1) Introduction

2) Object Class

3) String Class

4) StringBuffer Class

5) StringBuilder Class

Introduction

Object Class

<u>Methods</u>

1) public String toString()
2) public int hashCode()
   - toString() Vs hashCode()
3) public boolean equals(Object obj)
   - Relationship between '==' Operator and .equals()
   - Difference between == Operator and .equals()
   - What is the Main Difference between == Operator and .equals()?
   - Contract between .equals() and hashCode()
4) protected Object clone() throws CloneNotSupportedException
   - Shallow Cloning Vs Deep Cloning
5) protected void finalize() throws Throwable

6) public final Class getClass()

7) public final void wait() throws InterruptedException

8) public final void wait(long ms) throws InterruptedException

9) public final void wait(long ms, int ns) throws InterruptedException

10) public final void notify()

11) public final void notifyAll()

String Class

- Interning of String Objects
- Importance of SCP
- Why SCP Concept is Available Only for String Object but Not for StringBuffer?
- Why String Objects are Immutable where as StringBuffer Objects are Mutable?
- String Class Constructors
- String Class Methods
- Creation of Our Own Immutable Class
- final Vs Immutability

StringBuffer Class

Constructors

Methods

StringBuilder Class

String Vs StringBuffer Vs StringBuilder

Method Chaining

Introduction

- For writing any Java Program whether it is Simple OR Complex, the Most Commonly required Classes and Interfaces are defined in Separate Package which is nothing but java.lang Package.
- We are Not required to Import java.lang Package Explicitly because by Default it is Available to Every Java Program.

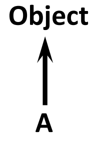
Object Class

- Every Class in a Java is the Child Class of the Object Class Either Directly OR Indirectly.
- So that Object Class Methods by Default Available to Every Java Class.
- Hence Object Class Acts as Root for All Java Classes.

Note:

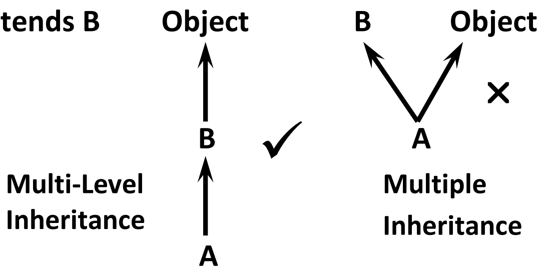
- If our Class won't extends any Other Class then Only it is the Direct Child Class of Object.

```
class A
{
}
```



- If our Class extends any Other Class then it is In - Direct Child Class of Object.

```
class A extends B
{
}
```



Methods:

1) public String toString()

2) public int hashCode()

3) public boolean equals(Object obj)

4) protected Object clone() throws CloneNotSupportedException

5) protected void finalize() throws Throwable

6) public final Class getClass()

7) public final void wait() throws InterruptedException

8) public final void wait(long ms) throws InterruptedException

9) public final void wait(long ms, int ns) throws InterruptedException

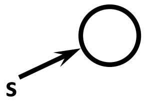
10) public final void notify()

11) public final void notifyAll()

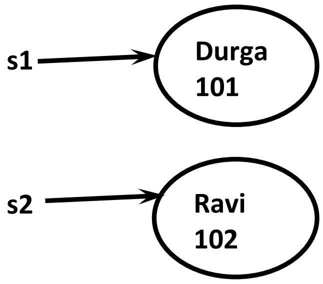
The toString():

- We can Use toString() to get String Representation of an Object.
- Whenever we are trying to Print any Object Reference Internally toString() will be Called.

```
Student s = new Student();
System.out.println(s);  ➡  System.out.println(s.toString());
```



```
class Student {
      String name;
      int rollno;

      Student(String name, int rollno) {
            this.name = name;
            this.rollno = rollno;
      }

      public static void main(String arg[]) {
            Student s1 = new Student ("Durga", 101);
            Student s2 = new Student ("Ravi", 102);
            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s1.toString());
      }
}
```



```
Student@1cb25f1
Student@2808b3
Student@1cb25f1
```

- In the Above Example Object Class toString() got executed which is implemented as follows

```
public String toString() {
          return getClass().getName() + "@" + Integer.toHexString(hashCode());
}

i.e. ClassName@Hexa_Decimal_String_of_hashcode
```

- To Provide Our Own String Representation we have to Override toString().
- For Example whenever we are trying to Print Student Reference to Print his Name and RollNo we can Override toString() as follows.

```
public String toString() {
        return name+"...."+rollno;
        return name;
        return "This is A Student with the Name: " +name+ " and rollno: "+rollno;
}
```

- In All Wrapper Classes, in All Collection Classes, in StringBuffer, StringBuilder toString() is Overridden for Meaningful String Representation.
- It is Highly Recommended to Override toString() in our Classes Also.

```
import java.util.*;
class Test {
    public String toString() {
        return "Test";
    }
    public static void main(String[] args) {

                String s = new String("Durga");
                Integer i = new Integer(10);
                ArrayList l = new ArrayList();

                l.add("A");
                l.add("B");

                HashMap m = new HashMap();
                m.put("A", 101);
                Test t = new Test();

                System.out.println(s); //Durga
                System.out.println(i); //10
                System.out.println(l); //[A, B]
                System.out.println(m); //{A=101}
                System.out.println(t); //Test
        }
}
```

The hashCode():

- For Every Object JVM will generate a Unique Number which is nothing but Hash Code.
- JVM will use Hash Code while Saving Objects into Hashing related Data Structures Like HashSet, Hashtable, HashMap Etc.
- The Main Advantage of saving Objects based on Hash Code is Search Operation will become Easy.
- The Most Powerful Search Algorithm up Today is Hashing.
- If we are Not Overriding hashCode() then Object Class hashCode() will be executed. Which generates hashCode based on Address of an Object. It doesn't Mean Hash Code Represents Address of the Object (It is Impossible to find Address of an Object in Java).
- Based on our Requirement we can Override hashCode() then it is No Longer related to Address.
- Overriding hashCode() is Said to be Proper Way if and Only if for Every Object we have to Generate a Unique Number as hashCode.
- It is Improper Way of Overriding hashCode() because for all Objects we are generating Same Number as Hash Code.
- It is Proper Way of Overriding hashCode() because we are generating a Unique Number as Hash Code.

