Ch-2 Derived Syntactical Construct in JAVA Marks: 18

Content

Constructor and Methods

Constructor:

Constructors are used to assign initial value to instance variable of the class.

Properties:

- A constructor is a special member which initializes an object immediately upon creation.
- It has the same name as class name in which it resides and it is syntactically similar to any method.
- When a constructor is not defined, java executes a default constructor which initializes all numeric members to zero and other types to null or spaces.
- Once defined, constructor is automatically called immediately after the object is created before new operator completes.
- Constructors do not have return value, not even 'void' because they return the instance if class.
- Constructor called by new operator.

Types of constructors are:

- Default constructor
- Parameterize constructor
- Copy constructor

Default Constructor

- It is constructor which is inserted by Java compiler when no constructor is provided in class. Every class has constructor within it.
- Even abstract class have default constructor.

- By default, Java compiler, insert the code for a zero parameter constructor.
- Default constructor is the no arguments constructor automatically generated unless you define another constructor.
- The default constructor automatically initializes all numeric members to zero and other types to null or spaces.

For e.g.

```
class Rect
 {
 int length, breadth;
 Rect() //constructor
 {
 length=4;
 breadth=5;
 }
 public static void main(String args[])
{
 Rect r = new Rect();
 System.out.println("Area : " +(r.length*r.breadth));
 }
 }
```

Output : Area : 20

Parameterized Constructor

- Constructor which have arguments are known as parameterized constructor.
- When constructor method is defined with parameters inside it, different value sets can be provided to different constructor with the same name.

Example

```
class Rect
{
 int length, breadth;
 Rect(int l, int b) // parameterized constructor
{
length=l; breadth=b;
}
public static void main(String args[])
{
Rect r = new Rect(4,5); // constructor with parameters
Rect r1 = new Rect(6,7);
System.out.println("Area : " +(r.length*r.breadth)); // o/p Area : 20
System.out.println("Area : " +(r1.length*r1.breadth)); // o/p Area : 42
}
}
```

Copy Constructor

- A copy constructor is a constructor that takes only one parameter which is same exact type as the class in which the copy constructor is defined.
- A copy constructor is used to create another object that is copy of the object that it takes as parameter.

For e.g.

```
class student
{
int id;
String name;
student(int i, String n)
{
id=i;
name=n;
}
student (student s)//copy constructor
{
id=s.id;
name=s.name;
}
void display()
{
System.out.println(id+“ ”+name)
}
public static void main(String args[])
student s1=new student(111, “ABC”);
s1.display();
student s2= new student(s1);
s2.display();
}
}
Output
111 ABC
111 ABC
```

Nesting of Methods

- If the method in Java calls method in same class, it is called as nesting of methods.
- When method calls the method in same the same class, (.) operator is not needed.
- A method can call more than one method in the same class.

For e.g.

```
class nest
{
int x=10,y=20;
void display();
System.out.println("Value of x"+x);
System.out.println("Value of y"+y);
}
void add()
{
display();
System.out.println("x+y="+(x+y));
}
}
class method
{
public static void main(String arg[])
{
nest n=new nest();
n.add();
}
}
Output:
Value of x=10
Value of x=20
x+y=30
```

Argument passing the "this" keyword

- The keyword this is useful when you need to refer to instance of the class from its method.
- This can be used inside any method to refer to the current object.
- 'this' is always a reference to the object on which method was invoked.
- You can use 'this' anywhere, a reference to an object of current class type is permitted. 'this' keyword helps us to avoid name conflict.

```
For e.g.
class student
{
int id;
String name;
student(int id, String name)
{
this.id=id;
this.name=name;
}
void display()
System.out.println(id+""+name);
}
 public static void main(String args[])
student s1=new student(1, "ram");
student s2=new student(2, “shyam”);
s1.display();
s2.display();
}
}
Output:
1 ram
2 shyam
```

Varargs: Variable Length Arguments

- The varrags allows the method to accept zero or muliple arguments.
- Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem.
- If we don't know how many arguments we will have to pass in the method, varargs is the betterapproach.

Syntax of varargs: The varargs uses ellipsis i.e. three dots after the data type.

```
return_type method_name(data_type... variableName)
{
//varargs code
}
```

Example:

```
public class VarArgs
{
   public static void main(String[] args)
   {
      System.out.println("The sum of 5 and 10 is " + sum(5, 10));
      System.out.println("The sum of 23, 78, and 56 is "  + sum(23, 78, 56));
      System.out.println("The sum when no parameter is passed is "  + sum());
      int[] numbers = { 23, 45, 89, 34, 92, 36, 90, 60 };
      System.out.println("The sum of array is " + sum(numbers));
   }
       static int sum(int... list)
   {
      int total = 0;

    // add all the values in list array
      for (int i = 0; i < list.length; i++)
      {
         total += list[i];  //total=total+list[i];
      }
      return total;    }      }
```

Garbage Collection

- When the object is not needed any more, the space occupied by such objects can be collected and use for later reallocation. Java performs release of memory occupied by unused objects automatically, it is called ***garbage collection***.
- Since objects are dynamically allocated by using the new operator, you might be wondering how such objects are destroyed and their memory released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator. Java takes a different approach; it handles deallocation for you automatically. The technique that accomplishes this is called garbage collection.
- JVM runs the garbage collector (gc) when it finds time. Garbage collector will be run periodically but you cannot define when it should be called.
- We cannot even predict when this method will be executed. It will get executed when the garbage collector, which runs on its own, picks up the object.

Example:

```
public class A
{
int p, q;
A ( )
{
p = 10;
p = 20;
}
}
class Test
{
public static void main(String args[])
{
A a1= new A();
A a2= new A();
a1=a2;                    // it deallocates the memory of object a1
}
}
```

Finalize() method

- Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held. To handle such situation finalize () method is used.
- Finalize () method is called by garbage collection thread before collecting object. Its the last chance for any object to perform cleanup utility.

- The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the java run time calls the finalize () method on the object.
- Syntax:
- The general form of the finalize () method is:
- protected void finalize ()
- {
- //finalization code
- }

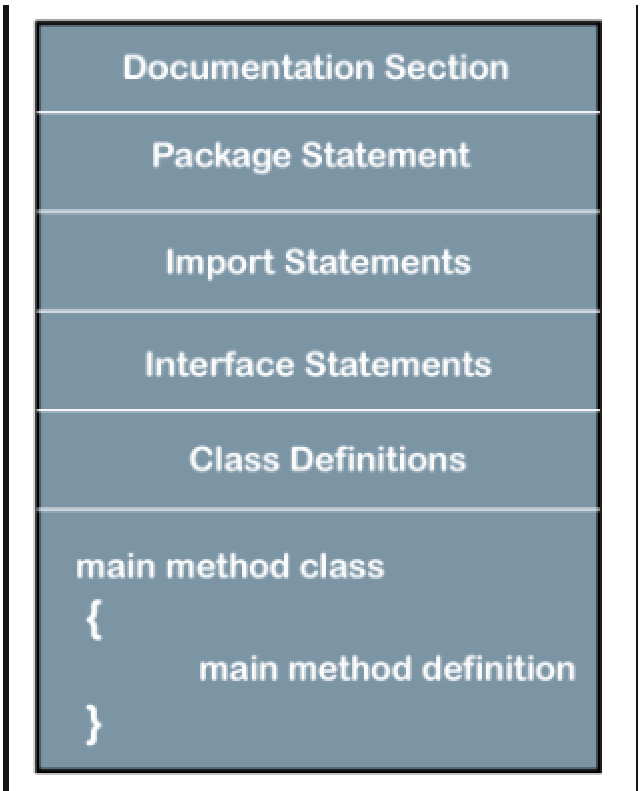
Example:

```
public class Test_gc
{

   public static void main(String[] args)
   {
      Test_gc t = new Test_gc();

      t=null;
     System.gc();  //explicitely
   }
   public void finalize()
   {
      System.out.println("Garbage Collected");
   }
}
```

The object class

Class:

- A class is user defined data type which groups data members and its associated function together.
- Once a new data type is defined using a class, it can be used to create objects of that types.
- Classes create objects and objects use methods to communicate between them.



