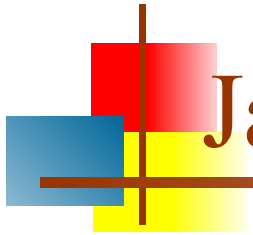
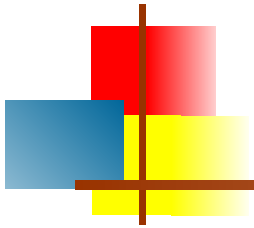


Java Programming- Chapter1-Part B

Basics of Java Programming



Methods

- A method is a block of code which only runs when it is called and must be declared inside a class
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.
- Create a Method
- It is defined with the name of the method, followed by parentheses ().
- Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Example

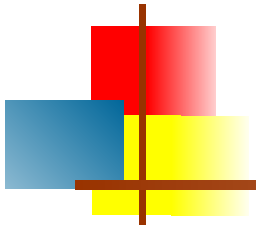
```
//create a method
public class MyClass
{
   void myMethod()
    {
    // code to be executed
    }
}
```

Call a Method

1. To call a method in Java, write the method's name followed by two parentheses () and a semicolon;
2. In the following example, myMethod() is used to print a text (the action), when it is called:

Example

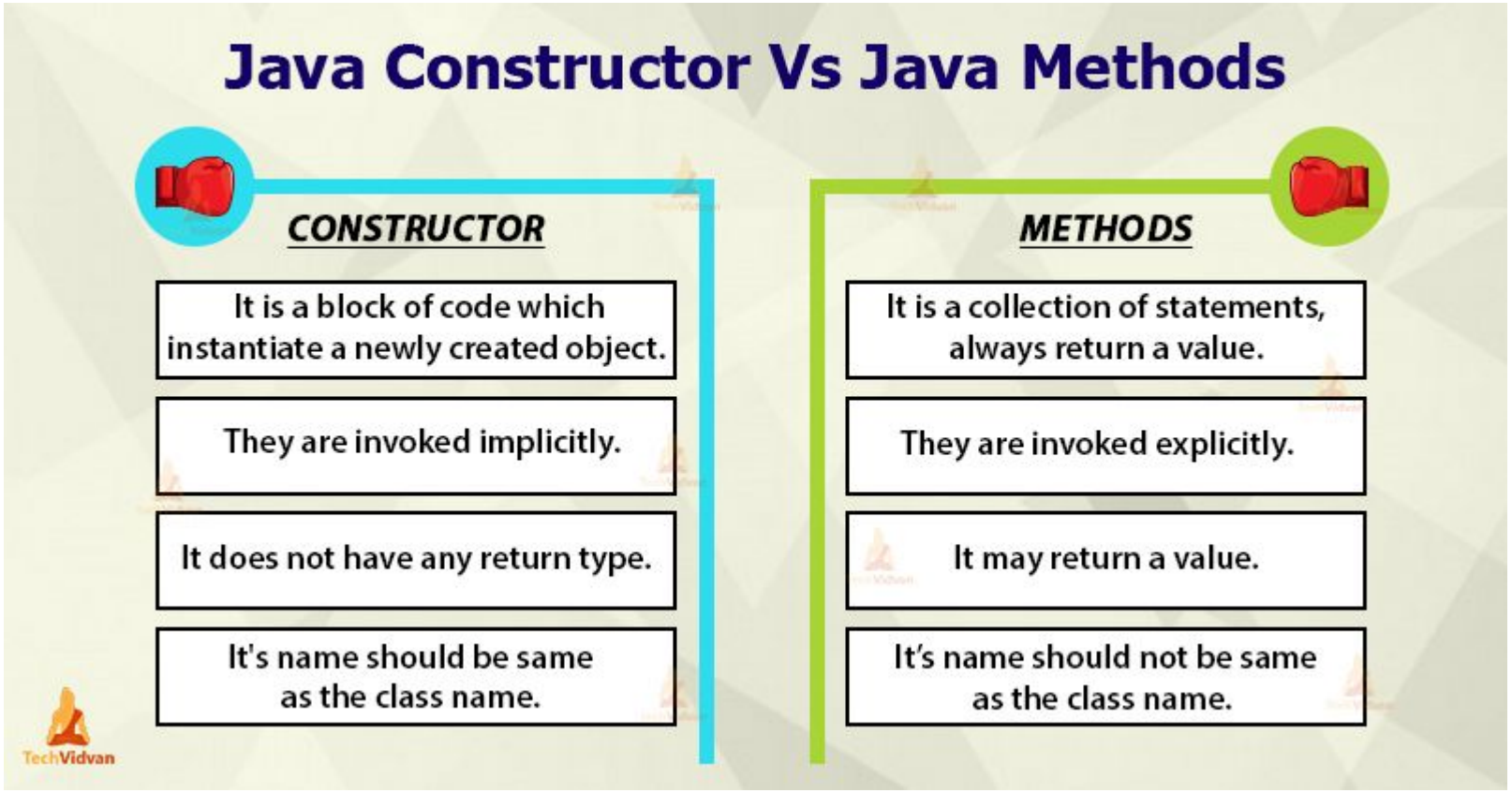
```
public class MyClass
 {
   void myMethod()
 {
 System.out.println("I just got executed!");
  }
  public static void main(String[] args)
 {
   MyClass m1=new MyClass();
   m1.myMethod();
  }
}
```