Improper Way

```
class Student {
    public int hashCode() {
        return 100;
    }
}
```

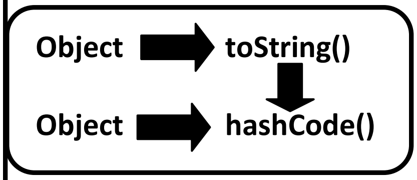
Proper Way

```
class Student {
    public int hashCode() {
        return rollno;
    }
}
```

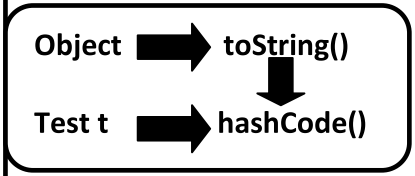
toString() Vs hashCode()

If we are giving the Chance to Object Class toString() then it will Call Internally hashCode().
But whenever we are Overriding toString() Call hashCode().

```
class Test {
    int i;
    Test(int i) {
        this.i = i;
    }
    public static void main(String[] args) {
        Test t1 = new Test(10);
        Test t2 = new Test(100);
        System.out.println(t1); //Test@6e3d60
        System.out.println(t2); //Test@17fa65e
    }
}
```



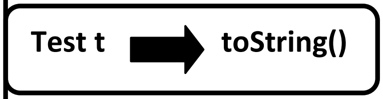
```
class Test {
     int i;
     Test(int i) {   this.i = i;  }
     public int hashCode() {  return i;  }
     public static void main(String[] args) {
            Test t1 = new Test(10);
            Test t2 = new Test(100);
            System.out.println(t1); //Test@a
            System.out.println(t2); //Test@64
      }
}
```



Object → toString()

Test t → hashCode()

```
class Test {
     int i;

     Test(int i) {  this.i = i;            }
     public int hashCode() {  return i;  }
     public String toString() {  return i + "";  }

     public static void main(String[] args) {
            Test t1 = new Test(10);
            Test t2 = new Test(100);
            System.out.println(t1); //10
            System.out.println(t2); //100
            }

}
```



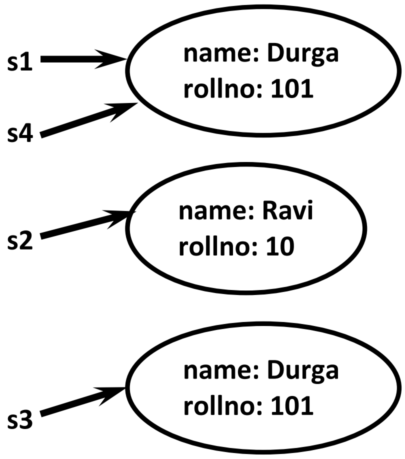
Test t → toString()

Note: The Above 3 Programs Shows Link between toString() and hashCode().

The equals():

- We can Use equals() to Check Equaliy of 2 Ojects.
- If our Class doesn’t contain equals() then Object Class equals() will be executed. Which is meant for Reference Comparison (Address Comparison) i.e. r1.equals(r2); Returns true if and only if both r1 and r2 pointing to the Same Object.

```
class Student {
    String name;
    int rollno;

    Student(String name, int rollno) {
        this.name = name;
        this.rollno = rollno;
    }

    public static void main(String[] args) {
        Student s1 = new Student("Durga", 101);
        Student s2 = new Student("Ravi", 102);
        Student s3 = new Student("Durga", 101);
        Student s4 = s1;
        System.out.println(s1.equals(s2)); //false
        System.out.println(s1.equals(s3)); //false
        System.out.println(s1.equals(s4)); //true
   }

}
```



- In the Above Example Object Class equals() got executed which is meant for Reference Comparison.
- Based on our Requirement we can Override equals() in our Class for Content Comparison.

While Overriding equals() we have to Consider the following things.

- What is the Meaning of Content Comparison. For Example Whether we have to Check only Names OR only rollnos OR Both.
- If we Pass different Type of Objects our equals() should Return false but not java.lang.ClassCastException i.e Whether to Handle ClassCastException to Return false.
- If we Pass null Argument our equals() should Return false but not java.lang.NullPointerException i.e. we have to Handle NullPointerException to Return false.

The following is the Valid Way of Overriding equals() for Content Comparison in Student Class.

- If 2 Students having the Same Name and Same rollno our equals() should Return true.

```
class Student {
    String name;
    int rollno;

    Student(String name, int rollno) {
        this.name = name;
        this.rollno = rollno;
    }

    public boolean equals(Object obj) {
        try {
                String name1 = this.name;
                int rollno1 = this.rollno;
                Student s2 = (Student)obj;
                String name2 = s2.name;
                int rollno2 = s2.rollno;
                if(name1.equals(name2) && rollno1 == rollno2) {  return true;  }
                else {  return false;  }
            }
            catch (ClassCastException c) {  return false;  }
            catch (NullPointerException e) {  return false;  }
    }
    public static void main(String arg[]) {

          Student s1 = new Student ("Durga", 101);
          Student s2 = new Student ("Ravi", 102);
          Student s3 = new Student ("Durga", 101);
          Student s4 = s1;

          System.out.println(s1.equals(s2)); //false
          System.out.println(s1.equals(s3)); //true
          System.out.println(s1.equals(s4)); //true
          System.out.println(s1.equals("Durga")); //false
          System.out.println(s1.equals(null)); //false
   }

}
```

Simplified Version of equals()

```
public boolean equals(Object obj) {
      try {
                Student s = (Student)obj;
                if(name.equals(s.name) && rollno == s.rollno) {
                           return true;
                }
                 else {
                            return false;
                 }
      }
      catch (ClassCastException c) {  return false;  }
      catch (NullPointerException e) {  return false;  }
}
```

More Simplified Version of equals()

```
public boolean equals(Object obj) {
        if (obj instanceof Student) {
             Student s = (Student)obj;
             if(name.equals(name) && rollno == s.rollno) {
                              return true;
              }
             else {  return false;  }
       }
       return false;
}
```

Note: To Make Our equals() More Efficient we have to Take the following Code at the beginning of equals().

```
if (obj == this)
          return true;
```

i.e if 2 References pointing to the Same Object then our equals() should Return true without performing any Comaprision.

