**String, StringBuffer & stringBuilder**

**Mutability vs immutability**

1. String

It is an object

In java program most commonly use object is string

|  |  |
| --- | --- |
| **String** | **stringbuffer** |
| immutable | mutable |
| Object value not change | Object value can be change |
| String s=new string(“sagar”);  s.concate(“Shingare”);  sopln(s); | stringBuffer sb=new stringBuffer(“sagar”);  sb.append(“Shingare”);  sopln(sb); |

why strings are immutable but not stringbuffer

s

here reference variable s will create object “sagar” in heap area

but for object sagarshingare there is no reference variable, therefore object “sagarshingare” will be eibible for garbage collection

in string for every reference variable separate object in heap memory created, therefore reference variable s will point towards to value “sagar” and value will eligible for GC.

Due to this strings are immutable in nature.

**stringBuffer**

sb

in stringbuffer when we call any method on reference variable then new object won’t be create for and reference variable will point to the new object created and old object will eligible for garbage collection.

Therefore stringBuffer are mutable.

**Note**:

In string for every literate it will create new object irrespective of reference variable

**Note**:

But in stringBuffer no separate new object in memory is created.

1. **== and .equals method**

== operator is always used for reference variable comparison whether it is at object class level, string level or stringBuffer class level.

**.equals method**

In string it is used for value level comparison.

What is reference comparison?

Means reference variable points to the same object in heap memory it return true else return false.

String s1=new String(“Sagar”);

String s2=new String(“Sagar”);

Sopln(s1==s2); //false as string will create separate object in heap memory

Sopln(s1.equals(s2)); // true value level comparison

.equals method

Object class (parent class)

(.equals method used for reference comparison only)

String class stringBuffer

(.equals method used for content (.equals method in stringBuffer class is used for

Comparison I’e it shadow object level class reference comparison I’e it doesn’t shadow

Method.) .equals method of object class)

stringBuffer sb1=new stringBuffer(“sagar”);

stringBuffer sb2=new stringBuffer(“sagar”);

sopln(sb1=sb2); //false

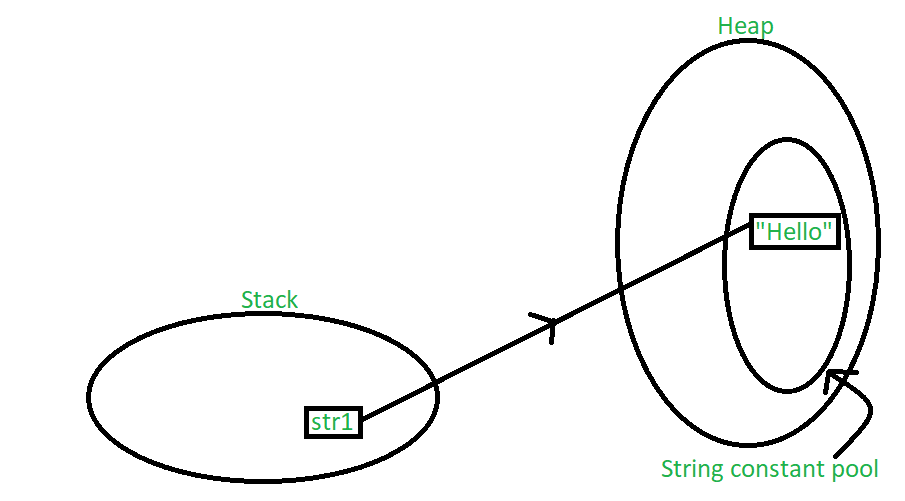
sopln(sb1.equals(sb2));

1. **Heap and string constant pool**

[String](https://www.geeksforgeeks.org/string-class-in-java/) is a sequence of characters. One of the most important characteristics of a string in Java is that they are immutable. In other words, once created, the internal state of a string remains the same throughout the execution of the program. This immutability is achieved through the use of a special string constant pool in the heap.