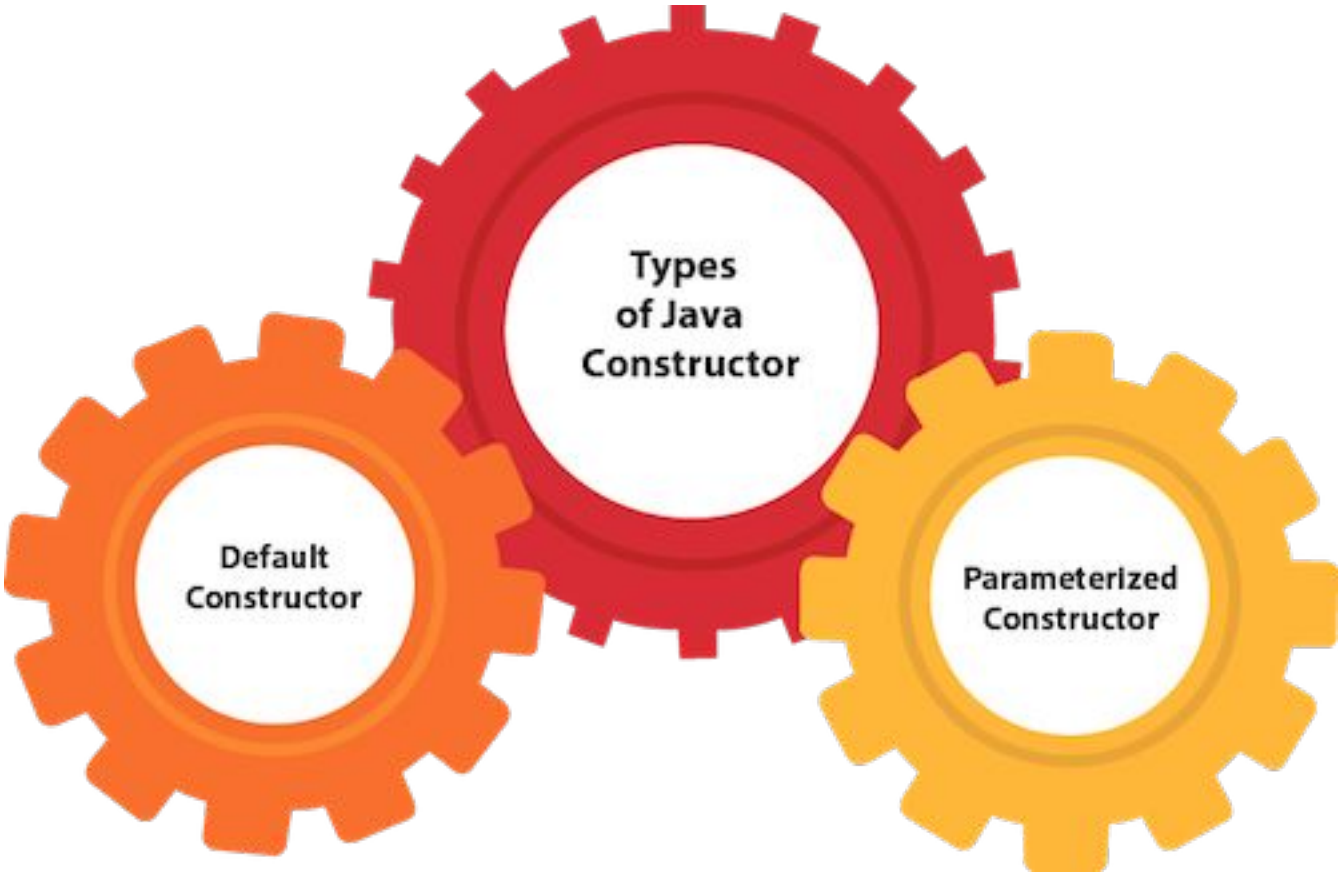
Constructors

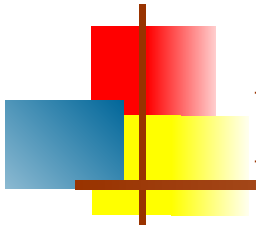
- Special member functions
- Whose name is same as class name
- Used to construct values of object : so name is constructor
- No return type
- Can not be declared as static, final
- No explicit return type
- Example book{} is class then book() is constructor of that class

Difference between method & constructor



Types of Constructor





Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.

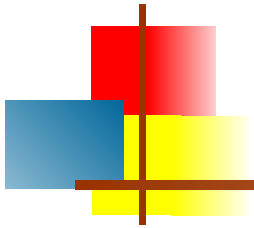
```
class Bike{
//creating a default constructor
Bike()
{
System.out.println("Bike is purchased");
}
public static void main(String args[])
{
//calling a default constructor
Bike b=new Bike();
}
```

Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects.

```
Public class Student{
   int id;
   String name;
   //creating a parameterized constructor
 Student(int i,String n)
    {
     id = i;
     name = n;
    }
   //method to display the values
   void display()
  {
   System.out.println(id+" "+name);
  }
```

```
public static void main(String args[])
{
   //creating objects and passing values
Student s1 = new Student(111,“Sita");
Student s2 = new Student(2,“Ramesh");
  //calling method to display the values of /
//object
   s1.display();
   s2.display();
  }
}
```



Parameterized Constructor

- Actual parameters are the variables declared in side class
- Formal parameters are the variables declared inside () brackets of constructor
- While actually writing constructor we must wrute it as actual_parameter_name=formal_parameter_name
- In above example

  actual=id& name

  formal=I & n
- So inside constructor id=I
- Program for addition of two complex numbers

Overloading of Constructor

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

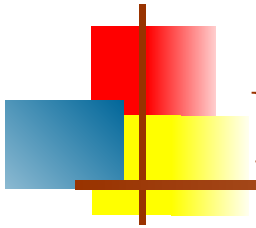