Structure of Java Program

Structure of program

```
class classname[extends superclassname]
{
[field declaration]
[method declaration]
}
```

Field Declaration

- A class declaration typically includes one or more field that declares data of various types. Tha data or variable define inside the class are called as instance variable.
- The declaration of instance variable is similar to declaration of local variables.
- For e.g.

```
class cat
{
String name;
int age;
float salary;
}
```

Method Declaration

- Only data fields in the class have no. of files. The objects created by such a class cannot respond to any messages.
- Methods are necessary for manipulating the data contain in a class.
- Syntax:-

```
type methodname (parameter _list)
{
Method body;
}
```

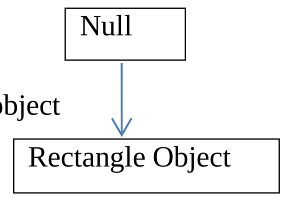
- For e.g:-

```
void getdata(int x, int y){
length=x;
breadth=y;
}
```

Creating Object

- An object in Java is essentially a block of memory that contains space to store all the instance variables.

Action	Statement	Result
Declare	Rectangle rect;	Null
Instantiate	rect= new Rectangle();	



Rect is reference to Rectangle object

<u>Steps for creating object: -</u>

Declaring an object: -

- Declaring a variable to hold an object is just like declaring variable to hold a value of primitive type.
- Declaration does not create new object. It is simply a variable that can refer to an object.
- For e.g. Rectangle rect;

Instantiating: -

- Creating an object is also referred to as instantiating an object.
- Object in Java are created using new operator. The new operator creates an object of the specified class and return reference to that object.
- For e.g. rect=new Rectangle();

  It assigns object reference to the variables. Variable rect is now object of the class.

Initializing an object:-

- A Class provides constructor methods to initialize a new object of that type.
- A may provide multiple constructor o perform different kind of initialization of new object.
- For e.g. Rectangle rect = new Rectangle(10,20)

Sr. No.	**Class**	**Object**
1	A class is piece of program source code that describe a particular type of object.	Object is called as instance of a class.
2	Classes are written by programmer.	Objects are created when program is running.
3	Class specifies the structure.	Objects hold specific value of attributes.
4	Class specifies possible behavior of its objects.	Objects behave appropriately when called upon.

Accessing class members: -

- All variables must be assigned values before they are used.
- Since we are outside the class, we cannot access the instance variables and methods directly.
- We can access those methods and variables using dot(.) operator.
- Syntax: -

```
Object_name.variable_name=value;
Object_name.method_name(parameter_list);
```

- Example:-

```
rect.length=10;
rect.width=20;
rectangle x1=new rectangle();
x1.getdata(10,20);
```

2.2 Visibility Controls

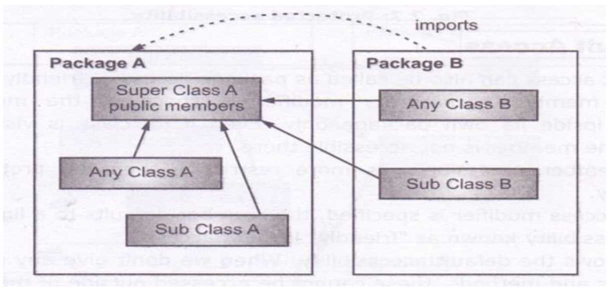
The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class. There are 5 types of java access modifiers:

- public
- private
- default (Friendly)
- protected
- private protected

Public:

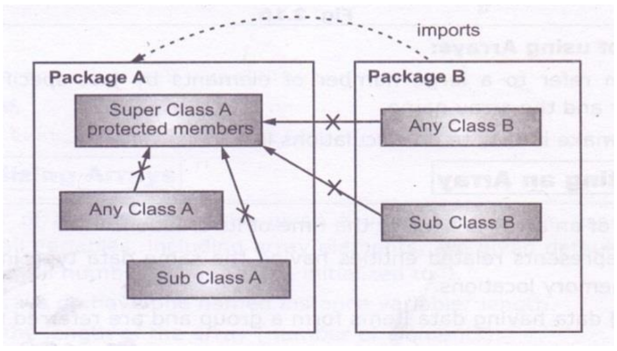
Public Specifiers achieves the highest level of accessibility. Classes, methods, and fields declared as public can be accessed from any class in the Java program, whether these classes are in the same package or in another package.

```
Example:   public   int   number;
 Public void sum() { ------------ }
```



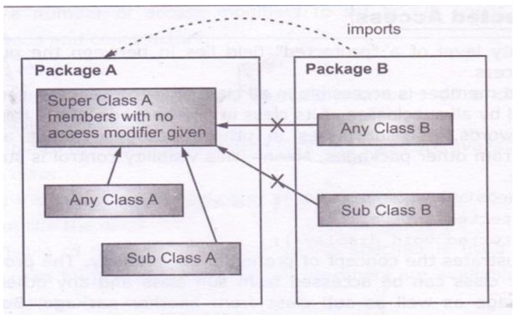
Private:

Private Specifiers achieves the lowest level of accessibility. private methods and fields can only be accessed within the same class to which the methods and fields belong. private methods and fields are not visible within subclasses and are not inherited by subclasses. So, the private access specifier is opposite to the public access specifier. Using Private Specifier, we can achieve encapsulation and hide data from the outside world.



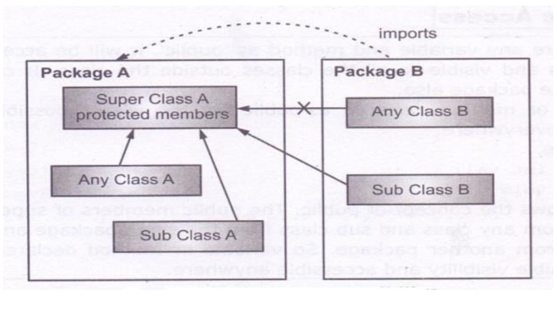
Default (Friendly Access):

When no access modifier is specified, the member defaults to a limited version of public accessibility known as "friendly" level of access. Using default specifier, we can access class, method, or field which belongs to same package, but not from outside this package.



Protected:

The visibility level of a "protected" field lies in between the public access and friendly access. Methods and fields declared as protected can only be accessed by the subclasses in other package or any class within the package of the protected members' class. Note that non sub classes in other packages cannot access the "protected" members.