A string constant pool is a separate place in the heap memory where the values of all the strings which are defined in the program are stored. When we declare a string, an object of type String is created in the stack, while an instance with the value of the string is created in the heap. On standard assignment of a value to a string variable, the variable is allocated stack, while the value is stored in the heap in the string constant pool

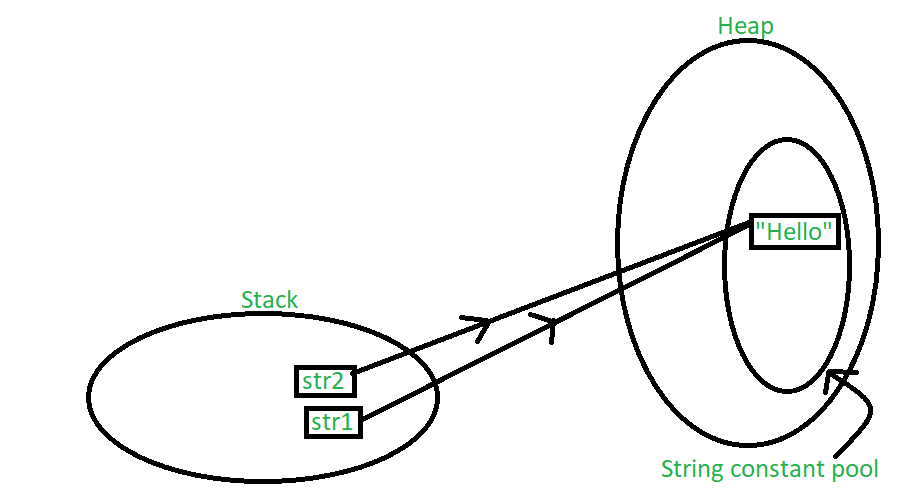
String str1 = "Hello";



in the above scenario, a string object is created in the stack, and the value “Hello” is created and stored in the heap. Since we have normally assigned the value, it is stored in the constant pool area of the heap. A pointer points towards the value stored in the heap from the object in the stack.

String str1 = "Hello";

String str2 = "Hello";

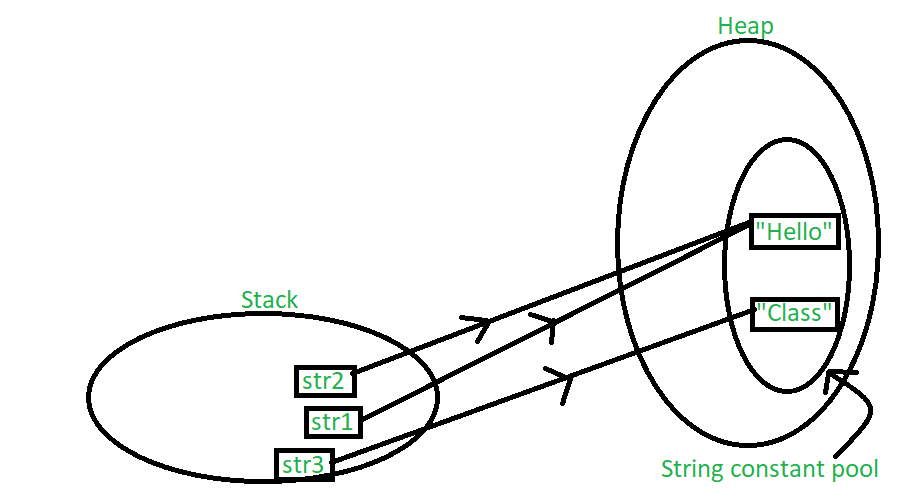


In this case, both the string objects get created in the stack, but another instance of the value “Hello” is not created in the heap. Instead, the previous instance of “Hello” is re-used. The string constant pool is a small cache that resides within the heap. Java stores all the values inside the string constant pool on direct allocation. This way, if a similar value needs to be accessed again, a new string object created in the stack can reference it directly with the help of a pointer. In other words, the string constant pool exists mainly to reduce memory usage and improve the re-use of existing instances in memory. When a string object is assigned a different value, the new value will be registered in the string constant pool as a separate instance.