```
class Student1{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student1(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student1(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
```

```
void display()
{
System.out.println(id+" "+name+" "+age);
}

    public static void main(String args[])
{
    Student1 s1 = new Student1(111,“Ram");
    Student1 s2 = new Student1(222,“Sita",15);
    s1.display();
    s2.display();
   }
}
```

Parameter passing using ‘this’

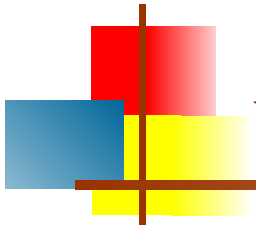
- ‘this’ is a reference variable that refers to the current object.

```
class Test
{
    int a;
    int b;

    // Parameterized constructor
    Test(int a, int b)
    {
        this.a = a;//pointing to current object
        this.b = b;
    }

    void display()
    {
        //Displaying value of variables a and b
        System.out.println("a = " + a + "  b = " + b);
    }

    public static void main(String[] args)
    {
        Test t1 = new Test(10, 20);
        t1.display();
    }
}
```



Nesting of Methods

```
class MyClass
{
   method1(){

      // statements
   }

   method2()
   {
     // statements

    // calling method1() from method2()
     method1();
   }
   method3()
   {
    // statements

    // calling of method2() from method3()
    method2();
   }
}
```



Variable Arguments (Varargs) in Java

- In JDK 5, Java has included a feature that simplifies the creation of methods that need to take a variable number of arguments.
- This feature is called varargs and it is short-form for variable-length arguments.
- A method that takes a variable number of arguments is a varargs method.
- The varargs feature offers a simpler, better option.
- Syntax of varargs :

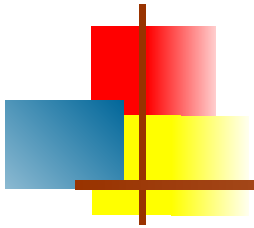
A variable-length argument is specified by three periods(…).