Eg:
```
String s1 = new String("Durga");
String s2 = new String("Durga");

StringBuffer sb1 = new StringBuffer("Durga");
StringBuffer sb2 = new StringBuffer("Durga");

System.out.println(s1.equals(s2)); //true
System.out.println(sb1.equals(sb2)); //false
System.out.println(s1.equals(sb1)); //false
```

- If String Class equals() is Overridden for Content Comparison but in StringBuffer Class equals() is not Overridden for Content Comparison Hence Object Class equals() will be executed which is meant for Reference Comparison.

Note: In All Wrapper Classes, in All Collection Classes, in String Class equals() is Overridden for Content Comparison. Hence it is Highly Recommended to Override equals() in our Class Also.

Relationship between '==' Operator and .equals()

- If r1 == r2 Return true then r1.equals(r2) is always true i.e if 2 Objects are Equal by == Operator then these Objects are always Equal by .equals() also.
- If r1 == r2 Returns false then r1.equals(r2) then we can't conclude anything about equals(). It may Returns true OR false.
- If r1.equals(r2) is true then we can't conclude anything about == Operator. It may Returns true OR false.
- If r1.equals(r2) is false then r1 == r2 is always false.

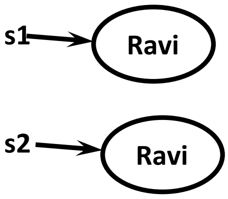
Difference between == Operator and .equals()

== Operator	.equals()
It is an Operator applicable for Primitives and Object References.	It is a Method applicable only for Object References.
In the Case of Object References == Operator always meant for Reference Comparison.	By Default equals() Present in Object Class also meant for Reference Comparison.
We can't Override == Operator for Content Comparison.	We can Override equals() for Content Comparison.
To use == Operator Compulsory there should be Some Relation between Argument Types Otherwise we will get CE: incompatible types.	If there is No Relation between Argument Type we won't get any CE/ RE. equals() Simply Returns false.
For Any Object References r, r == null Returns false.	For Any Object References r, r.equals(null) Returns false.

What is the Main Difference between == Operator and .equals()?

In General we can use .equals() for Content Comparison whereas == Operator for Reference Comparison.

```
String s1 = new String("Ravi");
String s2 = new String("Ravi");
System.out.println(s1.equals(s2)); //true
System.out.println(s1 == s2); //false
```



Contract between .equals() and hashCode()

2 Equivalent Objects should be placed in the Same Bucket but all Objects Present in the Same Bucket Need not be Equal.

- If 2 Objects are Equal by .equals() then Compulsory their hashCodes must be Same i.e. 2 Equivalent Objects should have Same hashCode. i.e. if *r1.equals(r2)* is true then
  r1.hashCode() == *r2.hashCode()* should be true.
- If 2 Objects are not Equal by .equals() then No Restriction on hashCodes may be Same OR may not be Same.
- If Hash Codes of 2 Objects are Equal we can't conclude anything about .equals() it may Returns true OR false.
- If Hash Codes of 2 Objects are not Equal then these Objects are always not Equal by .equals() Also.

To Specify Above Contract between equals() and hashCode() whenever we are Overriding equals() Compulsory we should Override hashCode() otherwise we won't get CE/ RE but it is Not a Good Programming Practice.

Eg: In String Class equals() is Overridden for Content Comparison hence hashCode() is also Overridden to generate Hash Code based on Content.

```
String s1 = new String("Durga");
String s2 = new String("Durga");
System.out.println(s1.equals(s2)); //true
System.out.println(s1.hashCode()); //2539842
System.out.println(s2.hashCode()); //2539842
```

Consider the following Person Class:

```
public boolean equals(Object obj)
{
            if(obj instanceof Person)
            {
                        Person p = (Person)obj;
                        if(name.equals(p.name) && age == p.age) {  return true;  }
                        else {
                           return false;
                        }
            }
           return false;
 }
```

Which of the following hashCode() is Appropriate for Person Class?

- public in hashCode() { return 100; }
- public int hashCode() { return name.length()+height; }
- public int hashCode() { return name.hashCode()+age; } √
- No Restriction

Note: Based on which Parameters we Override equals() use the Same Parameters by Overriding hashCode() also so that we can maintain Contract between equals() and hashCode().

The clone():

- The Process of Creating exactly Duplicate Object is known as Cloning.
- The Main Purpose of Cloning is to maintain Backup and to Preserve Initial State of an Object.
- We can cloned an Object by using clone() of Object Class.
  protected native Object clone() throws CloneNotSupportedException

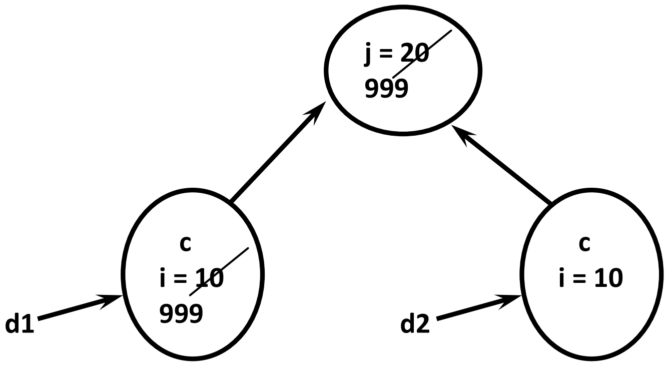
- To Perform Cloning Compulsory the Object should be Cloneable Object.
- An Object is said to be Cloneable if and only if the Corresponding Class implements Cloneable Interface. It is a *Marker Interface* because it doesn't contain any Methods.
- If we are trying to Clone a Non-Cloneable Object we will get a RE: CloneNotSupportedException

