING
- ✓ METHOD OVERRIDING
- ✓ ARRAYS
- ✓ STRINGS

JDK STANDARD EDITION



Object-Oriented Programming:

It is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing various concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object:

An entity that has state and behavior is known as an object.

It can be physical or logical. Example: chair, bike, marker, pen, table, car etc.

- An object has three characteristics:
  - State: It represents data (value) of an object.
  - Behavior: It represents the behavior (functionality) of an object such as deposit, withdraw etc.
  - Identity : It gives a unique name to an object and enables one object to interact with other objects.

Class:

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

A class contains:

- fields
- methods
- constructors
- blocks
- nested class and interface

Syntax:

```
class <class_name>
{
    field;
    method;
}
```

Constructor:

It is a special method having same name of class having no return type, used to initialize the instance members and the object.

Types of constructor:

- Default constructor -

  A constructor that have no parameter is known as default constructor.

  syntax: <class_name> () {}

- Parameterized constructor - A constructor that have parameters is known as parameterized constructor.

  syntax: <class_name>(parameters) {}

Inheritance:

It is a mechanism in which one object acquires all the properties and behaviors of parent object.

Inheritance represents the IS-A relationship, also known as parent-child relationship.

Why inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

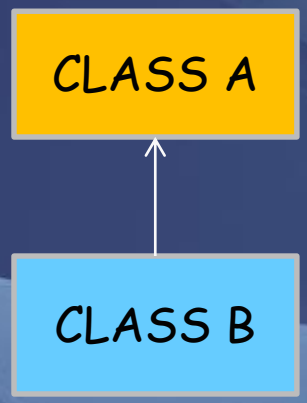
Syntax:

```
Class  Superclass-name
{
   //methods and fields
}
```

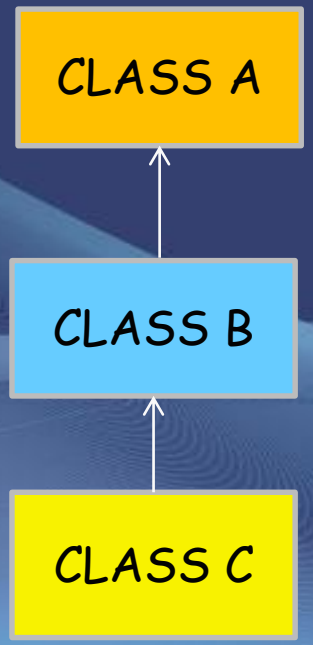
```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

Types of Inheritance:

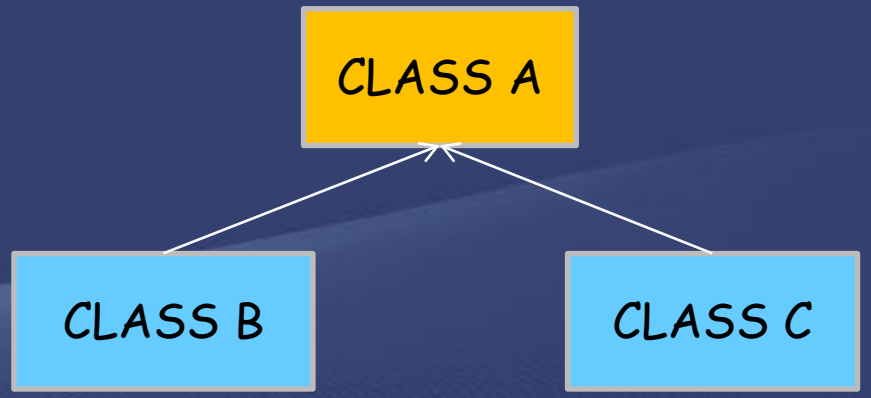
- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance



Single



Multilevel



Hierarchical

Example for Single Inheritance:

```
class Animal {
Void  eat() {
System.out.println("eating..."); }
}
class Dog extends Animal {
void bark() {
System.out.println("barking..."); }
}
class TestInheritance {
public static void main(String args[]) {
Dog d=new Dog();
d.bark();
d.eat();
}} /* O/P -   barking...
                eating...   */
```

Example for Multilevel and Hierarchical Inheritance:

```
class Animal {
void eat() {
System.out.println("eating...");
} }
class Dog extends Animal {
void bark() {
System.out.println("barking...");
} }
class BabyDog extends Dog {
void weep() {
System.out.println("weeping...");
} }
class TestInheritance2 {
public static void main(String args[]) {
BabyDog d=new BabyDog();
d.weep();
d.eat(); } } /* O/P - weeping...
                      eating... */
```