For Example,

```
public static void fun(int ... a)
{
   // method body
}
```

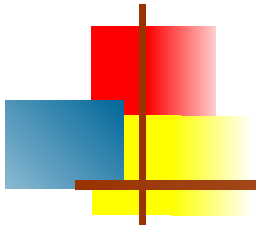
Program on varargs

```
class Test1
{
    // A method that takes variable number of integer
    // arguments.
    static void fun(int ...a)
    {
        System.out.println("Number of arguments: " + a.length);

        // using for each loop to display contents of a
        for (int i: a)
            System.out.print(i + " ");
        System.out.println();
    }

    // Driver code
    public static void main(String args[])
    {
        // Calling the varargs method with different number
        // of parameters
        fun(100);// one parameter, method is static so called without object
        fun(1, 2, 3, 4);  // four parameters
        fun();            // no parameter
    }
}
Output:
Number of arguments: 1 100
Number of arguments: 4 1 2 3 4
Number of arguments: 0
```



Java Garbage Collection

- In java, garbage means unreferenced/unused objects.
- Garbage Collection is process of releasing the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects.
- in java it is performed automatically. So, java provides better memory management.
- Advantage of Garbage Collection
  - It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
  - It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.



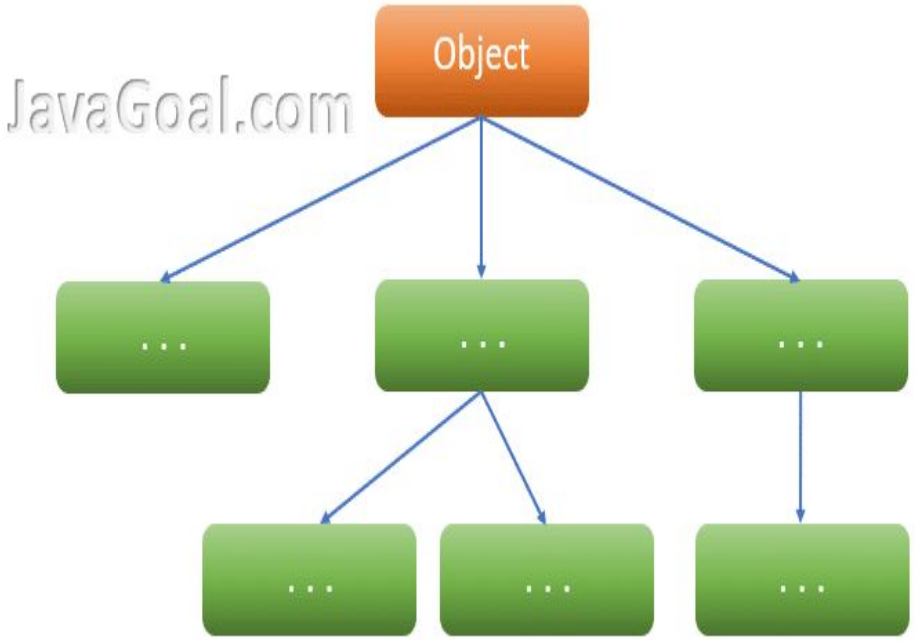
finalize() method

- The finalize() method is invoked each time before the object is garbage collected.
- This method can be used to perform cleanup processing. This method is defined in Object class as:

  protected void finalize(){}
- Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).
- gc() method
  - The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.
  - public static void gc(){}

