Aggregation (Weak Referance) :

Aggregation in Java is another form of association between classes that represents a "HAS-A" relationship, but with a weaker bond compared to composition.

In aggregation, one class contains an object of another class, but the contained object can exist independently of the container. If the container object is destroyed, the contained object can still exist.

```
package com.ravi.aggregation;

public class College
{
    private String collegeName;
    private String collegeLocation;

    public College(String collegeName, String collegeLocation)
    {
        super();
        this.collegeName = collegeName;
        this.collegeLocation = collegeLocation;
    }

    @Override
    public String toString()
    {
        return "College [collegeName=" + collegeName + ", collegeLocation=" + collegeLocation
+ "]";
    }


}
```

```
package com.ravi.aggregation;

public class Student
{
    private int studentId;
    private String studentName;
    private College college; // HAS- A relation

    public Student(int studentId, String studentName, College college)
    {
        super();
        this.studentId = studentId;
        this.studentName = studentName;
        this.college = college;
    }

    @Override
    public String toString()
    {
        return "Student [studentId=" + studentId + ", studentName=" + studentName + ",
college=" + college + "]";
    }
}
```

```
package com.ravi.aggregation;

public class AggregationDemo {

    public static void main(String[] args)
    {
        College c1 = new College("VIT", "Vellore");

        c1  = new College("NIT", "Hyderabad");

        Student s1 = new Student(101, "Scott", c1);
        System.out.println(s1);

        Student s2 = new Student(102, "Smith", c1);
        System.out.println(s2);
    }
}
```

How System.out.println() statement works internally ?
---

* System is a predefined final class available in java.lang package. It contains private constructor.

```
 public final class System
 {
    public static final java.io.PrintStream  out;   //HAS-A relation

    private System()
    {
    }

    static
    {
      out = new PrintStream();   //PrintStream object is created inside static block
    }
 }
```

System.out.println();

Working with immutable objects :

String Handling :

A string literal in Java is basically a sequence of characters. These characters can be anything like alphabets, numbers or symbols which are enclosed with double quotes. So we can say String is a collection of alpha-numeric character.

How we can create String in Java :-

In java, String can be created by using 3 ways :-

1) By using String Literal

```
String x = "Ravi";
```

2) By using new keyword

```
String y = new String("Hyderabad");
```

3) By using character array

```
char z[] = {'H','E','L','L','O'};
```

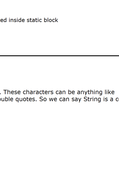
Immutability in String :
---

* String objects are **immutable (un-modifiable)** as well as not eligible for Garbage Collector.

Example :

```
String str = "india";
str.toUpperCase();
System.out.println(str); //india
```

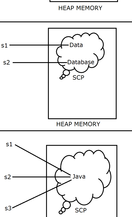
str : Reference Variable
india : String object



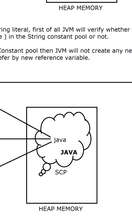
* Whenever we create a String object by using **String literal** then the String object will be created in a very special memory area called **SCP** (String Constant Pool) area.

* Actually o place the String object inside the SCP area, JVM internally call a method called **intern()** method.

```
String s1 = "Java";
s1 =  s1.toUpperCase();
System.out.println(s1); //JAVA
```



```
String s1 = "Data";
String s2 = s1.concat("base");
System.out.println(s1); //Data
System.out.println(s2); //Database
```

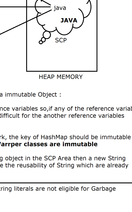


Facts about String and memory :-

```
String s1 = "Java";

String s2 = "Java";

String s3 = "Java";
```



In java Whenever we create a new String object by using String literal, first of all JVM will verify whether the String we want to create is pre-existing (already available) in the String constant pool or not.

If the String is pre-existing (already available) in the String Constant pool then JVM will not create any new String object, the same old existing String object would be refer by new reference variable.

Note :- In SCP area we can't have duplicate String Object.

Why Strings are immutable in Java ?

```
String s1 = "Java";

String s2 = "Java";

String s3 = "Java";
        s3.toUpperCase();
```

The following are the reason so, java made String Object as a immutable Object :

1) In SCP Area, String objects are refered by multiple reference variables so,if any of the reference variable will modify the String Object value then it would be very difficult for the another reference variables pointing to same String object to get the original value.

2) While working with HashMap object in collection framework, the key of HashMap should be immutable type, To support this concept of HashMap, **String and Warrper classes are immutable**

3) Whenever we perform any operation on the existing String object in the SCP Area then a new String object will be created (due to immutability), It will enhance the reusability of String which are already available in the SCP Area.

WAP in Java that describes String objects created by using String literals are not eligible for Garbage Collector.

```
package com.ravi.String_handling;

public class StringGC
{
    public static void main(String[] args) throws Exception
    {
        String s1 = "Java";
        System.out.println("s1 Hashcode is :"+s1.hashCode());

        s1 = null;

        System.gc(); //calling Garbage Collector Explicitly

        //Put the main method into 3 second sleeping mode
        Thread.sleep(5000);
        System.out.println("Main wake up after 5 sec");

        String s2 = "Java";
        System.out.println("s2 hashcode is :"+s2.hashCode());
    }
}
```

Note : Here hashcode is same so, String object is not deleted by GC.