Private Protected:

- The visibility level lies between private and protected access.
- This modifier makes fields visible in all subclasses regardless of what package they are in.
- These fields are not accessible by other classes in the same package.

Access Level

Modifier	Same class	Subclass In Same package	Other classes In same package	Sub class In other classes	Non sub class In other classes
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Friendly (default)	Yes	Yes	Yes	No	No
Private Protected	Yes	Yes	No	Yes	No
Private	Yes	No	No	No	No

Arrays

- It is collection of related data item that share a certain name. the complete set of values is referred as an array and individual values are called as elements.
- Array can be of any variable type.

Types of array:

1. <u>One Dimensional Array</u>

A list of item can be given one variable name using only one subscript and such variable is called as single subscripted variable or one dimensional array.

Creation of Array

```
type array_name[];     or
type []array_name;
```

For e.g.:-

```
int number[];          or
float []price;
```

Initialization of array: -

To put values into the array is known as initialization. Once an array is instantiated its size cannot be changed**.**

Syntax:-

```
type array_name[]=value;
```

For e.g.:-

int number[]={ 10, 20, 30, 40, 50};

Java allows us to create array using new operator

type array_name=new type[size];

For e.g.:-int number[]= new int[5];

Program 1:

```
class array_simple_example
{
    public static void main (String[] args)
    {
      // declares an Array of integers.
     // int arr[];

      // allocating memory for 5 integers.
     int arr[] = new int[5];

      // initialize the first elements of the array
      arr[0] = 10;

      // initialize the second elements of the array
      arr[1] = 20;

      //so on...
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50;

      // accessing the elements of the specified array
      for (int i = 0; i < arr.length; i++)
         System.out.println("Element at index " + i + " : "+ arr[i]);
    }
}
```

Program 2:

```
class array_simple_example2
{
    public static void main (String[] args)
    {
      // declares an Array of integers.
     // int arr[];

      // allocating memory for 5 integers.
     //int arr[] = new int[5];
```

```
      // initialize the first elements of the array
      int arr[]={4,2,3,5,7};

      // accessing the elements of the specified array
      for (int i = 0; i < arr.length; i++)
        System.out.println("Element at index " + i + " : "+ arr[i]);
    }
}
```

Demonstration of Array Methods:

<u>equals(array1, array2): This method checks if both the arrays are equal or not.</u>

```
// Java program to demonstrate Arrays.equals() method

import java.util.Arrays;

public class arr_equals
 {
    public static void main(String[] args)
    {
          // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 }; // array literal

        // Get the second Arrays
        int intArr1[] = { 1,2,5,4 };
          // To compare both arrays
        System.out.println("Integer Arrays on comparison: "   + Arrays.equals(intArr, intArr1));
    }
}
```

<u>sort(originalArray, fromIndex, endIndex): This method sorts the specified //range of array in ascending order.</u>

```
import java.util.Arrays;
public class arr_sort
 {
    public static void main(String[] args)
    {
         // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        Arrays.sort(intArr, 1, 3);
          System.out.println("Integer Array: "
                         + Arrays.toString(intArr));
    }
}
```

binarySearch(intArr, intKey):This methods searches for the specified element in the array with the help of Binary Search algorithm.

```
import java.util.*;
  public class arr_bin_search
 {
    public static void main(String[] args)
    {
        // Get the Array
      int intArr[] = { 40,10, 20, 15, 22, 35 };
        Arrays.sort(intArr);
        int intKey = 15;
        System.out.println(intKey + " found at index = "   + Arrays.binarySearch(intArr, intKey));
    }
}
```

// Java program to demonstrate Arrays.toString() method

```
import java.util.Arrays;
 public class arr_tostring
{
    public static void main(String[] args)
    {
        // Get the Array
      int intArr[] = { 10, 20, 15, 22, 35 };
        // To print the elements in one line
      System.out.println("Integer Array: "
                    + Arrays.toString(intArr));
    }
}
```

Two Dimensional Array

These are usually represented with row-column (Spreadsheet style)

Syntax:-

type array_name[row_size][Column_size] ;

For e.g.:-

int a[2][5]

Program :

```
class array_multiD
{
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };

        // printing 2D array
        for (int i=0; i< 3 ; i++)
        {
```

```
                                 for (int j=0; j < 3 ; j++)
                                    System.out.print(arr[i][j] + " ");
                                  System.out.println();
                              }
                          }
                      }
```

More examples on an array:

<u>Passing array to methods:</u>

```
class array_passingarray_methods
{
    public static void main(String args[])
   {
      int arr[] = {3, 1, 2, 5, 4,8,9,10};

      // passing array to method m1
      sum(arr);
    }

   public static void sum(int[] arr)
   {
      // getting sum of array values
      int sum = 0;

      for (int i = 0; i < arr.length; i++)
         sum+=arr[i];//sum=sum+arr[i];

      System.out.println("sum of array values : " + sum);
   }
}
```

Array of Object:

```
class Student
{
   public int roll_no;
   public String name;
   Student(int roll_no, String name)
```

```
    {
        this.roll_no = roll_no;
        this.name = name;
    }
}
  // Elements of the array are objects of a class Student.
public class array_stud_example
{
    public static void main (String[] args)
    {
        // declares an Array of integers.
        Student[] arr;
          // allocating memory for 5 objects of type Student.
        arr = new Student[5];
          // initialize the first elements of the array
        arr[0] = new Student(1,"Prince");
          // initialize the second elements of the array
        arr[1] = new Student(2,"Hasan");
         arr[2] = new Student(3,"Amman");
        arr[3] = new Student(4,"Chirag");
        arr[4] = new Student(5,"Shreyas");
          // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at " + i + " : " +
                        arr[i].roll_no +" "+ arr[i].name);
    }
}
```

Strings

- The string class is commonly used for holding and manipulating strings of texts in Java program.
- Strings are class object and implemented using two classes i.e. String and StringBuffer.
- A Java string is an instantiated object of the string class.
- A Java string is not a character array and is not null terminated.

Declaration of String

String String_name;

String_name= new String(“String”);

For E.g.

String name;

name= new String(“ABCD”); or String name= new String(“ABCD”);

Methods of String class

- The String class in Java defines a number of methods that allow us to accomplish a variety of string manipulation tasks.
- The java.lang.String class provides a lot of methods to work on string. For string method import java.lang.*; package has to be imported.
- By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

---

1. **toLowerCase():**

Converts all of the characters in this String to lower case.

Syntax: s1. toLowerCase()

Example: String s="Sachin";

System.out.println(s.toLowerCase());

Output: sachin

---

2. **toUpperCase():**

Converts all of the characters in this String to upper case

Syntax: s1.toUpperCase()

Example: String s="Sachin";

System.out.println(s.toUpperCase());

Output: SACHIN

---

3. **trim():**

Returns a copy of the string, with leading and trailing whitespace omitted.

Syntax: s1.trim()

Example: String s="   Sachin   ";

System.out.println(s.trim());

Output:Sachin

---

4. **replace():**

Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

Syntax: s1.replace(‘x’,’y’)

Example:
```
String s1="Java is a programming language. Java is a platform.";
String s2=s1.replace("Ja","Ka");//replaces all occurrences of "Ja" to "Ka"
System.out.println(s2);
```
Output: Kava is a programming language. Kava is a platform.

---

5. **charAt():**

Returns the character at the specified index.