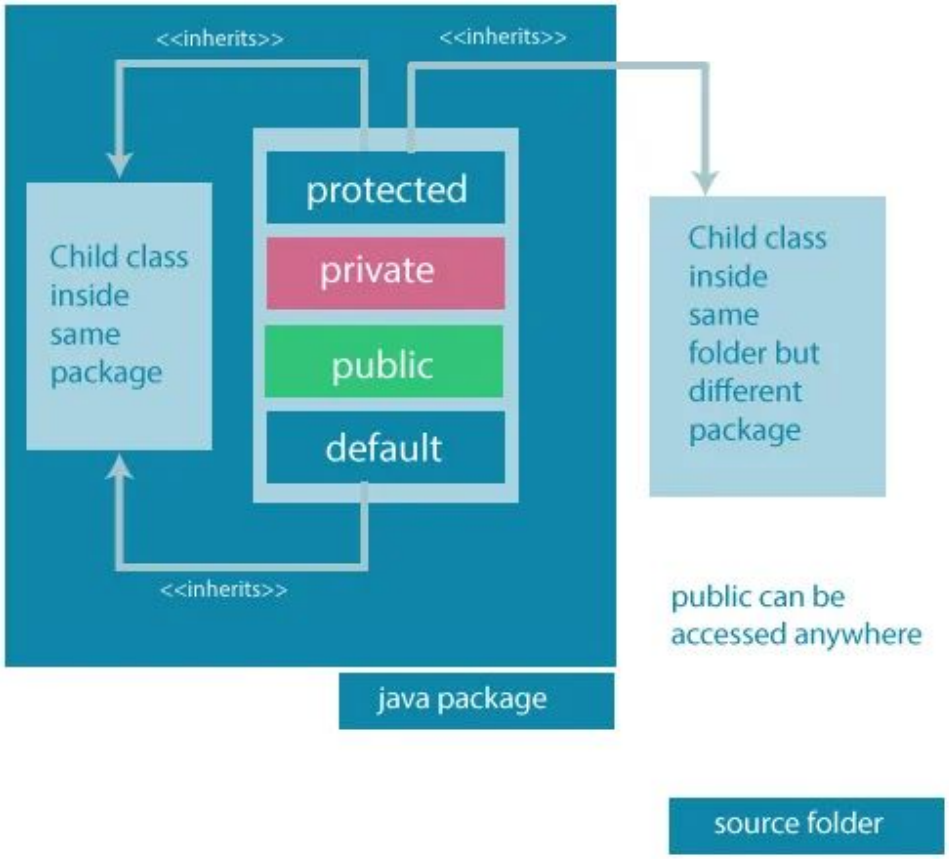
Object class

- In Java, Object is the super most class that is present in java.lang package.
- It means every class in java is a subclass, which inherited the properties from Object class(Super most).
- Every class in Java is directly or indirectly inheriting the properties from the Object class.



- If you want to refer any object whose type is unknown then the Object class is beneficial.
- Because Object is parent class so that you can refer to the child class object.
- Object obj1=new Aclass();

Visibility Control or Access Specifiers



Visibility Control or Access Specifiers

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Array

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- Array is collection of similar data types
- Train bogies is real-time example of array & chairs arranged in a line
- To declare an array, define the variable type with square brackets:
- Datatype[] array_name;
  Example String[] cars;
- To insert values inside array

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

- To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

- You access an array element by referring to the index number.

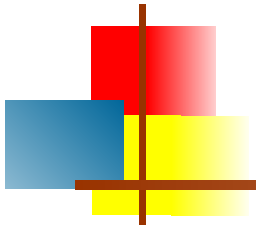
```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Array…

- You can use for loop to access all array elements.
- use the length property to specify how many times the loop should run.
- The following example outputs all elements in the cars array:
- Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++)
{
 System.out.println(cars[i]);
}
```

Cars.length gives length of array. Number of elements in the array.



String & String class

- Strings in Java are Objects that are packed internally by a char array.
- Since arrays are immutable(cannot grow), Strings are immutable as well.
- Whenever a change to a String is made, an entirely new String is created.
- String Object Declaration:
- String is a readymade class available in Java(so starting S has to be capital)
- There are two ways to create objects

  1. String Object is created as a literal, the object will be created in String constant pool. This allows JVM to optimize the initialization of String literal.

  For example: String str = “MIT";

  2. The string can also be declared using new operator i.e. dynamically allocated. In case of String are dynamically allocated they are assigned a new memory location in heap. This string will not be added to String constant pool.