String str1 = "Hello";

String str2 = "Hello";

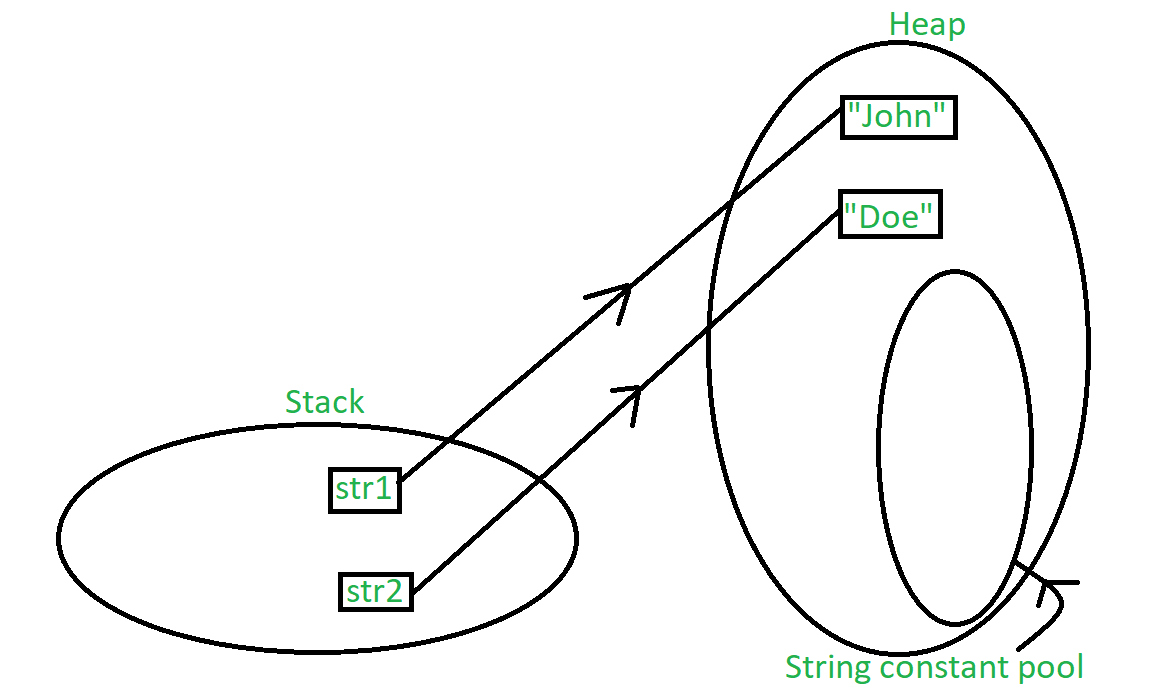
String str3 = "Class";



One way to skip this memory allocation is to use the [new keyword](https://www.geeksforgeeks.org/new-operator-java/) while creating a new string object. The ‘new’ keyword forces a new instance to always be created regardless of whether the same value was used previously or not. Using ‘new’ forces the instance to be created in the heap outside the string constant pool which is clear, since caching and re-using of instances isn’t allowed here.

String str1 = new String("John");

String str2 = new String("Doe");



**Note**: if we are using “new” keyword then reference variable will create inside stack memory and “value” will create inside heap memory for all new keyword irrespective of the value,

But if we are not creating with new keyword then then reference variable will create inside stack and value will create inside string constant pool, if values are same then no new value will create inside

SCP

heap

String str1=new String(“Durga”);

String str2=new String(“Durga”);

String str3=”Durga”;

String str4=”Durga”;

stack

2)

String s=new String(“Durga”);

s.concate(“software”);

s=s.concate(“solution”);

stack heap SCP

s durga Durga

durgasoftware software

durgasolution solution

**Note**: for every string literal one object will compulsory create in SCP memory.

**If because of runtime operation if new object is created then that object will only create in heap memory and not in SCP.**

**In this case durgasoftware is created at runtime with no reference variable, so it will only create in heap memory and eligible for garbage collection,**

**Also**

**But in “durgasolution” object will create in heap memory, but there is a reference variable therefore it is not eligible for garbage collection**

String s1=new String(“spring”);

s1.concate(“full”);

String s2=s1.concate(“winter”);

s2.concate(“summer”);

sopln(s1); //spring

sopln(s2); //springwinter

Heap SCP

s1 spring spring

springfull full

s2 springwinter winter

springwintersummer summer

5)

String s1=new String(“you cannot change me”);

String s2=new String(“you cannot change me”);