```
class Animal {
void eat() {
System.out.println("eating...");
} }
class Dog extends Animal{
void bark() {
System.out.println("barking...");
} }
class Cat extends Animal{
void meow() {
System.out.println("meowing...");
} }
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat(); } } /* O/P - meowing...
                      eating... */
```

Encapsulation :

Encapsulation in java is a process of wrapping data and the methods together into a single unit.

The Java Bean class is the example of fully encapsulated class.

Example:

```
//save as Student.java

package com.pack;
public class Student {
private String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name=name
} }
```

```
//save as Test.java

package com.pack1;
class Test{
public static void main(String[] args)
{
Student s=new Student();
s.setName("vijay");
System.out.println(s.getName());
}
}      // O/P - vijay
```

Access Modifiers:

It specifies accessibility (scope) of a data member, method, constructor or class.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

There are 4 types of access modifiers:

- Private : Accessible only within class.
- Default : Accessible only within package.
- Protected : Accessible within package and outside the package but through inheritance only.
- Public : Accessible everywhere. It has the widest scope among all other modifiers.

Polymorphism:

It is a concept by which we can perform a single action by different ways.

- Types of Polymorphism:
  - Compile time polymorphism
  - Runtime polymorphism.

- It perform in java by method overloading and method overriding.

Method Overloading:

A class has multiple methods having same name but different in parameters. It increases the readability of the program

Example:

```
class Adder {
static int add(int a,int b) {
return a+b; }
static int add(int a,int b,int c) {
return a+b+c; }
}
class TestOverloading1 {
public static void main(String[] args) {
System.out.println(Adder.add(11,11));  // O/P - 22
System.out.println(Adder.add(11,11,11));  // O/P - 33
}}
```

Method Overriding:

Subclass provides the specific implementation of the method that has been provided by one of its parent class.

Example:

```
class Vehicle {
Void run() {
System.out.println("Vehicle is running"); }
}
class Bike2 extends Vehicle {
void run() {
System.out.println("Bike is running safely"); }
public static void main(String args[]) {
Vehicle obj = new Bike2();
obj.run();
}                    // O/P – Bike is running safely
```

Abstraction:

It is the process of hiding certain details and only show essential features of the object.

- Abstraction in Java is achieved by using abstract class and interface.
- Abstract class in Java:

A class that is declared as abstract is known as abstract class.

It needs to be extended and its method implemented. It cannot be instantiated.

Example: abstract class A{}

Example for Abstract class:

```
abstract class Bank {
abstract int getRateOfInterest();
}
class SBI extends Bank {
int getRateOfInterest() {
return 9; }
}
class PNB extends Bank{
int getRateOfInterest() {
return 10; }
}
class TestBank {
public static void main(String args[]) {
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}  /* O/P - Rate of Interest is: 9 %
                Rate of Interest is: 10 %  */
```

Interface:

It is an abstract type that is used to specify a behaviour that classes must implement.

Interfaces are declared using the interface keyword, and may only contain method signature and constant declarations. All methods of an Interface do not contain implementation (method bodies).

Advantages of interface:

- Used to achieve abstraction
- Support the functionality of multiple inheritance.

Example for Interface:

```
interface printable{
void print();
}
class A1 implements printable {
public void print() {
System.out.println("Hello"); }
public static void main(String args[])
{
A1 obj = new A1();
obj.print();
 } }  // O/P - Hello
```

Arrays:

- It is an indexed collection of fixed number of homogeneous data elements. Once created with some size, we can't alter the size.
- It can store both primitive data and objects.
- Represent multiple values with a single variable. So that reusability of the code will be improved.
- A specific element in an array is accessed by its index.
- We can add elements into array and can't insert an element into specified index of an array.
- It may have one or more dimensions.

Single dimension arrays:

Syntax : type[] var-name; or
type var-name[]; or
type []var-name;

Example : int[] a = new int[2];
a[0]= 10;
a[1]= 20;
int a[] = {10, 20};
int []a = new int[]{10,20};

Program: Average an array of values:

```
class Average
{
public static void main(String args[]) {
  double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
  double result = 0;
  int i;
  for(i=0; i<5; i++)
  result = result + nums[i];
  System.out.println("Average is " + result / 5);
}
}   // O/P – Average is 12.29..
```

Two-dimensional arrays:

- These are arrays of arrays.
- To declare a two dimensional array variable, specify each additional index using another set of square brackets.

Syntax : type[][] var-name; or
type var-name[][];

Example : int[][] a = new int[2][3];
int a[][] = {{10, 20},{30,40}};

Program: 2D Print