Shallow Cloning Vs Deep Cloning

Shallow Cloning

- The Process of creating Bit Wise Copy of an Object is called Shallow Cloning.
- If the Main Object contain any Primitive Variables exactly Duplicate Copy of those Variables will be created in cloned Object.
- If the Main Object contain any Referenced Variable then the Corresponding Object won't be Created Just Duplicate Reference Variable will be Created by pointing to Old contained Object.
- By using Main Object Reference if we Perform any Change to the contained Object then those Changes will be reflected to the Cloned Object.
- Object Class clone() by Default meant for Shallow Cloning.

```
class Cat {
        int j;
        Cat(int j) {  this.j = j;  }
}
class Dog implements Cloneable {
        Cat c;
        int i;
        Dog(Cat c, int i) {
                this.c = c;
                this.i = i;
        }
        public Object clone() throws CloneNotSupportedException {
                 return super.clone();
        }
}
class ShallowCloning {
        public static void main(String[] args) throws CloneNotSupportedException {
                Cat c = new Cat(20);
                Dog d1 = new Dog(c, 10);
                System.out.println(d1.i+"....."+d1.c.j); //10.....20
                Dog d2 = (Dog)d1.clone();
                d1.i = 888;
                d1.c.j = 999;
                System.out.println(d2.i+"....."+d2.c.j); //10.....999
        }
}
```

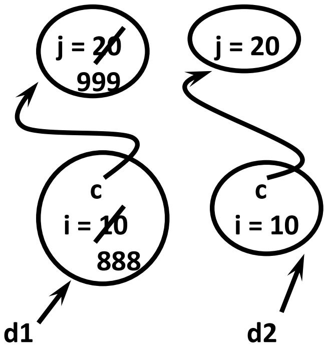


- Shallow Cloning is the Best Choice if the Object contains Only Primitive Values.
- In Shallow Cloning by using Main Object Reference if we perform any Changes to the contained Object then those Changes will be reflected to the cloned Copy.
- To overcome this Problem we should go for Deep Cloning.

Deep Cloning:

- The Process of creating exactly Independent Duplicate Object (including contained Objects also) is called Deep Cloning.
- In Deep Cloning if Main Object contain any Reference Variable then corresponding Object also will be created in the cloned Copy.
- By Default Object Class clone() meant for Shallow Cloning if we want Deep Cloning then Programmer is Responsible to implement that by Overriding clone().

```
class Cat {
        int j;
        Cat(int j) {  this.j = j;  }
}
class Dog implements Cloneable {
        Cat c;
        int i;
        Dog(Cat c, int i) {
                this.c = c;
                this.i = i;
        }
        public Object clone() throws CloneNotSupportedException {
                Cat c1 = new Cat(c.j);
                Dog d = new Dog(c1, i);
                return d;
        }
}
class DeepCloning {
        public static void main(String[] args) throws CloneNotSupportedException {
                Cat c = new Cat(20);
                Dog d1 = new Dog(c, 10);
                System.out.println(d1.i+"....."+d1.c.j); //10.....20
                Dog d2 = (Dog)d1.clone();
                d1.i = 888;
                d1.c.j = 999;
                System.out.println(d2.i+"....."+d2.c.j); //10.....20
        }
}
```



- In Deep Cloning by using Main Object Reference if we perform any Change to the contained Object then those Changes won’t be reflected to the cloned Objects.

Which Cloning is the Best?

- If the Object contains Only Primitive Variables then Shallow Cloning is the Best Choice.
- If the Object contains Referenced Variables then Deep Cloning is the Best Choice.

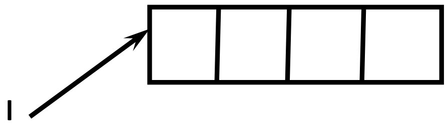
getClass():

We can Use getClass() to get Runtime Class Definition of an Object.

Eg:

```
Object o = l.get(o);
System.out.println(o.getClass().getName());
```

$\underbrace{\texttt{o.getClass()}}_{\text{Class Object}}$



We can Use this Method Very Commonly in Reflections.

finalize():

- Just before destroying an Object Garbage Collector calls this Method for Clean-Up Activities.
- Once finalize() completes Automatically Garbage Collector destroys that Object.

wait(), notify() and notifyAll(): We can use all these Methods in Multi-Threading for Inter Thread Communication.

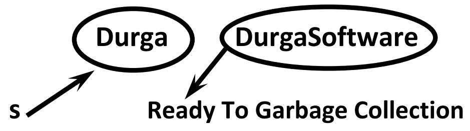
String

Difference between *String* and *StringBuffer* Objects?

Case 1

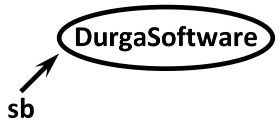
```
String s = new String("Durga");
s.concat("Software");
System.out.println(s); //Durga
```

Once we created a String Object we can't Perform any changes in the existing Object. If we are trying to Perform any changes with those changes a New Object will be created. This non-changeable Nature is nothing but Immutability.



```
StringBuffer sb = new StringBuffer("Durga");
sb.append("Softwate");
System.out.println(sb); //DurgaSoftware
```

Once we created a StringBuffer Object we can Perform any Type of changes in the existing Object. This Changeable Nature is nothing but Mutability of StringBuffer Objects.



Case 2

```
String s1 = new String("Durga");
String s2 = new String("Durga");
System.out.println(s1 == s2); //false
System.out.println(s1.equals(s2)); //true
```

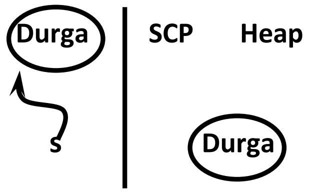
In String Class .equals() is Overridden for Content Comparison Hence if the Content is Same .equals() Returns True Even though Objects are Different.

```
StringBuffer sb1 = new StringBuffer("Durga");
StringBuffer sb2 = new StringBuffer ("Durga");
System.out.println(sb1 == sb2); //false
System.out.println(sb1.equals(sb2)); //false
```

In StringBuffer Class .equals() is not Overridden for Content Comparison Hence Object Class .equals() will be executed which is meant for Reference Comparison. Due to this if Objects are different equals() Returns false Event though Content is Same.