Sopln(s1==s2); //false

String s3=”you cannot change me”;

Sopln(s1==s3); //false

String s4=”you cannot change me”:

Sopln(s3==s4); //true

String s5=”you cannot” + “change me”;

Sopln(s4==s5);

String s6=”you cannot”;

String s7=s5 + “change me”;

Sopln(s4==s7); //false

final String s8=”you cannot”;

String s9=”change me”;

Sopln(s4==s9); //true

**Note**:

for s5 “you cannot” and “change me” are concate therefore + operation will be done inside compile time and form you cannot change me, therefore inside SCP.

for s6 and s7 since s6 is variable, therefore concatenation operation will perform at runtime.

final= concate keyword

final variable value will be replaced in compiled time only therefore it same like s5.

1. **Importance of String Constant Pool**

Because of SCP concept string objects are immutable

The **String is immutable** in [Java](https://www.javatpoint.com/java-tutorial) because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it.

The String objects are cached in the String pool, and it makes the [String immutable](https://www.javatpoint.com/immutable-string). The cached String literals are accessed by multiple clients. So, there is always a risk, where action performs by one client affects all other clients. For example, if one client performs an action and changes the string value from Pressure to PRESSURE, all remaining clients will also read that value. For the performance reason, caching of String objects was important, so to remove that risk, we have to make the String Immutable.

* The String pool cannot be possible if String is not immutable in Java. A lot of heap space is saved by [JRE](https://www.javatpoint.com/java-jre). The same string variable can be referred to by more than one string variable in the pool. String interning can also not be possible if the String would not be immutable.
* If we don't make the String immutable, it will pose a serious security threat to the application. For example, database usernames, passwords are passed as strings to receive database connections. The [socket programming](https://www.javatpoint.com/socket-programming) host and port descriptions are also passed as strings. The String is immutable, so its value cannot be changed. If the String doesn't remain immutable, any hacker can cause a security issue in the application by changing the reference value.
* The String is safe for multithreading because of its immutableness. Different threads can access a single "String instance". It removes the synchronization for thread safety because we make strings thread-safe implicitly.
* Immutability gives the security of loading the correct class by Classloader. For example, suppose we have an instance where we try to load java.sql.Connection class but the changes in the referenced value to the myhacked.Connection class does unwanted things to our database.

1. **Important FAQ’S on String and stringBuffer**
2. Why SCP concept and immutability is only applicable for string object and not for stringBuffer

We all know that the String class in Java is mutable i.e. once we create a String variable we cannot modify its data or do any manipulations.

But, there may be scenarios where we need to modify the data of String variables. In such cases, we could use StringBuffer class.

1. Why string object are immutable wherein stringbuffer objects are mutable
2. In addition to string objects any other objects are immutable in java

All wrapper class objects are immutable in java.

1. **Important constructors of string class**

1) String s=new String();

Create new empty string object in heap area.

2) String s=new String(string literal);

For a given string literal equivalent object will create in heap area.

3)String s=new String(stringBuffer sb);

For a given stringBuffer object equivalent string object will create

4)String s=new String(stringBuilder sb);

For a given stringBuilder object equivalent string object will create .

5)String s=new String(char[] chr);

For a given character array a given string object will create

Char[] chr={ ‘j’,’a’,’v’,a’};

String s=new String(chr);

Sopln(s) //java

6)String s=new String(byte[] b);

For byte array

1. **Important methods of string class**
2. charAt(int index)

String s=”Durga”;

Sopln(s.charAt(3));

1. concate (+)

String s=”Durga”;

S1=s.concate(“software”);

Sopln(S1); // durgasoftware

1. equals()

to check equality of string object value

String s=”Durga”;

Sopln(s.equals(“durga”) //false

1. equalsIgnoreCase()

o/p true

1. boolean isEmpty();

String s=” “ ;

Sopln(s.isEmpty()); //true

1. int length()

String s=”Durga”;

Sopln(s.length());

1. replace(old chr, new chr)

String s=”abcd”;

Sopln(s.peplace(‘a’, ‘r’); //rbcd

1. substring(int begin);

return from index I to end of the string

String s=”abcdefg”;