```
Class twodimensional
{
public static void main(String args[]) {
int arr[][] = {{1,2,3},{4,5,6},{7,8,9}};
    for (int i=0; i< 3 ; i++) {
      for (int j=0; j < 3 ; j++)
    System.out.print(arr[i][j] + " ");
      System.out.println();
       }
   }
}
/* O/P -  1  2 3
          4  5  6
          7  8  9    */
```

String:

- It is an object that represents a sequence of characters.
- The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes.
- Objects of String are immutable and fixed in length.
- The java.lang.String class is used to create string object.
- It uses to make Java more memory efficient.
- It is used to have constants in program.

Immutable class:

- Once the immutable class created we cannot change the value.
- String is immutable class due to this we cannot change the content .
- How to create Immutable class.
  - Declare the class as final (so it cannot be extended)
  - All Class member should be private so that it cannot be accessed from outside
  - Should not contain any setter methods to change the value of class members
  - The getter method should return the copy of the members
  - The class members only initialized through constructor

Immutable class:

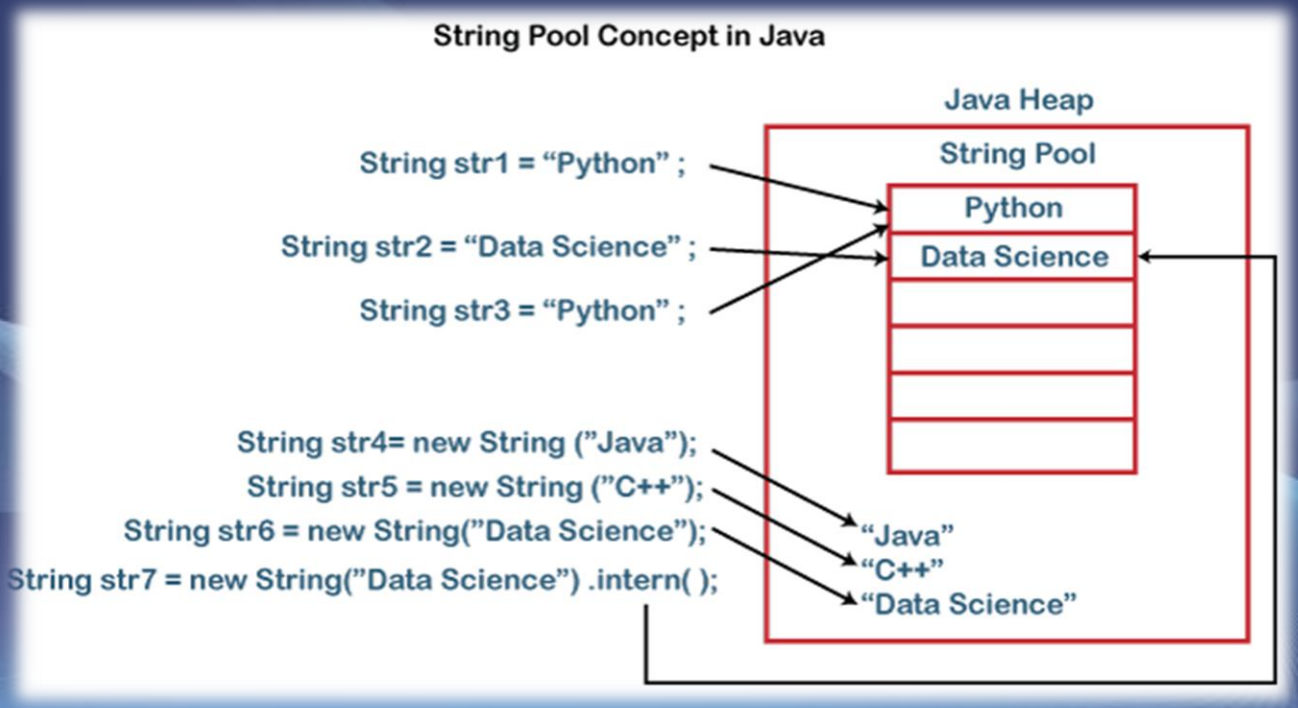
```
final class Immutable {

private string name;
        Immutable(String name){
        this.name = name;
        }

       public String getName(){
       return name;
       }
}
```

Advantages Immutable class:

- Immutable class remains in exactly one state . Object is thread safe and no synchronization issue
- Easir to design and implement
- Immutable objects good to use for Map keys and Set elements
- Internal state is consistent even you have exceptions
- It can be cached because internally never going to change

StringPool:

- It is storage area in heap memory which stores the string literals.
- This is also called String Intern Pool.