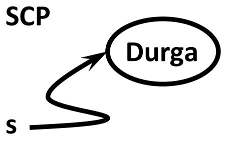
Case 3: Difference between *String s = new String("Durga");* and *String s = "Durga";*

- String s = new String("Durga");
  In this Case 2 Objects will be Created, One is in the Heap and the Other is in String Constant Pool (SCP) and S is always pointing to Heap Object.



- **String s = "Ravi";**

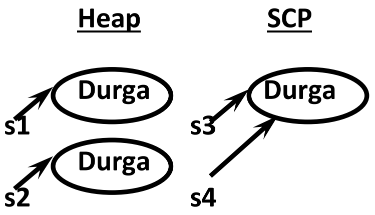
  In this Case only one Object will be created in the SCP and S is always refers to that Object.



Note:

- Garbage Collector is Not allowed to Access SCP Area Hence Even though Object Not having the Reference Still it is Not Eligible for Garbage Collector if it is Present in SCP Area.
- At the Time of JVM Shutdown all SCP Objects will be destroyed automatically.
- Object Creation in SCP is always Optional. First JVM will check is any Object already Present in SCP with required Content if it is already Present then existing Object will be re-used. If it is not already Present then only a New Object will be created.

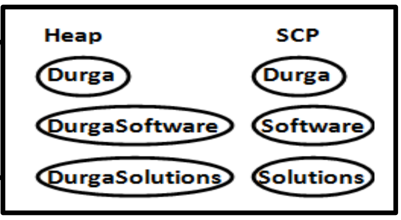
```
String s1 = new String("Durga");
String s2 = new String("Durga");

String s3 = "Durga";
String s4 = "Durga";
```



Note:

- Whenever we are using New Operator Compulsory a New Object will be created in the Heap Area.
- There may be a Chance of existing 2 Objects with Same Content on the Heap but No Chance of existing 2 Objects with the Same Content in SCP Area i.e. Duplicate Objects are Possible in Heap but there is No Chance of existing Duplicate Objects in SCP.

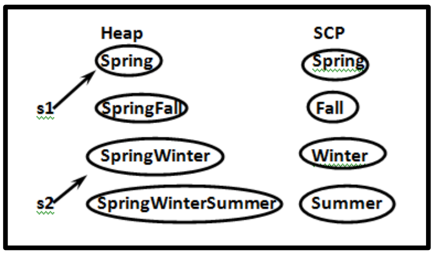
```
String s = new String("Durga");
s.concat("Software");
s = s.concat("Solutions");
```



Note:

- For every String Constant one Object will be created on the SCP.
- Because of Some Runtime Operation like Method Call if an Object is required to create that Object will be created only in Heap but not in SCP.

```
String s1 = new String("Spring");
s1.concat("Fall");
String s2 = s1.concat("Winter");
s2.concat("Summer");
System.out.println(s1); //Spring
System.out.println(s2); //SpringWinter
```

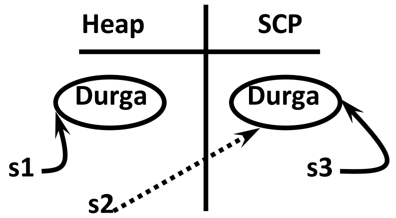


```
class StringTest {
	public static void main(String[] args) {

		String s1 = new String("You cannot Change Me");
		String s2 = new String("You cannot Change Me");
		System.out.println(s1 == s2); //false

		String s3 = "You cannot Change Me";
		System.out.println(s1 == s3); //false

		String s4 = "You cannot Change Me";
		System.out.println(s3 == s4); //true

		String s5 = "You cannot"+" Change Me";
		System.out.println(s4 == s5); //true

		String s6 = "You cannot";
		String s7 = s6 + " Change Me";
		System.out.println(s4 == s7); //false

		final String s8 = "You cannot";
		String s9 = s8 + " Change Me";
		System.out.println(s4 == s9); //true

	}
}
```

Interning of String Objects:

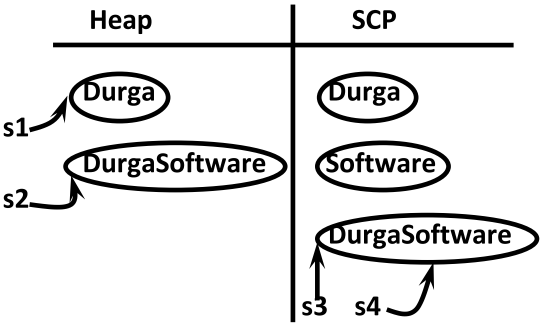
By Using Heap Object Reference if we want to get corresponding SCP Object Reference then we should go for intern().

```
class StringTest {
	public static void main(String[] args) {
		String s1 = new String("Durga");
		String s2 = s1.intern();
		System.out.println(s1 == s2); //false
		String s3 = "Durga";
		System.out.println(s2 == s3); //true
	}
}
```



If the corresponding SCP Object is Not Available then intern() will Create that Object in SCP and Returns it.

```
String s1 = new String("Durga");

String s2 = s1.concat("Software");
String s3 = s2.intern();

String s4 = "DurgaSoftware";

System.out.println(s3 == s4); //true
```



Importance of SCP:

- In Our Program if any String Objects required repeatedly then it is not recommended to create a Separate Object for every Requirement because it reduces Memory Utilization and Creates Performance Problems.
- Instead of creating a Separate Object for every Requirement we can Create a Single Object and we can Reuse the Same Object for every Similar Requirement so that Memory Utilization and Performance will be improved. We can achieve this by using SCP Concept. Hence the Main Advantage of SCP is Memory Utilization and Performance will be improved.
- The Main Disadvantage of SCP is in SCP Several References pointing to the Same Object. By using One Reference if we are allowed to Change the Content then the remaining References will be impacted. To overcome this Problem SUN People implemented String Objects as Immutable. According to this once we created a String Object we can’t perform any Changes in the existing Objects. If we are trying to Change with those Changes a New Object will be created.
- SCP is a specially designed Memory Area for the String Constants.

Why SCP Concept is Available Only for String Object but Not for StringBuffer?