  For example: String str = new String(“MIT");

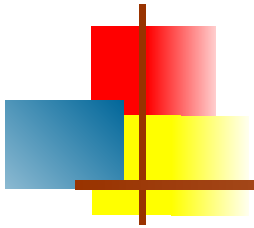
String class methods

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	String substring(int beginIndex)	returns substring for given begin index.
4	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
5	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
6	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
7	boolean equals(Object another)	checks the equality of string with the given object.
8	boolean isEmpty()	checks if string is empty.
9	String concat(String str)	concatenates the specified string.
10	String replace(char old, char new)	replaces all occurrences of the specified char value.
11	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
12	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
13	String[] split(String regex)	returns a split string matching regex.
14	String[] split(String regex, int limit)	returns a split string matching regex and limit.
15	int indexOf(int ch)	returns the specified char value index.
16	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
17	String toLowerCase()	returns a string in lowercase.
18	String toUpperCase()	returns a string in uppercase.
19	String trim()	removes beginning and ending spaces of this string.
20	static String valueOf(int value)	converts given type into string. It is an overloaded method.

Java code to illustrate different constructors and methods String class.

```
class Test
{
        public static void main (String[] args)
        {
                String s;
Scanner sc=new Scanner(System.in);
S=sc.next();
String s1=”abc”;
Boolean out=s.equals(s1);
If(out==true)
System.out.println(“correct password”);
Else
System.out.println(“incorrect”);
// Returns the number of characters in the String.
System.out.println("String length = " + s.length());
// Returns the character at ith index.
Char c=s.charAt(3);
System.out.println(“char is=” +c);
System.out.println("Character at 3rd position = " +
s.charAt(3));
// Return the substring from the ith index character
// to end of string
System.out.println("Substring " + s.substring(3));
// Returns the substring from i to j-1 index.
System.out.println("Substring = " + s.substring(2,5));
```

```
String s1 = "MIT";
        String s2 = "Polytechnic";
System.out.println("Concatenated string = " +
                                                s1.concat(s2));
                // Returns the index within the string
                // of the first occurrence of the specified string.
                String s4 = "Learn Share Learn";
                System.out.println("Index of Share " +
                                                s4.indexOf("Share"));
                // Returns the index within the string of the
                // first occurrence of the specified string,
                // starting at the specified index.
                System.out.println("Index of a = " +
                                                s4.indexOf('a',3));
                // Checking equality of Strings
                Boolean out = "MIT".equals("Mit");System.out.println("Checking
Equality " + out);
                out = "MIT".equals("MIT");
System.out.println("Checking Equality " + out);
                out = "MIT".equalsIgnoreCase("Mit ");
System.out.println("Checking Equality " + out);

//If ASCII difference is zero then the two strings are similar
                int out1 = s1.compareTo(s2);
System.out.println("the difference between ASCII value is="+out1);
                // Converting cases
                String word1 = "MIT Polytechnic, PUne";
System.out.println("Changing to lower Case " +word1.toLowerCase());
                // Converting cases
                String word2 = "welcome!";
System.out.println("Changing to UPPER Case " +word2.toUpperCase());
                // Trimming the word
                String word4 = " Learn Share Learn ";
System.out.println("Trim the word " + word4.trim());
                // Replacing characters
                StringBuffer str1 = "MIT POLYTECHNIC";
```



StringBuffer Class

- String class has some disadvantages
  - It is immutable means once value is assigned to object, we are not able to change it
  - Ex: Sting s1="MIT"; then value MIT can not be changed
- So StringBuffer Class is designed which is mutable

  Create Object: StringBuffer str=new StringBuffer();
- Methods are

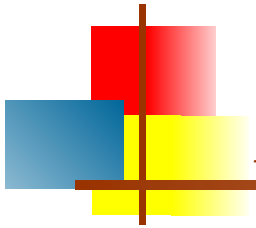
  1 StringBuffer append(String s)

  2 StringBuffer reverse()

  3 void delete(int start, int end)

  4 void insert(int offset, int i)

  5 StringBuffer replace(int start, int end, String str)

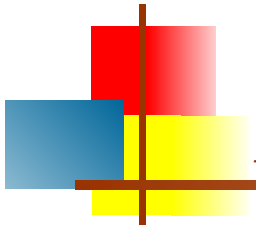


Array of Objects

- It is collection of objects of same class

  Student std[]=new Student[5];

  this is an array of five objects of student class.
- Drawbacks of array of objects
  - We can not modify, add or delete array elements
  - Length of array is fixed so expansion or reduction in size of array
  - Objects are of same class only.
- So new concept of Vectors is derived



Array Vs Vector :

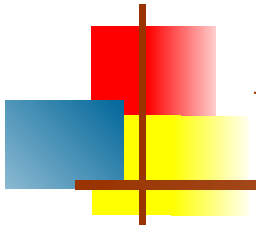
- Both Array and Vector in Java are similar. Both are used for storing the data. But there are some basic difference between Array & Vector .