Syntax: s1.CharAt(n)

Example:
```
String s="Sachin";
System.out.println(s.charAt(0));
System.out.println(s.charAt(3));
```
Output:
```
S
H
```

---

6. **equals():**

The String equals() method compares the original content of the string. It compares values of string for equality.

Syntax: s1.equals(s2)

Example:
```
String s1="Sachin";
String s2="Sachin";
String s3="Saurav"
System.out.println(s1.equals(s2));     //true
System.out.println(s1.equals(s3));    //false
```
Output:
```
True
False
```

---

7. **equalsIgnoreCase():**

comparing two Strings by ignoring case.

Syntax: s1.equalsIgnoreCase(s2)

Example: String s1="Sachin";

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

Output:

```
False
True
```

---

8. **compareTo():**

   The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

   Suppose s1 and s2 are two string variables. If:

   s1 == s2 :0

   s1 > s2 : positive value

   s1 < s2 : negative value

   Syntax: s1.compareTo(s2)

   Example:
   ```
   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";
   System.out.println(s1.compareTo(s2));          //0
   System.out.println(s1.compareTo(s3));          //1(because s1>s3)
   System.out.println(s3.compareTo(s1));          //-1(because s3 < s1 )
   ```

---

9. **length():**

   The string length() method returns length of the string.

   Syntax: s1.length()

   Example:
   ```
   String s="Sachin";
   System.out.println(s.length());
   ```

   Output: 6

---

10. **concat():**

    string concatenation forms a new string that is the combination of multiple strings.

    Syntax: s1.concat(s2)

    Example:
    ```
    String s1="Sachin ";
    String s2="Tendulkar";
    String s3=s1.concat(s2);
    ```

System.out.println(s3);

Output: Sachin Tendulkar

---

11. Substring():

A part of string is called substring. In other words, substring is a subset of another string.

Syntax: s1.substring(n)

Example: String s="Sachin Tendulkar";

System.out.println(s.substring(6));

Output: Tendulkar

---

12. substring(n,m):

Gives substring starting from nth character up to mth .

Syntax: s1.substring(n,m)

String s="Sachin Tendulkar";

System.out.println(s.substring(0,6));

Output: Sachin

---

13. indexOf():

Gives the position of the first occurrence of ‘x’ in the string s1.

Syntax: s1.indexOf(‘x’)

Example: String s1=”sachin”;

s1.indexOf(‘a’);

Output: 1

---

14. indexOf(‘x’,n):

Gives the position of ‘x’ that occurs after nth position in the string s1.

Example: String s1=”sanjay”; s1.indexOf(‘a’,2);

Output: 4

---

15. valueOf():

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

Syntax: String.ValueOf(p)

Example: int a=10;

String s=String.valueOf(a);

System.out.println(s+10);

Output: 1010

---

16. endsWith():

This method tests if this string ends with the specified suffix.

Syntax: endsWith(String suffix)

Example:
```
String Str = new String("This is really not immutable!!");
boolean retVal;
retVal = Str.endsWith( "immutable!!" );
System.out.println("Returned Value = " + retVal );
retVal = Str.endsWith( "immu" );
System.out.println("Returned Value = " + retVal );
```

Output:
```
Returned Value = true
Returned Value = false
```

---

17. startsWith():

This method has two variants and tests if a string starts with the specified prefix beginning a specified index or by default at the beginning.

Syntax: startsWith(String prefix)   or   startsWith(String prefix, int toffset)

Example:
```
String Str = new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :" );
System.out.println(Str.startsWith("Welcome") );
System.out.print("Return Value :" );
System.out.println(Str.startsWith("Tutorials") );
```

Output:
```
Return Value : true
Return Value : false
```

Program to implement all String program in one method.

```
class box
{
public static void main(String arg[])
{
String S1="sachin";
String S2="nikita";
```

```
System.out.println(S1.toLowerCase());
System.out.println(S1.toUpperCase());
String S3=S2.replace("nikita","ratan");
System.out.println(S3);
System.out.println(S1.charAt(0));
System.out.println(S1.charAt(3));
String S4="nikita";
String S5="nikita";
String S6="omkars";
System.out.println(S4.equals(S5));
System.out.println(S4.equals(S6));
String S7="nikita";
String S8="NIKITA";
System.out.println(S7.equals(S8));
System.out.println(S7.equalsIgnoreCase(S8));
System.out.println(S1.length());
String  S9="nikita";
String S10="kumthekar";
String S11=S9.concat(S10);
System.out.println("String S11="+S11);
String S12="SachinTendulkar";
System.out.println(S12.substring(6));
System.out.println(S12.substring(0,6));
System.out.println(S12.substring(5,9));
String S13="samreen";
System.out.println(S13.indexOf('a'));
String S14="Sanjay";
System.out.println(S14.indexOf('a',4));
String S15="sachin";
String S16="sachin";
String S17="ratan";
System.out.println(S15.compareTo(S16));
System.out.println(S15.compareTo(S17));
System.out.println(S17.compareTo(S15));
}
}
```

StringBuffer Class Methods

- It is used to represent character that can be modified.
- This is simply used foe concatenation or manipulation of string.
- We can insert characters or substrings in middle of string or append another string to the end.
- Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in Java is same as string class except it is mutable.

Methods of String class

- Java StringBuffer class is used to created mutable (modifiable) string.

---

1. **Append():**

   The append() method concatenates the given argument with this string.

   Syntax: s1.append(s2)

   Example:
   ```
   StringBuffer sb=new StringBuffer("Hello ");
   sb.append("Java");            //now original string is changed
   System.out.println(sb);
   ```
   output: Hello Java

---

2. **Insert():**

   The insert() method inserts the given string with this string at the given position.

   Syntax: s1.insert(n,s2)

   Example:
   ```
   StringBuffer sb=new StringBuffer("Hello ");
   sb.insert(1,"Java");                 //now original string is changed
   System.out.println(sb);
   ```
   Outputs: HJavaello

3. **Replace():**

The replace() method replaces the given string from the specified beginIndex and endIndex.

Syntax: replace(int start, int end, String str)

Example: StringBuffer sb=new StringBuffer("Hello");

```
sb.replace(1,3,"Java");
System.out.println(sb);
```

Output: HJavao

4. **delete():**

The delete() method of StringBuffer class deletes the string from the specified beginIndex toendIndex.

Syntax: delete(int start, int end)

example: StringBuffer sb=new StringBuffer("Hello");

```
sb.delete(1,3);
System.out.println(sb);//prints Ho
```

5. **reverse():**

The reverse() method of StringBuilder class reverses the current string.

Example: StringBuffer sb=new StringBuffer("Hello");

```
sb.reverse();
System.out.println(sb);
```

Output: olleH

6. **setCharAt():**

Modifies the nth character to x.

Example: StringBuffer s1= new StringBuffer(“vijay”);

```
s1.setCharAt(3,’e’);
```

Output: vijey

7. **deleteCharAt():**

This is the deleteCharAt() function which is used to delete the specific character fromthe buffered string by mentioning that's position in the string.

Example: StrigBuffer s1 = new StringBuffer("vijay");

s1.deleteCharAt(2);

Output: viay

---

8. **setLength()**:

This method sets the length of the character sequence. Sets the length of the string s1 to n. if n<s1.length() s1 is truncated. If n>s1.length() zeros are added to s1.