- String Objects are Most Commonly used Objects in Java Hence SUN People provided a Special Memory Management for the String Objects.
- StringBuffer Objects are not commonly used Objects Hence SUN People won’t provide any Special Memory Area for the StringBuffer Objects.

Why String Objects are Immutable where as StringBuffer Objects are Mutable?

- In the Case of String Objects Just Because of SCP A Single Object can be referred by Multiple References.
- By using One Reference if we are allowed to Change the Content then remaining References will be impacted. To overcome this Problem SUN People implemented String Objects as Immutable.
- According to this once we created a String Object we can’t perform any Changes in the existing Object. If we are trying to Change with those Changes a New Object will be created.

- But in StringBuffer there is no Concept like SCP. Hence for every Requirement a New Object will be created by using One Reference. If we are changing the Content in existing Object there is No Effect on remaining References. Hence Immutability Concept not required for StringBuffer Objects.

Note: SCP is the Only Reason why String Objects are Immutable.

In Addition to String Objects any Other Objects are Immutable in Java?
All Wrapper Class Objects also Immutable.

String Class Constructors:

1) String s = new String(); Creates an Empty String Object.

2) String s = new String(String literal); Creates an Equivalent String Object for the given String Literal.

3) String s = new String(StringBuffer sb); Creates an Equivalent String Object for the given StringBuffer.

4) String s = new String(char [] ch); Creates an Equivalent String Object for the given char[].

Eg:

```
char[] ch = {‘a’, ’b’, ’c’, ’d’};
String s = new String(ch);
System.out.println(s); //abcd
```

5) String s = new String(byte[] b); Creates an Equivalent String Object for the given byte[].

Eg:

```
byte[] b = {100, 101, 102, 103, 104};
String s = new String(b);
System.out.println(s); //defgh
```

Important Methods of String Class

1) public char charAt(int index); Returns the char locating at specified Index.

Eg:

```
String s = “Durga”;
System.out.println(s.chatAt(3)); //g
System.out.println(s.charAt(30));  //java.lang.StringIndexOutOfBoundsException:
                                        String index out of range: 30
```

2) public String concat(String s):
The Overloaded ‘+’ and ‘+=’ Operators Acts as Concatenation Operation Only.

Eg:

```
String s = "Durga";
s = s.concat("Software");
//s = s+"Software"; OR s += "Software";
System.out.println(s); //DurgaSoftware
```

3) public boolean equals(Object o)
To Perform Content Comparison where Case is Important.
This is Overriding Version of Object Class equals().

4) public boolean equalsIgnoreCase(String s);
To Perform Content Comparison where Case is Not Important.

Eg:

```
String s = "Java";
System.out.println(s.equals(“JAVA”); //false
System.out.println(s.equalsIgnoreCase("JAVA")); //true
```

Note: In General we can use *equalsIgnoreCase()* to Compare User Names where Case is Not Important, whereas we can use *equals()* to Compare Passwords where Case is Important.

5) public int length(); Return Number of Characters Present in the String.

Eg:

```
String s = "Java";
System.out.println(s.length()); //4
System.out.println(s.length); //CE: cannot find symbol
                                    symbol:   variable length
                                    location: variable s of type String
```

Note: length is the Variable applicable for Array Objects where as length() is the Method applicable for String Objects.

6) public String replace(char old, char new);

Eg:

```
String s = "ababa";
System.out.println(s.replace('a','b')); //bbbbb
```

7) public String substring(int begin);
Returns Sub String from Begin Index to End of the String.

8) public String substring(int begin, int end);
Returns the Characters from Begin Index to End-1 Index.

Eg:

```
String s = "abcdefg";
System.out.println(s.substring(3)); //defg
System.out.println(s.substring(2, 5)); //cde
```

9) public String toLowerCase();

10) public String toUpperCase();

11) public String trim(); To Remove Blank Spaces Present at Starting and Ending of the Sting. But not Middle Blank Spaces.

12) public int indexOf(char ch);
Returns the Index of First Occurance of specified Character.

13) public int lastIndexOf(char ch);

Returns the Index of Last Occurance of specified Character.

Eg:

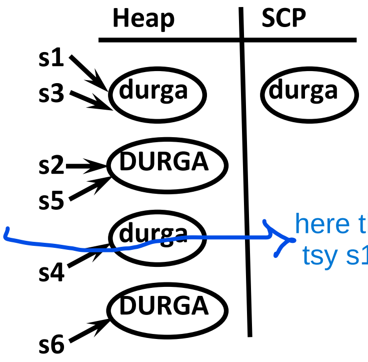
```
String s = “ababa”;
System.out.println(s.indexOf(‘a’)); //0
System.out.println(s.lastIndexOf(‘a’));  //4
```

Note:

- Once We Created A String Object We Can't Perform Any Changes In The Existing Object.
- If We Are Trying To Perform Any Operation And If There Is A Change In The Content With Those Changes A New Object Will Be Created.
- With Our Operation If There Is No Change In The Content Then New Object Won't Be Created Existing Object Will Be Reused. This Rule Is Same Whether The Object Present In Heap Or SCP.

```
String s1 = new String("durga");

String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();

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

String s4 = s2.toLowerCase();
String s5 = s2.toUpperCase();
String s6 = s4.toUpperCase();
```

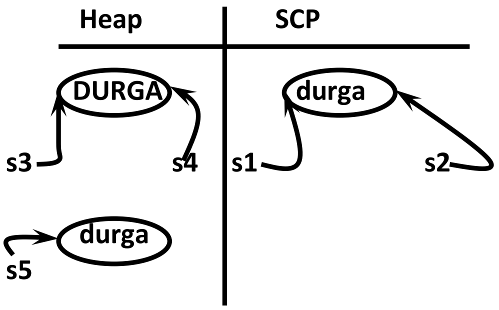
true



here three is no change in original object tsy s1==s3 // true

here original object s2 changes thats why s4 == s1 false

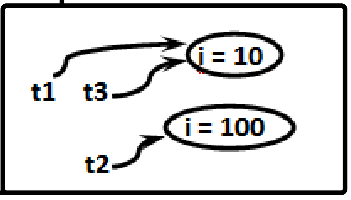
```
String s1 = "durga";

String s2 = s1.toLowerCase();
String s3 = s1.toUpperCase();

String s4 = s3.toString();
String s5 = s3.toLowerCase();
```