1. Array is the static memory allocation , while vector is the dynamic memory allocation.

2. Array allocates the memory for the fixed size ,in array there is wastage of memory, while Vector allocates the memory dynamically means according to the requirement no wastage of memory .

3. Vector has many more methods in comparing to Array.

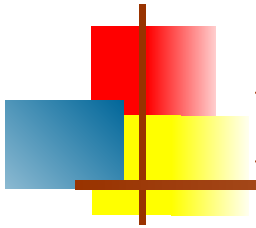
4. By using Vector we can add, remove, & find the size of elements dynamically ,while it is not possible in Array.



Vectors

- The Vector class implements a growable array of objects.
- Vectors basically fall in legacy classes but now it is fully compatible with collections.
- It is found in the java.util package and implements the List interface
- Constructor of Vector

SN	Constructor	Description
1)	vector()	It constructs an empty vector with the default size as 10.
2)	vector(int initialCapacity)	It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
3)	vector(int initialCapacity, int capacityIncrement)	It constructs an empty vector with the specified initial capacity and capacity increment.
4)	Vector(Collection<? extends E> c)	It constructs a vector that contains the elements of a collection c.



Linear Search in Java

Linear search is used to search a key element from multiple elements. Linear search is less used today because it is slower than binary search and hashing.

Algorithm:

- Step 1: Read the array
- Step 2: Match the key element with array element
- Step 3: If key element is found, return the index position of the array element
- Step 4: If key element is not found, return -1

Linear Search

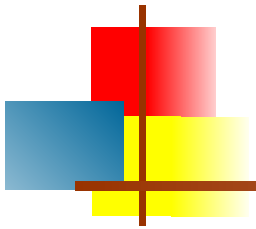
```
import java.util.Scanner;
  class LinearSearchExample
{
 public static void main(String args[])
 {
  int c, n, search, array[];
     Scanner in = new Scanner(System.in);
  System.out.println("Enter number of elements");
  n = in.nextInt();
  array = new int[n];
     System.out.println("Enter those " + n + " elements");

  for (c = 0; c < n; c++)
   array[c] = in.nextInt();

  System.out.println("Enter value to find");
  search = in.nextInt();

  for (c = 0; c < n; c++)
  {
   if (array[c] == search)     /* Searching element is present */
   {
      System.out.println(search + " is present at location " + (c + 1) + ".");
       break;
   }
 }
 if (c == n)  /* Element to search isn't present */
```

Vector class methods

1)	add()	It is used to append the specified element in the given vector.
2)	addElement()	It is used to append the specified component to the end of this vector. It increases the vector size by one.
3)	capacity()	It is used to get the current capacity of this vector.
4)	clear()	It is used to delete all of the elements from this vector.
5)	clone()	It returns a clone of this vector.
6)	contains()	It returns true if the vector contains the specified element.
7)	copyInto()	It is used to copy the components of the vector into the specified array.
8)	elementAt()	It is used to get the component at the specified index.
9)	equals()	It is used to compare the specified object with the vector for equality.
10)	firstElement()	It is used to get the first component of the vector.
11)	insertElementAt()	It is used to insert the specified object as a component in the given vector at the specified index.
12)	isEmpty()	It is used to check if this vector has no components.
13)	removeAll()	It is used to delete all the elements from the vector that are present in the specified collection.
14)	removeAllElements()	It is used to remove all elements from the vector and set the size of the vector to zero.
15)	removeElement()	It is used to remove the first (lowest-indexed) occurrence of the argument from the vector.
16)	removeElementAt()	It is used to delete the component at the specified index.
17)	replaceAll()	It is used to replace each element of the list with the result of applying the operator to that element.
18)	size()	It is used to get the number of components in the given vector.
19)	Sort()	It is used to sort the list according to the order induced by the specified Comparator.

