Unit-7-OOP NOTES

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'s','e','c','t', 'i','o','n'};
String s=new String(ch);
```

is same as:

```
String s="section";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Java String class methods

1. Java String charAt()

The **Java String class charAt()** method returns *a char value at the given index number*.

```
public class CharAtExample{
public static void main(String args[]){
String name="section";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

2. Java String length()

The Java String class length() **method finds the length of a string. The length of the Java string is the same as the Unicode code units of the string.**

Syntax

The signature of the string length() method is given below:

```
public int length()
```

Example

```
public class LengthExample{
public static void main(String args[]){
String s1="Parul";
String s2="University";
System.out.println("string length is: "+s1.length());
System.out.println("string length is: "+s2.length());
}}
```

3. Java String isEmpty()

The **Java String class isEmpty()** method checks if the input string is empty or not. Note that here empty means the number of characters contained in a string is zero.

Syntax

The signature or syntax of string isEmpty() method is given below:

```
public boolean isEmpty()
```

Example

```
public class IsEmptyExample{
public static void main(String args[]){
String s1="";
String s2="University";
System.out.println(s1.isEmpty());
System.out.println(s2.isEmpty());
}}
```

4. Java String replace()

The **Java String class replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

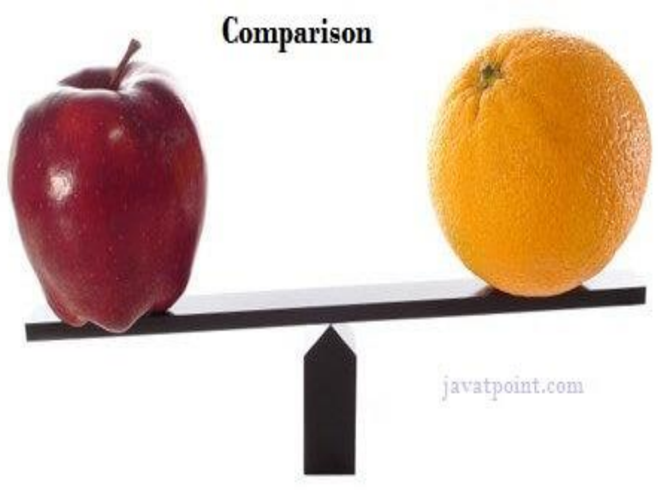
```
public class ReplaceExample1{
public static void main(String args[]){
String s1="W3 school is a very good website";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
}}
```

5. Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
public class SplitExample{
public static void main(String args[]){
String s1="java string split method";
String[] words=s1.split("\\s");//splits the string based on whitespace
//using java foreach loop to print elements of string array
for(String w:words){
System.out.println(w);
}
}}
```

6. Java String compare



We can compare String in Java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

(a) By Using equals() Method

```
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

Example 2

```
class Teststringcomparison2{
public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";
 System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

2) By Using == operator

The == operator compares references not values.

Teststringcomparison3.java

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true (because both refer to same instance)
   System.out.println(s1==s3);//false(because s3 refers to instance created in o
npool)
 }
}
```

3) By Using compareTo() method

The String class 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 objects. If:

- **s1 == s2** : The method returns 0.
- **s1 > s2** : The method returns a positive value.
- **s1 < s2** : The method returns a negative value.

Teststringcomparison4.java

```
class Teststringcomparison4{
 public static void main(String args[]){
   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 )
 }
}
```

String Concatenation in Java

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:

1. By + (String concatenation) operator
2. By concat() method

String Concatenation by + (String concatenation) operator

Java String concatenation operator (+) is used to add strings. For Example:

```
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 }
}
```

String Concatenation by concat() method

```
class TestStringConcatenation3{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);
   System.out.println(s3);//Sachin Tendulkar
  }
}
```

How to reverse String in Java

```
public class TestStringFormatter {
public static void main(String[] args) {
    System.out.println(StringFormatter.reverseString("my name is Ali"));
    System.out.println(StringFormatter.reverseString("I am Arshad"));
    }
}
```

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile & run java package

To Compile: javac -d . Filename.java
To Run: java packageName.Classname

Importing PACKAGES:

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Interfaces in Java

An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.

The interface in Java is *a* mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**.

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the behaviour.
- Interface do not have constructor.

Syntax:

```
interface {

    // declare constant fields

    // declare methods that abstract

    // by default.

}
```

// Java program to demonstrate working of
// interface

```
import java.io.*;

// A simple interface
interface In1 {

    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

// A class that implements the interface.
class TestClass implements In1 {
    // Implementing the capabilities of
    // interface.
    public void display(){
      System.out.println("Geek");
    }

    // Driver Code
    public static void main(String[] args)
    {
        TestClass t = new TestClass();
        t.display();
        System.out.println(a);
    }
}
```

S. No.	**Class**	**Interface**
1.	In class, you can instantiate variables and create an object.	In an interface, you can't instantiate variables and create an object.
2.	Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
3.	The access specifiers used with classes are private, protected, and public.	In Interface only one specifier is used- Public.