Creation of Our Own Immutable Class:

- Once we created an Object we can’t perform any Changes in the existing Object.
- If we are trying to perform any Changes with those Changes a New Object will be created.
- If there is No Change in the Content then New Object won’t be created and existing Object will be reused.

```
final class Test {
        private int i;
        Test(int i) {
             this.i = i;
        }
        public Test modify(int i) {
                if(this.i == i)
                     return this;
                else
                     return (new Test(i));
        }
}

Test t1 = new Test(10);
Test t2 = t1.modify(100);
Test t3 = t1.modify(10);
System.out.println(t1 == t2); //false
System.out.println(t1 == t3); //true
```

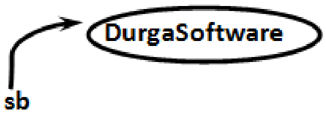


Once we created a Test Object we can't perform any Changes in the existing Object. If we are trying to perform any Changes with those Changes a New Object will be created. Hence Test Class Objects are Immutable.

final Vs Immutability:

- final Keyword applicable for Variables but not for Objects whereas Immutability Concept is applicable for Objects but not for Variables.
- final and Immutability both are different Concepts.
- By declaring a Reference Variable as final we won't get any Immutability Nature but we can't perform re-assignment for that Reference Variable.

```
class Test {
    public static void main(String[] args) {
          final StringBuffer sb = new StringBuffer("Durga");
          sb.append("Software");
          System.out.println(sb); //true
          sb = new StringBuffer("Solutions"); //CE: cannot assign a value to final variable sb
    }
}
```



Note:

- If the Content is Not Fixed and keep on changing then it is Never recommend to go for String because for every Change a New Object will be created Internally.
- To Handle this Requirement we should go for StringBuffer Class.
- The Main Advantage of StringBuffer over String is all required Changes will be performed in the existing Object only Instead of creating a New Object.

Which of the following is Meaningful?

1) final variable √
2) final object ✕

3) Immutable variable ✕
4) Immutable object √

StringBuffer

Constructors:

1) StringBuffer sb = new StringBuffer();
Creates an Empty StringBuffer Object with Default Initial Capacity 16.

Eg:

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity()); //16

sb.append("abcdefghijklmnop");
System.out.println(sb.capacity()); //16

sb.append("q");
System.out.println(sb.capacity()); //34
```

Once StringBuffer reaches Max Capacity then a New StringBuffer Object will be created with *new capacity = (currentcapacity + 1) * 2*

2) StringBuffer sb = new StringBuffer(int initialCapacity);
Creates an Empty StringBuffer Object with the specified Initial Capacity.

3) StringBuffer sb = new StringBuffer(String s);
Creates an Equivalent StringBuffer Object for the given String Object with
capacity = s.length()+16

Eg:

```
StringBuffer sb = new StringBuffer(“Durga”);
System.out.println(sb.capacity()); // 5 + 16 = 21
```

Methods

1) public int length();

2) public int capacity();

3) public char charAt(int index);

Eg:

```
StringBuffer sb = new StringBuffer("Durga");
System.out.println(sb.charAt(3)); //g
System.out.println(sb.charAt(30)); //RE: java.lang.StringIndexOutOfBoundsException:
                                         String index out of range: 30
```

4) public void setCharAt(int index, char ch);
Replaces the Character Present at specified Index with the provided Character.

5) public StringBuffer append(String s);
public StringBuffer append(int i);
public StringBuffer append(float f);
public StringBuffer append(double d);
public StringBuffer append(boolean b);
public StringBuffer append(Object o);
::
::

Overloaded Methods

Eg:

```
StringBuffer sb = new StringBuffer();
sb.append("PI Value is: ");
sb.append(3.14);
sb.append(" It is Exactly: ");
sb.append(true);
System.out.println(sb); //PI Value is: 3.14 It is Exactly: true
```

6) public StringBuffer insert(int index, String s);
public StringBuffer insert(int index, int i);
public StringBuffer insert(int index, float f);
public StringBuffer insert(int index, double d);
public StringBuffer insert(int index, boolean b);
public StringBuffer insert(int index, Object o);
;;;

Overloaded Methods

Eg:

```
StringBuffer sb = new StringBuffer("abcdefgh");
sb.insert(2, "xyz");
System.out.println(sb); //abxyzcdefgh
```

7) public StringBuffer delete(int begin, int end);
To Delete Characters locating from Begin Index to End-1 Index.

8) public StringBuffer deleteCharAt(int index);

9) public StringBuffer reverse();

10) public void setLength(int length);

```
StringBuffer sb = new StringBuffer("AishwaryaAbhi");
sb.setLength(8);
System.out.println(sb); //Aishwarya
```

11) public void ensureCapascty(int capacity);
To Increase Capacity on Fly based on our Requirement.

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity()); //16
sb.ensureCapacity(1000);
System.out.println(sb.capacity()); //1000
```

12) public void trimToSize();

To De-allocate Extra allocated Free Memory.

```
StringBuffer sb = new StringBuffer(1000);
sb.append("abc");
sb.trimToSize();
System.out.println(sb.capacity()); //3
```

Note:

- Every Method Present in StringBuffer is synchronized.
- Hence at a Time Only One Thread is allowed to operate on StringBuffer Object.
- It increases waiting Time of Threads and Creates Performance Problems.
- To Overcome this Problem SUN People introduced StringBuilder Class in 1.5 Version.

StringBuilder

StringBuilder is exactly Same as StringBuffer (Including Constructors and Methods Also) Except the following Differences.

StringBuffer	StringBuilder
Every Method Present In Stringbuffer Is Synchronized.	No Method Present In Stringbuilder Is Synchronized.
At A Time Only One Thread Is Allow To Operate On Stringbuffer Object And Hence It Is Thread Safe.	At A Time Multiple Thread Are Allowed To Operate On Stringbuilder Object And Hence It Is Not Thread Safe.
Threads Are Required To Wait To Operate On Stringbuffer Object And Hence Relatively Performance Is Slow.	Threads Are Not Required To Wait To Operate On Stringbuilder Object And Hence Relatively Performance Is High.
Introduced In 1.0 Version.	Introduced In 1.5 Version.