Syntax: s1.setLength(n)

Example:

```
StringBuffer sb = new StringBuffer("Java Programming");
System.out.println(sb + "\nLength = " + sb.length());// Java ProgrammingLength = 16
sb.setLength(4);
System.out.println(sb + "\nLength = " + sb.length());// Java Length = 4
```

Output: Java

Program to implement all StringBuffer class

```
import java.lang.*;
class method
{
public static void main(String arg[])
{
StringBuffer s1=new StringBuffer("Hello");
StringBuffer s2=new StringBuffer("Java");
System.out.println(s1.append(s2));
StringBuffer s3=new StringBuffer("Hello");
StringBuffer s4=new StringBuffer("Java");
System.out.println(s3.insert(2,s4));
StringBuffer s5=new StringBuffer("Hello");
s3.replace(1,3,"Java");
System.out.println(s5);
StringBuffer s6=new StringBuffer("Hello");
s6.delete(1,3);
System.out.println(s6);
StringBuffer s7=new StringBuffer("Java");

s7.reverse();

System.out.println(s7);

StringBuffer s8=new StringBuffer("Java");

s8.setCharAt(3,'e');

System.out.println(s8);

StringBuffer s9=new StringBuffer("Welcome");

s9.deleteCharAt(2);

System.out.println(s9);

StringBuffer s10=new StringBuffer("Java Programming");

System.out.println(s10+"\nlength=" + s10.length());

s10.setLength(4);

System.out.println(s10+"\nlength=" + s10.length());

}

}
```

Compare String Class and StringBuffer Class

String class	StringBuffer class
String is a major class	StringBuffer is a peer class of String
Length is fixed (immutable)	Length is flexible (mutable)
Contents of object cannot be modified	Contents of object can be modified
Object can be created by assigning String constants enclosed in double quotes.	Objects can be created by calling constructor of StringBuffer class using “new”
Ex:- String s=”abc”‖;	Ex:- StringBuffer s=new StringBuffer (“abc”);

Vector

- Vector class is in java.util package of java.
- Vector is dynamic array which can grow automatically according to the requirement.
- Vector does not require any fix dimension like String array and int array.
- Vectors are used to store objects that do not have to be homogeneous.
- Vector contains many useful methods.

Vectors are created like arrays. It has three constructor methods

- Vector list = new Vector();   //declaring vector without size
- Vector list = new Vector(3); //declaring vector with size
- Vector list = new Vector(5,2);   //create vector with initial size and whenever it need to grows, it grows by value specified by increment capacity

<u>Vector Methods</u>

Methods	Task performed
firstElement()	It returns the first element of the vector.
lastElement()	It returns last element.
addElement(item)	Adds the item specified to the list at the end.
elementAt(int index)	Gives the name of the object present at index position.
size()	Gives the number of objects present.
setSize(int newSize)	Sets the size of this vector.
Set(int index, object element)	Replaces the element at the specified position in the vector with the specified element.
capacity()	Returns the current capacity of this vector. By default capacity is 10 for vector.
Boolean Contains(object elem)	Tests if the specified object is a component in the vector.
clear()	Removes all of the elements from this vector
remove(Object o)	Removes the first occurrence of the specified element in this vector, If the vector does not contain the element, it is unchanged.
remove(int index)	Removes the element at the specified position in this vector.
removeElementAt(n)	Removes the item stored in the nth position of the list.
removeElement(item)	Removes the specified item from the list.
removeRange(int fromIndex, int toIndex)	This method removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
removeAllElements()	Removes all the elements in the list.
copyInto(array)	Copies all items from list of array.
insertElementAt(item, n)	Inserts the item at nth position.
boolean contains(Object elem)	Tests if the specified object is a component in this vector.
elements()	It returns an enumeration of the element.
hasMoreElements()	It checks if this enumeration contains more elements or not.
nextElement()	It checks the next element of the enumeration.
int indexOf(Object o)	This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int indexOf(Object o, int index)	This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.

int lastIndexOf(Object o)	This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

Differentiate between Array and Vector

Array	**Vector**
An array is a structure that holds multiple values of the same type.	The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.
An array is a homogeneous data type where it can hold only objects of one data type.	Vectors are heterogeneous. You can have objects of different data types inside a Vector.
After creation, an array is a fixed-length structure.	The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
Array can store primitive type data element.	Vector are store non-primitive type data element.
Array is unsynchronized i.e. automatically increase the size when the initialized size will be exceed.	Vector is synchronized i.e. when the size will be exceeding at the time; vector size will increase double of initial size.
Declaration of an array : int arr[] = new int [10];	Declaration of Vector: Vector list = new Vector(3);
Array is the static memory allocation.	Vector is the dynamic memory allocation.
Array allocates the memory for the fixed size ,in array there is wastage of memory.	Vector allocates the memory dynamically means according to the requirement no wastage of memory.
No methods are provided for adding and removing elements.	Vector provides methods for adding and removing elements.
In array wrapper classes are not used.	Wrapper classes are used in vector.
Array is not a class.	Vector is a class.

Q. Write a program to implement a vector class and its method for adding and removing elements. After remove display remaining list.

```
import java.io.*;
import java.lang.*;
import java.util.*;
class vector2
{
public static void main(String args[])
{
vector v=new vector();
Integer s1=new Integer(1);
Integer s2=new Integer(2);
String s3=new String("fy");
String s4=new String("sy");
Character s5=new Character('a');
Character s6=new Character('b');
Float s7=new Float(1.1f);
Float s8=new Float(1.2f);
v.addElement(s1);
v.addElement(s2);
v.addElement(s3);
v.addElement(s4);
v.addElement(s5);
v.addElement(s6);
v.addElement(s7);
v.addElement(s8);
System.out.println(v);
v.removeElement(s2);
v.removeElementAt(4);
System.out.println(v);
}
}
```

Q. Write a program to create a vector with seven elements as (10, 30, 50, 20, 40, 10, 20). Remove element at 3rd and 4th position. Insert new element at 3rd position. Display the original and current size of vector.

```
import java.util.*;
public class VectorDemo
{
public static void main(String args[])
{
Vector v = new Vector();
v.addElement(new Integer(10));
v.addElement(new Integer(20));
v.addElement(new Integer(50));
v.addElement(new Integer(20));
v.addElement(new Integer(40));
v.addElement(new Integer(10));
v.addElement(new Integer(20));
System.out println(v.size());          // display original size
v.removeElementAt(2);                  // remove 3rd element
v.removeElementAt(3);                  // remove 4th element
v.insertElementAt(11,2)                // new element inserted at 3rd position
System.out.println("Size of vector after insert delete operations: " + v.size());
}
}
```

Q. Write a program to implement vector class and its method for adding and removing elements. (6 Marks)

```
import java.util.*;
 public class VectorDemo
{
 public static void main(String args[])
{
Vector v = new Vector();
v.addElement(new Integer(1));
v.addElement(new Double(2523));
v.addElement(new Float(3.55f));
v.addElement(“java”);
 System.out.println("Size of vector after four additions: " + v.size());
v.removeElementAt(2);
v.removeElementAt(1);
System.out.println("Size of vector after removed : " + v.size()); System.out.println("\nElements in
vector:"); System.out.println(v);
 }
 }
```

Output:-

```
Size of vector after four additions:4
Size of vector after removed :2
 1 java
```

Q. Write a program to implement vector that accept 5 elements from command line arguments and store them in a vector & display the object stores in the vector.

```
import java.util.*;
class vectordemo
{
public static void main(String arg[])
{
Vector v=new Vector();
String name;
for(int i=0; i<arg.length;i++)
{
name=arg[i];
v.addElement(arg[i]);
}
```

Wrapper Class

- Vectors cannot handle primitive data types like int, float, long, char and double.
- Primitive data types may be converted into object types by using the wrapper classes.
- The wrapper classes for numeric primitives have the capability to convert any valid number in string format into its corresponding primitive object.
- For example “10" can be converted into an Integer by using:

```
Integer intVal = new Integer("10");
```

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a variable and in this variable, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.