Program using Vector(Expt 14)

```
import java.util.*;
import java.io.*;
class AddElementsToVector {
        public static void main(String[] arg)
        {
        // create default vector
        Vector v1 = new Vector();
        // Add elements using add() method
        v1.add(1);
        v1.add(2);
        v1.add("Polytechnic");
        v1.add("MIT");
        v1.add(3);
        // print the vector to the console
        System.out.println("Vector v1 is " + v1);
        // create generic vector
        Vector<Integer> v2 = new Vector<Integer>();
        v2.add(1);
        v2.add(2);
        v2.add(3);
        System.out.println("Vector v2 is " + v2); }}
```



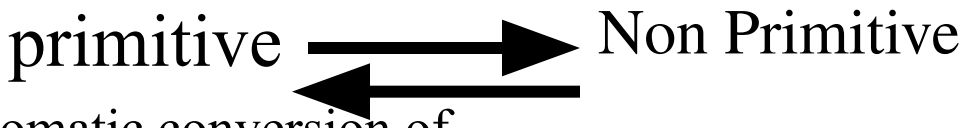
Output

Vector v1 is [1, 2, Polytechnic, Pune, 39.99]

Vector v2 is [1, 2, 3]

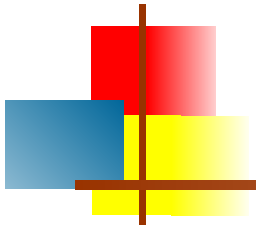
Wrapper Classes

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.



- The automatic conversion of primitives into objects is unboxing
- The automatic conversion of primitive into an object is autoboxing
- Wrapper classes available in java are

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean



```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Number Class methods

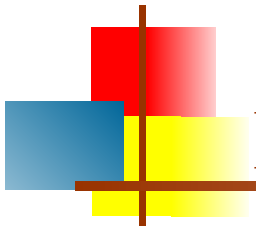
Modifier & Type	Method	Description
Byte	byteValue()	It converts the given number into a byte type and returns the value of the specified number as a byte.
abstract double	doubleValue()	It returns the value of the specified number as a double equivalent.
abstract float	floatValue()	It returns the float equivalent value of the specified Number object.
abstract int	intValue()	It returns the value of the specified number as an int.
abstract long	longValue()	It returns the value of the specified number object as long equivalent.
short	shortValue()	It returns the value of the specified number as a short type after a primitive conversion.

Program on wrapper classes(Expt 15 & 16)

```
class Wrapping
{
   public static void main(String args[])
   {
      //  byte data type
      byte a = 1;
        // wrapping around Byte object
      Byte byteobj = new Byte(a);
        // int data type
      int b = 10;
        //wrapping around Integer object
      Integer intobj = new Integer(b);
        // float data type
      float c = 18.6f;
        // wrapping around Float object
      Float floatobj = new Float(c);
        // double data type
      double d = 250.5;
        // Wrapping around Double object
      Double doubleobj = new Double(d);
```

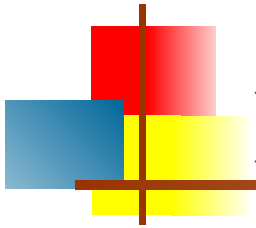
Program on wrapper classes(Expt 15 & 16)

```
// char data type
        char e='a';
        // wrapping around Character object
        Character charobj=e;
        //  printing the values from objects
System.out.println("Values of Wrapper objects (printing as objects)");
        System.out.println("Byte object byteobj:  " + byteobj);
        System.out.println("Integer object intobj:  " + intobj);
        System.out.println("Float object floatobj:  " + floatobj);
  System.out.println("Double object doubleobj:  " + doubleobj);
  System.out.println("Character object charobj:  " + charobj);
```



Program on wrapper classes(Expt 15 & 16)

```
// unwrapping objects to primitive data types
        byte bv = byteobj; int iv = intobj;
        float fv = floatobj;  double dv = doubleobj; char cv = charobj;
        // printing the values from data types
        System.out.println("Unwrapped values (printing as data
types)");
        System.out.println("byte value, bv: " + bv);
        System.out.println("int value, iv: " + iv);
        System.out.println("float value, fv: " + fv);
        System.out.println("double value, dv: " + dv);
        System.out.println("char value, cv: " + cv);
}}
```



Methods

- toString()- coverts object into string
- parseInt()-coverts primitive int to object int
- xxxValue()-value associated with corresponding object like intValue()-value associated with object of Integer

- Thank you