String Vs StringBuffer Vs StringBuilder

- If the Content is Fixed and won't changed frequently then we should go for String.
- If the Content is Not Fixed and keep on changing but Thread Safety is required then we should go for StringBuffer.
- If the Content is Not Fixed and keep on changing but Thread Safety is Not required then we should go for StringBuilder.

Method Chaining:

- For most of the Methods in String, StringBuffer and StringBuilder the Return Types are the Same Type (String, StringBuffer and StringBuilder Objects) only.
- Hence After Applying a Method on the Result we can Call Another Method which forms a Method Chaining. *sb.m1().m2().m3().m4().m5()…..;*
- All these Method Calls will execute from Left to Right.

```
StringBuffer sb = new StringBuffer();

sb.append("Durga")
   .append("Software") // DurgaSoftware
   .append("Solutions") // DurgaSoftwareSolutions
   .insert(2, "xyz") // DuxyzrgaSoftwareSolutions
   . delete(7, 15) // DuxyzrgeSolutions
   .reverse() // snoituloSegrzyxuD
   .append("Hyd");  //snoituloSegrzyxuDHyd
    System.out.println(sb); //snoituloSegrzyxuDHyd
```

Practice Questions for String and StringBuilder

Q1. Given the code fragment:

```
public class Test
{
   public static void main(String[] args)
   {
      StringBuilder sb=new StringBuilder(5);
      String s="";
      if(sb.equals(s))
      {
         System.out.println("Match 1");
      }
      else if(sb.toString().equals(s.toString()))
      {
         System.out.println("Match 2");
      }
      else
      {
         System.out.println("No Match");
      }
   }
}
```

StringBuilder equals is not for content comparison

What is the result?

A. Match 1
B. Match 2
C. No Match
D. NullPointerException is thrown at runtime

Answer: B

Q2. Given:

```
public class Test
{
   public static void main(String[] args) {
      String ta="A";
      ta=ta.concat("B");
      String tb="C";
      ta=ta.concat(tb);
      ta.replace('C','D');
      ta=ta.concat(tb);
      System.out.println(ta);
   }
}
```

What is the result?

A. ABCD
B. ACD
C. ABCC
D. ABD
E. ABDC

Answer: C

Q3. Given the code fragment:

```
public class Test
{
   public static void main(String[] args)
   {
      StringBuilder sb1= new StringBuilder("Durga");
      String str1=sb1.toString();
      //insert code here==>Line-1
      System.out.println(str1==str2);
   }
}
```

Which code fragment,when inserted at Line-1,enables the code to print true?

A. String str2=str1;
B. String str2=new String(str1);
C. String str2=sb1.toString();
D. String str2="Durga";

Answer: A

Q4. You are developing a banking module. You have developed a class named ccMask that has a maskcc method.
Given the code fragment:

```
class CCMask
{
   public static String maskCC(String creditCard)
   {
      String x="XXXX-XXXX-XXXX-";
      //Line-1
   }
   public static void main(String[] args)
   {
      System.out.println(maskCC("1234-5678-9101-1121"));
   }
}
```

You must ensure that maskCC method returns a String that hides all digits of the credit card number except last four digits(and the hyphens that seperate each group of 4 digits)

Which two code fragments should you use at line 1, independently to achieve the requirement?

A) StringBuilder sb=new StringBuilder(creditCard);
   sb.substring(15,19);
   return x+sb;

B) return x+creditCard.substring(15,19);

C) StringBuilder sb=new StringBuilder(x);
   sb.append(creditCard,15,19);
   return sb.toString();

D) StringBuilder sb=new StringBuilder(creditCard);
   StringBuilder s=sb.insert(0,x);
   return s.toString();

Answer: B,C

Q5. Consider the following code

```
public class Test
{
   public static void main(String[] args)
   {
      String str=" ";
      str.trim();
      System.out.println(str.equals("")+" "+str.isEmpty());
   }
}
```

What is the result?

A. true false
B. true true
C. false true
D. false false

Answer: D

Q6. Consider the following code:

```java
public class Test
{
    public static void main(String[] args)
    {
        String s1="Java";
        String s2= new String("java");
        //Line-1
        {
            System.out.println("Equal");
        }
        else
        {
            System.out.println("Not Equal");
        }
    }
}
```

To print "Equal" which code fragment should be inserted at Line-1

A. String s3=s2;
   if(s1==s3)

B. if(s1.equalsIgnoreCase(s2))

C. String s3=s2;
   if(s1.equals(s3))

D. if(s1.toLowerCase()==s2.toLowerCase())

Answer: B

Q7. Given the following code:

```java
class MyString
{
    String msg;
    MyString(String msg)
    {
        this.msg=msg;
    }
}
public class Test
{
    public static void main(String[] args)
    {
      System.out.println("Hello "+ new StringBuilder("Java SE 8"));
      System.out.println("Hello "+ new MyString("Java SE 8"));
   }
}
```

What is the result?

A) Hello Java SE 8
   Hello MyString@<hashcode>

B) Hello Java SE 8
   Hello Java SE 8

C) Hello java.lang.StringBuilder@<hashcode>
   Hello MyString@<hashcode>

D) Compilation Fails

Answer: A

Q8. Given:

```
public class Test
{
   public static void main(String[] args)
   {
      String s="Java Duke";
      int len=s.trim().length();
      System.out.println(len);
   }
}
```

What is the result?

A. 9
B. 8
C. 10
D. Compilation Fails

Answer: A

Q9. Which statement will empty the contents of a StringBuilder variable named sb?

A. sb.deleteAll();
B. sb.delete(0,sb.size());
C. sb.delete(0,sb.length());
D. sb.removeAll();

Answer: C

Q10. Given

```
public class Test
{
   public static void main(String[] args)
   {
      String s="Hello World";
      s.trim();
      int i1=s.indexOf(" ");
      System.out.println(i1);
   }
}
```

What is the result?

A. An exception is thrown at runtime
B. -1
C. 5
D. 0

Answer: C