Simple data types and their corresponding wrapper class types are as follows:

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

Wrapper classes have number of unique methods for handling primitive data types and object which are listed below: -

1. **Converting primitive numbers to object numbers using constructor methods**

Constructor calling	**Conversion action**
Integer IntVal = new Integer(i);	Primitive integer to Integer objects.
Float floatVal = new Float(f);	Primitive float to Float object.
Double DoubleVal = new Double(d);	Primitive double to Double object.
Long LongVal = new Long(l);	Primitive long to Long object.

2. **Converting object numbers to primitive numbers using typeValue() method**

Method calling	**Conversion action**
int i = IntVal.intValue();	Object to primitive integer.
float f = FloatVal.floatValue();	Object to primitive float.
long l = LongVal.longValue();	Object to primitive long.
double d = DoubleVal.doubleValue();	Object to primitive double.

3. **Converting numbers to Strings using to String() method**

Method calling	**Conversion action**
str = Integer.toString(i);	Primitive integer to string.
str = Float.toString(f);	Primitive float to string.
str = Double.toString(d);	Primitive double to string.
str = Long.toString(l)	Primitive long to string.

4. **Converting String objects to numeric objects using the static method ValueOf()**

Method calling	Conversion action
DoubleVal = Double.Valueof(str);	Converts string to Double object.
FloatVal = Float.ValueOf(str);	Converts string to Float object.
IntVal = Integer.Valueof(str);	Converts string to Integer object.
LongVal = Long.ValueOf(str);	Converts string to Long object.

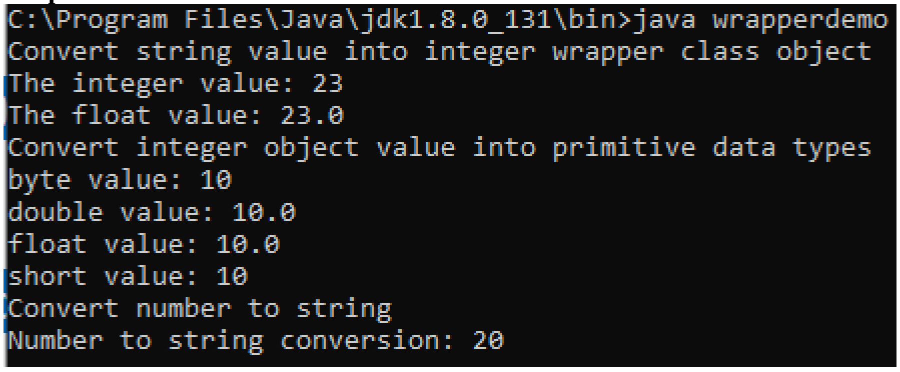
5. **Converting numeric strings to primitive numbers using Parsing methods**

Method Calling	**Conversion action**
int i = Integer.parseInt(str);	Converts string to primitive integer
long i = Long.parseLong(str);	Converts string to primitive long
float i = Float.parseFloat(str);	Converts string to primitive float

Example:

```
class wrapperdemo
{
public static void main(String args[])
{
//Convert string value into integer Wrapper class object
String str = "23";
Integer i = Integer.valueOf(str);
System.out.println("Convert string value into integer wrapper class object");
System.out.println("The integer value: "+i);
Float f=Float.valueOf(str);
System.out.println("The float value: "+f);

//convert integer object value into primitive data types byte, short and double
System.out.println("Convert integer object value into primitive data types");
Integer itr = new Integer(10);
System.out.println("byte value: "+itr.byteValue());
System.out.println("double value: "+itr.doubleValue());
System.out.println("float value: "+itr.floatValue());
System.out.println("short value: "+itr.shortValue());

//Convert number to string
System.out.println("Convert number to string");
int n=20;
str=Integer.toString(n);
System.out.println("Number to string conversion: "+str);
}
}
```

Output:



```
C:\Program Files\Java\jdk1.8.0_131\bin>java wrapperdemo
Convert string value into integer wrapper class object
The integer value: 23
The float value: 23.0
Convert integer object value into primitive data types
byte value: 10
double value: 10.0
float value: 10.0
short value: 10
Convert number to string
Number to string conversion: 20
```

Autoboxing and Unboxing

- The Autoboxing and *Unboxing* was released with the Java 5.
- During assignment, the automatic transformation of primitive type (int, float, double etc.) into their object equivalents or wrapper type (Integer, Float, Double,etc) is known as ***Autoboxing*.**
- During assignment or calling of constructor, the automatic transformation of wrapper types into their primitive equivalent is known as **Unboxing**.

Example:

```
int i = 0;  //p
i = new Integer(5);              // auto-unboxing
Integer i2 = 5;        // autoboxing
```

Wrapper class Examples

1) **To convert integer number to string :**

   Method is toString()

   String str = Integer.toString(i) –converts the integer i to string value.

   class IntegerExample

```
 {
 public static void main(String a[])
 {
String str = Integer.toString(5);
 System.out.println("String value of i "+str );
 }
 }
```

---

2) To convert numeric string to integer number :

Method is parseInt() int i = Integer.parseInt(str) – converts the string value of numeric string str to int i.

```
class StringConversion
{
public static void main(String a[])
 {
 String str = "5";
 int i = Integer.parseInt(str);
 System.out.println("Integer value of str is "+i);
}
 }
```

---

3) To convert object number to primitive number using typevalue() method:

The method here is typeValue(), meaning that type can be the data type of the number.

For example : if x is an Integer object with value 10,then it can be stored as primitive int y with the method intValue() as

```
Integer x = new Integer(10);
int y = x.intValue();
```

Similarly, it can be done with other numeric types as floatValue(), doubleValue() etc

Q. What is the use of wrapper classes in Java? Explain float wrapper with its methods.

- Java provides several primitive data types. These include int (integer values), char (character), double (doubles/decimal values), and byte (single-byte values).
- Sometimes the primitive data type isn't enough and we may have to work with an integer object.
- Wrapper class in java provides the mechanism to convert primitive into object and object into

primitive.

- Float wrapper Class: Float wrapper class is used to wrap primitive data type float value in an object.