```
String s1 = "HELLO";
       String s2 = "HELLO";
       String s3 =  new String("HELLO");

       System.out.println(s1 == s2); // true
       System.out.println(s1 == s3); // false
       System.out.println(s1.equals(s2)); // true
       System.out.println(s1.equals(s3)); // true
```

Accessor methods: length(), charAt(i), getBytes(), getChars(istart,iend,gtarget[],itargstart), toCharArray(), valueOf (g,iradix), substring(iStart [,iEndIndex)]) [returns up to ***but not including*** iEndIndex]

Modifier methods: concat(g), replace(cWhich, cReplacement), toLowerCase(), toUpperCase(), trim(). The method format(gSyn,g) uses c-like printf syntax for fixed fields if required in reports.

Boolean test methods: contentEquals(g), endsWith(g), equals(g), equalsIgnoreCase(g), matches(g), regionMatches (i1,g2,i3,i4), regionMatches(bIgnoreCase,i1,g2,i3,i4), startsWith(g)

Integer test methods: compareTo(g) [returns 0 if object equals parameter, -1 if object is before parameter in sort order, +1 if otherwise], indexOf(g) [returns position of first occurrence of substring g in the string, -1 if not found], lastIndexOf(g) [returns position of last occurrence of substring g in the string, -1 if not found], length().

Two ways to create String object: (With 'new' or without 'new')

1. By string literal

   Syntax: String s="Welcome";

2. By new keyword

   String s=**new** String("Welcome");

Example:

```
public class StringExample {
public static void main(String args[]) {
String s1="java"; //creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch); //converting char array to string
String s3=new String("example"); //creating java string by new keyword
System.out.println(s1);  // O/P - java
System.out.println(s2);  // O/P - strings
System.out.println(s3);  // O/P - example
}}
```

Methods of String:

Modifier and Type	Method	Modifier and Type	Method
public char	charAt(int index)	public boolean	contains(CharSequence s)
public int	length()	public boolean	equals(Object another)
public int	indexOf(int ch)	public String	substring(int beginIndex)
public int	indexOf(int ch, int fromIndex)	public String	substring(int beginIndex, endIndex)
public int	indexOf(String substring)	public String	concat(String str)
public int	indexOf(String substring, int fromIndex)	public String	replace(char old, char new)

Methods of String: (Continue..)

Modifier and Type	Method	Modifier and Type	Method
public String	replace(CharSequence old, CharSequence new)	public String	trim()
public String	intern()	public String[]	split(String regex)
public String	toLowerCase()	public String[]	split(String regex, int limit)
public String	toLowerCase(Locale l)	static String	format(String format, Object... args)
public String	toUpperCase()	static String	format(Locale l, String format, Object... args)
public String	toUpperCase(Locale l)	static String	valueOf(int value)

Example for methods in String:

```
public class Test {
public static void main(String[] args) {
      String s= "Snipe IT Solutions";
        System.out.println("length = " + s.length());     // O/P – length = 18
        System.out.println("At 3rd position = " + s.charAt(3)); // O/P - At 3rd position = p
        System.out.println("Substring  = " + s.substring(3,5)); // O/P - Substring  = pe
        String s1 = "Snipe";
        String s2 = "Tutorials";
        System.out.println("Concatenated string  = " + s1.concat(s2)); /* O/P - Concatenated string  =
                                                                          Snipe Tutorials */
        System.out.println("Index of a  = " + s.indexOf('s',1)); // O/P - Index of a  = 17
        int out1 = s2.compareTo(s1);
        System.out.println("If s1 = s2 " + out1);  // O/P - If s1 = s2 1
        System.out.println("Changing to upper Case = " + s1.toUpperCase()); /* O/P - Changing to upper
                                                                              Case =  SNIPE */
        String s3 = "SNYPE".replace('Y' ,'I') ;
        System.out.println("Replaced Y with I -> " + s3); //  O/P - Replaced Y with I ->  SNIPE
}}
```

StringBuffer:

- StringBuffer class objects allow manipulation of strings without creating a new object each time manipulation occurs.

Accessor methods: capacity(), charAt(i), length(), substring(iStart [,iEndIndex])

Modifier methods: append(g), delete(i1, i2), deleteCharAt(i), ensureCapacity(), getChars(srcBeg, srcEnd, target[], targetBeg), insert(iPosn, g), replace(i1,i2,gvalue), reverse(), setCharAt(iposn, c), setLength(),toString(g)

StringBuffer:

- Objects of String are mutable and not fixed in length and should be created only with "new" operator.
- Every method present in StringBuffer is synchronized.
- Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.
- It increases waiting time of threads. Hence relatively performance is low.
- StringBuffer defines three constructors:

  - StringBuffer( ) Ex: StringBuffer s = new StringBuffer();
  - StringBuffer(int size) Ex: StringBuffer s = new StringBuffer(10);
  - StringBuffer(String str) Ex: StringBuffer s = new StringBuffer("SNIPE");

Important Methods of StringBuffer class:

Modifier and Type	Method	Modifier and Type	Method
public synchronized StringBuffer	append(String s)	public int	capacity()
public synchronized StringBuffer	insert(int offset, String s)	public char	charAt(int index)
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	public int	length()
public synchronized StringBuffer	delete(int startIndex, int endIndex)	public String	substring(int beginIndex)
public synchronized StringBuffer	reverse()	public String	substring(int beginIndex, int endIndex)

Example for methods in StringBuffer class:

```
public class Test {
    public static void main(String[] args) {
        StringBuffer sb=new StringBuffer("SNIPE ");
        System.out.println(sb.append("Solutions")); // O/P – SNIPE Solutions
        System.out.println(sb.insert(5," IT")); // O/P – SNIPE IT Solutions
        System.out.println(sb.replace(10,11,"a")); // O/P - SNIPE IT Salutions
        System.out.println(sb.delete(17,18)); // O/P - SNIPE IT Salution
        System.out.println(sb.charAt(4)); // O/P - E
        System.out.println(sb.substring(6)); // O/P - IT Salution
        System.out.println(sb.substring(6, 8)); // O/P - IT
        System.out.println(sb.reverse()); // O/P - noitulaS TI EPINS
        System.out.println(sb.length()); // O/P - 17
        System.out.println(sb.capacity()); // O/P - 22
    }
}
```

StringBuilder:

- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized and is not thread-safe
- StringBuilder is more efficient than StringBuffer.
- Threads are not required to wait to operate in StringBuilder object. Hence relatively performance is high.
- StringBuilder defines three constructors:

• StringBuilder( )	Ex: StringBuilder s = new StringBuilder();
• StringBuilder(int length)	Ex: StringBuilder s = new StringBuilder(10);
• StringBuilder(String str)	Ex: StringBuilder s = new StringBuilder("SNIPE");

Important Methods of StringBuilder class:

Modifier and Type	Method	Modifier and Type	Method
public StringBuilder	append(String s)	public int	capacity()
public StringBuilder	insert(int offset, String s)	public char	charAt(int index)
public StringBuilder	replace(int startIndex, int endIndex, String str)	public int	length()
public StringBuilder	delete(int startIndex, int endIndex)	public String	substring(int beginIndex)
public StringBuilder	reverse()	public String	substring(int beginIndex, int endIndex)

Example for methods in StringBuilder class:

```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb=new StringBuilder("SNIPE ");
        System.out.println(sb.append("Solutions")); // O/P – SNIPE Solutions
        System.out.println(sb.insert(5," IT")); // O/P – SNIPE IT Solutions
        System.out.println(sb.replace(10,11,"a")); // O/P - SNIPE IT Salutions
        System.out.println(sb.delete(17,18)); // O/P - SNIPE IT Salution
        System.out.println(sb.charAt(4)); // O/P - E
        System.out.println(sb.substring(6)); // O/P - IT Salution
        System.out.println(sb.substring(6, 8)); // O/P - IT
        System.out.println(sb.reverse()); // O/P - noitulaS TI EPINS
        System.out.println(sb.length()); // O/P - 17
        System.out.println(sb.capacity()); // O/P - 22
}}
```

String vs StringBuffer vs StringBuilder in Java

Example:

```
class Test {
      public static void concat1(String s1)          {
            s1 = s1 + "IT Solutions";       }
      public static void concat2(StringBuffer s2) {
            s2.append(" IT Solutions ");  }
      public static void concat3(StringBuilder s3) {
            s3.append(" IT Solutions ");  }
      public static void main(String[] args)         {
            String s1 = "SNIPE";
      concat1(s1); // s1 is not changed
System.out.println("String: " + s1);   // O/P – String: SNIPE
             StringBuffer s2 = new StringBuffer("SNIPE ");
      concat2(s2); // s2 is changed
System.out.println("StringBuffer: " + s2); // O/P – StrigBuffer: SNIPE IT Solutions
            StringBuilder s3 = new StringBuilder("SNIPE ");
      concat3(s3); // s3 is changed
System.out.println("StringBuilder: " + s3); // O/P – StrigBuilder: SNIPE IT Solutions
}}
```

THANK YOU