  Methods:

  1. floatValue( ) method:
     It is used to return value of calling object as float.
  2. parseFloat( ) method:
     It is used to return float of a number in a string in radix 10.
  3. toString(float f) method:
     It is used to find the string equivalent of a calling object.
  4. valueOf(String s) method:
     It is used to return Float object that has value specified by str.
  5. compare(float f1, float f2) method: It is used to compare values of two numbers.
     If it returns a negative value, then, n1<n2.
     If it returns a positive value, then, n1>n2.
     If it returns 0, then both the numbers are equal.
  6. compareTo (float f1) method:
     is used to check whether two numbers are equal, or less than or greater than each other.
     If the value returned is less than 0 then, calling number is less than x.
     If the value returned is greater than 0 then, calling number is greater than x.
     If value returned is 0, then both numbers are equal.
  7. equals(Object obj) method:
     It is used to check whether two objects are equal. It returns true if objects are equal,otherwise false.

Q. Write a program to accept number from user & convert it into binary by using wrapper class method

```
 import java.io.*;
public class MyIntegerToBinary
 {
 public static void main(String args[])throws IOException
{
int i;
BufferedReader obj = new BufferedReader (new InputStreamReader(System.in));
System.out.print("Enter number that you want to convert to binary: ");
i=Integer.parseInt(obj.readLine());
String binary = Integer.toBinaryString(i); //Converts integer to binary
System.out.println("Binary value: "+binary);
 }
 }
```

Advantages of wrapper class:

In Java, sometimes we might need to use objects instead of primitive data types.

For example, while working with collections.

```
// error
Vector<int> list = new Vector<>();
// runs perfectly
Vector <Integer> list = new Vector <>();
```

In such cases, wrapper classes help us to use primitive data types as objects.

We can store the null value in wrapper objects.

For example,

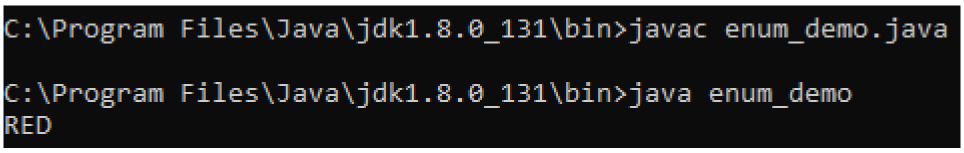
```
// generates an error
int a = null;
// runs perfectly
Integer a = null;
```

Enumerated Types

- Enum type is atype which consists of fixed set of constant fields. This keyword can be used similar to the static final constants.
- Enumerations serve the purpose of representing a group of named constants in a programming language.
- In Java (from 1.5), enums are represented using **enum** data type. Java enums are more powerful than C/C++ enums . In Java, we can also add variables, methods and constructors to it. The main objective of enum is to define our own data types(Enumerated Data Types).

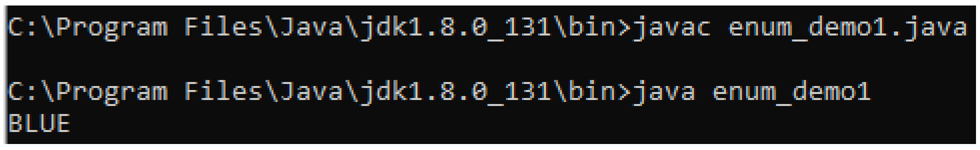
Declaration of enum in java :
Enum declaration can be done outside a Class or inside a Class but not inside a Method.

Example:
```
enum Color
{
   RED, GREEN, BLUE;
}
 public class enum_demo
{
       public static void main(String[] args)
   {
       Color c1 = Color.RED;
       System.out.println(c1);
   }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>javac enum_demo.java

C:\Program Files\Java\jdk1.8.0_131\bin>java enum_demo
RED
```

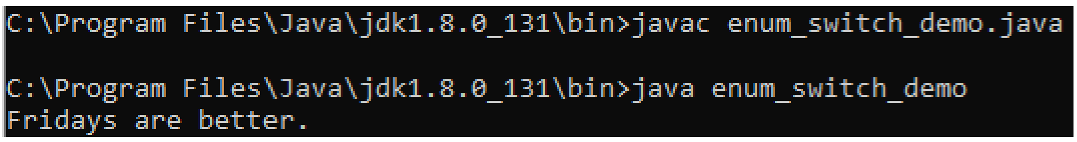
```
// enum declaration inside a class.
public class enum_demo1
{
   enum Color
   {
      RED, GREEN, BLUE;
   }

   public static void main(String[] args)
   {
      Color c1 = Color.BLUE;
      System.out.println(c1);
   }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>javac enum_demo1.java

C:\Program Files\Java\jdk1.8.0_131\bin>java enum_demo1
BLUE
```

Important points of enum :

Every enum internally implemented by using Class.
```
/* internally above enum Color is converted to
class Color
{
    public static final Color RED = new Color();
    public static final Color BLUE = new Color();
    public static final Color GREEN = new Color();
}*/
```
Every enum constant represents an object of type enum.
enum type can be passed as an argument to **switch** statement.
```
enum Day
{
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY;
}
public class enum_switch_demo
{
    Day day;
    public enum_switch_demo(Day day)   // Constructor
    {
        this.day = day;
    }

    // Prints a line about Day using switch
    public void dayIsLike()
    {
        switch (day)
        {
        case MONDAY:
            System.out.println("Mondays are bad.");
            break;
        case FRIDAY:
            System.out.println("Fridays are better.");
            break;
        case SATURDAY:
        case SUNDAY:
            System.out.println("Weekends are best.");
            break;
        default:
            System.out.println("Midweek days are so-so.");
            break;
        }
    }
     public static void main(String[] args)
    {
        String str = "FRIDAY";
        enum_switch_demo t1 = new enum_switch_demo(Day.valueOf(str));
        t1.dayIsLike();
    }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>javac enum_switch_demo.java

C:\Program Files\Java\jdk1.8.0_131\bin>java enum_switch_demo
Fridays are better.
```

Every enum constant is always implicitly public static final. Since it is static, we can access it by using enum Name. Since it is final, we can’t create child enums.

Advantages:

- Compile time type safety.
- We can use the enum keyword in switch statements.

Programs:

Q. Write a program to convert a decimal number to binary form and display the value

```
import java.lang.*;
import java.io.*;
class dtob
{
public static void main(String args[]) throws IOException
{
BufferedReader bf=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the decimal value=");
String hex=bf.readLine();
int i=Integer.parseInt(hex);
String bi=Integer.toBinaryString(i);
System.out.println("Binary number="+bi);
}
}
```

Output:

```
Enter the decimal value=12
Binary number=1100
```

Q. Define a class item having data member code and price. Accept data for one object and display it.

```
import java.io.*;
class Item_details
{
int code;
float price;
Item_details()
{
code=0;
price=0;
}
Item_details(int e,float p)
{
code=e;
price=p;
}
void putdata()
{
System.out.println("Code of item :"+code);
System.out.println("Price of item:"+price);
}
public static void main(String args[]) throws IOException
{
Int co;
Float pr;
BufferedReaderbr=new BufferedReader (new InputStreamReader(System.in)); System.out.println("enter
code and price of item");
co=Integer.parseInt(br.readLine());
pr=Float.parseFloat(br.readLine());
Item_details obj=new Item_details(co,pr);
obj.putdata();
}
}
```

Q. Write a program to accept a number as command line argument and print the number is even or odd

```
public class oe
{
public static void main(String key[])
{
int x=Integer.parseInt(key[0]);
if (x%2 ==0)
{
System.out.println("Even Number");
}
else
{
System.out.println("Odd Number");
}
}
}
```

Q. Define a class ‘employee’ with data members empid, name and salary. Accept data for five objects using Array of objects and print it.

```
class employee
{
int empid;
String name;
double salary;
void getdata()
{
BufferedReader obj = new BufferedReader (new InputStreamReader(System.in)); System.out.print("Enter Emp
number : ");
empid=Integer.parseInt(obj.readLine());
System.out.print("Enter Emp Name : ");
name=obj.readLine();
System.out.print("Enter Emp Salary : ");
salary=Double.parseDouble(obj.readLine());
}
void show()
{
System.out.println("Emp ID : " + empid);
System.out.println(“Name : " + name);
System.out.println(“Salary : " + salary);
}
}
classEmpDetails
{
public static void main(String args[])
{
employee e[] = new employee[5];
for(inti=0; i<5; i++)
{
e[i] = new employee();
e[i].getdata();
}
System.out.println(" Employee Details are : ");
for(inti=0; i<5; i++)
e[i].show();
}
}
```

Q. Write a program to accept first name, middle name and surname in three different strings and then concatenate the three strings to make full name.

```
import java.io.*;
class StringConcat
{
public static void main(String[] args) throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str1,str2,str3,namestr="";
System.out.println("Enter first name:");
str1=br.readLine();
System.out.println("Enter middle name:");
str2=br.readLine();
System.out.println("Enter last name:");
str3=br.readLine();
namestr = namestr.concat(str1);
namestr = namestr + " ";
namestr = namestr.concat(str2);
namestr = namestr + " ";
namestr = namestr.concat(str3);
System.out.println("Your FULL NAME is : " + namestr);
}
}
```

Q. Write a program to create and sort all integer array.

```
class bubble
{
public static void main(String args[])
{
int a[]={10,25,5,20,50,45,40,30,35,55};
int i=0;
int j=0;
int temp=0;
int l=a.length;
for(i=0;i<l;i++)
{
for(j=(i+1);j<l;j++)
{
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
System.out.println("Ascending order of numbers:");
for(i=0;i<l;i++)
System.out.println(""+a[i]);
}
}
```

Q. Write a program to check whether the given string is palindrome or not

```
import java.lang.*;
import java.io.*;
import java.util.*;
class palindrome
{
public static void main(String arg[ ]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in)); System.out.println("Enter
String:");
String word=br.readLine( );
int len=word.length(  )-1;
int l=0;
int flag=1;
int r=len;
while(l<=r)
{
if(word.charAt(l)==word.charAt(r))
{
l++;
r--;
}
else
{
flag=0; break;
}
}
if(flag==1)
{
System.out.println("palindrome");
}
else
{
 System.out.println("not palindrome");
}
}
}
```

Q. Write a program to accept a password from the user and authenticate the user if password is correct.

```
import java.io.*;
class test
{
public static void main(String a[]) throws Exception
{
String S1="Admin";
String S2="12345";
String username, userpwd;
try
{
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the user name");
username=b.readLine();
System.out.println("Enter the password");
userpwd= b.readLine();
if(S1.equals(username) && S2.equals(userpwd))
{
System.out.println("Welcome!! You are authenticated");
}
else
{
System.out.println("You have entered wrong information!!");
}
}
catch(Exception e)
{
System.out.println("I/O Error");
}
}
}
```

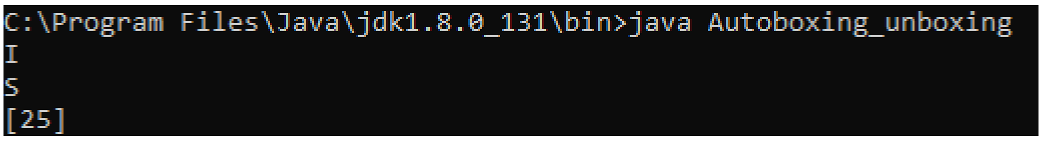
Output:

```
Enter the user name
Admin
Enter the password
12345
Welcome!! You are authenticated
Enter the user name
Admin
Enter the password
123
You have entered wrong information!!
```

More examples on Wrapper classes:

Program1)

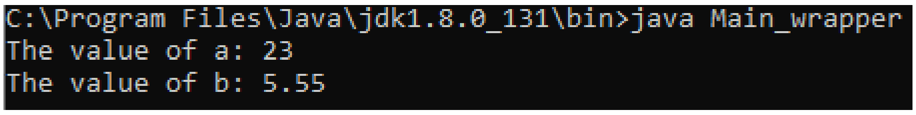
```
import java.util.*;
class Autoboxing_unboxing
{
    public static void main(String[] args)
    {
        char ch = 'I';

        // Autoboxing- primitive to Character object conversion
        Character a = ch;
       System.out.println(a);
Character y = 'S';
 // unboxing - Character object to primitive conversion
char b=y;
System.out.println(b);
        Vector<Integer> v = new Vector<Integer>();

        // Autoboxing because ArrayList stores only objects
        v.add(25);

        // printing the values from object
        System.out.println(v);
    }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>java Autoboxing_unboxing
I
S
[25]
```

Program 2)

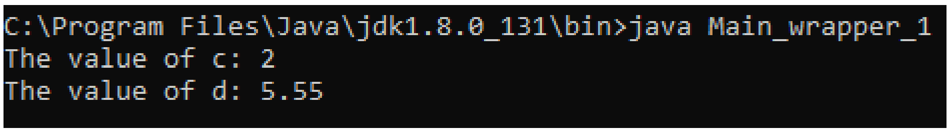
```
class Main_wrapper {
  public static void main(String[] args)
 {

    // creates objects of wrapper class
    Integer aObj = Integer.valueOf(23);
    Double bObj = Double.valueOf(5.55);

    // converts into primitive types
    int a = aObj.intValue();
    double b = bObj.doubleValue();

    System.out.println("The value of a: " + a);
    System.out.println("The value of b: " + b);
  }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>java Main_wrapper
The value of a: 23
The value of b: 5.55
```

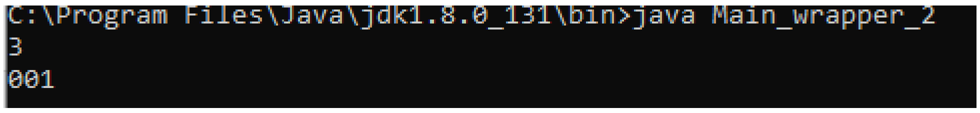
Program 3)

```
class Main_wrapper_1
 {
  public static void main(String[] args)
 {
Integer cObj = Integer.valueOf(2);
// converts into int type
int c = cObj;

Double dObj = Double.valueOf(5.55);
// converts into double type
double d = dObj;
System.out.println("The value of c: " + c);
System.out.println("The value of d: " + d);
}
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>java Main_wrapper_1
The value of c: 2
The value of d: 5.55
```

```
class Main_wrapper_2
{
  public static void main(String[] args)
 {
    Integer myInt = 100;
    String myString = myInt.toString();
    System.out.println(myString.length());
    StringBuffer sb=new StringBuffer(myString);
    System.out.println(sb.reverse());
  }
}
```



```
C:\Program Files\Java\jdk1.8.0_131\bin>java Main_wrapper_2
3
001
```