Sopln(s.subString(3)); // defg

1. substring(int begin, int end)

return string from i to i-1

String s=”abcdef”;

Sopln(s.substring(3,5); //de I’e 3 to 5-1

1. i

int [] x=[1,2,3,4,5];

sopln(s.length) =4

Note:

length variable is applicable for array (length)

length method is applicable for string (length())

1. indexOf(char c)

return index of specified character

String s=’Durga’;

Sopln(s.indexOf(‘g’); //g

1. lastindexOf(char c);

return the index of last char

1. toLowerCase();

convert into lower case

String s=’DURGA’;

Sopln(s.toLowerCase());

1. toUpperCase();

convert into uppercase

1. trim();

used to remove blanked spaced at the beginning and the ending of the string but not in between.

String s=”\_\_durga\_\_”;

String s1=”Durga soft”:

Sopln(s.trim().lenght()); //4

Sopln(s.trim().length()); //10

**Important conclusion about string immutability**

What is immutability?

Once we create an object we are not allowed to change its content

If we are trying to change its content then with those changes a new object will create

If no changes in content then existing object will reused.

String s1=new String(“durga”);

String s2=s1.toUpperCase();

String s3=s1.toLowerCase();

Sopln(s1==s2); //false

Sopln(s1==s3); //true

Stack heap SCP

S1 durga durga

S2 DURGA

S3

**Creation of our immutable class**

final class Test{

private int I;

Test(){

this.i=i;

}

Public test modifier(int i){

if(this.i==i){

return this;

}

else

{

return new Test(i);

}

}

}

Because of this modifier method test class become immutable

All immutable classes are declare as final

Eg

String, Wrapper

Public static void main(Strin[] args){

Test t1=new Test(10);

Test t2=t1.modify(100);

Test t3=t1.modify(10);

Sopln(t1==t2); //false

Sopln(t1==t3); //true

}

}

final vs immutability

immutability is related to object

final related to reference variable

**Note**:

If you declare reference variable as final you can’t reassign that to new object

By declaring reference variable final we can’t make class immutable.

1. **Need of StringBuffer**

When all required changes need to perform on existing object the we should go for stringBuffer.

stringBuffer class present inside java.lang.stringBuffer class

**10)Important constructors of stringBuffer**

1) stringBuffer sb=new stringBuffer();

-by default capacity of empty stringBuffer is 16

-and

If we want to add to literal which is more then 16 then new stringBuffer will create,it will copy all existing data into new stringBuffer and previous existing stringbuffer will eligible for garbage collection.

New capacity=(current capacity +1)\*2

stringBuffer sb=new StringBuffer();

sopln(sb.capacity());

sb.append(‘asedfvtgbhyjmnky’);

sopln(sb.capacity()); //16

sb.append(‘t’);

sopln(ab.capacity()); //34

2) stingBuffer sb=new stringBuffer(int initialcapacity);

stringBuffer sb=new stringBuffer(1000);

3) equivalent stringBuffer for a given string

stringBuffer sb=new stringBuffer(String 3);

capacity=s.length + default capacity of stringBuffer

string s=’durga’

capacity=s+16 =21

1. **Important methods of stringBuffer class**
2. **length**()
3. **capacity**()
4. **charAt(int index)**
5. stringBuffer sb=new stringBuffer(“Durga”);

sopln(sb.charAt(4)); //g

1. public void **setcharAt**(int index, new char);

stringBuffer sb=new stringBuffer(“java”);

sb.setcharAt(0, ‘y’);

sopln(sb); //yava

1. **append**(string s);

it append values

append(string s/byte b/int i/ )

stringBuffer sb=new stringBuffer();

sb.append(‘PI value is’);

ab.append(3.14);

ab.append(‘it is exactly’);

ab.append(true);

sopln(sb) //PI value is3.14it is exactlytrue

1. public stringBuffer **insertAt**(int index, string r/double d/boolean b/char c)

stringBuffer sb=new stringBuffer(‘abcdefg’);

sb.insertAt(2,’xyz’);

sb.insertAt(3,true);

sb.insertAt(4,10.5t);

sopln(sb);

1. public **stringBuffer** delete(int begin, int end);

delete char from begin index to end-1 index

stringBuffer sb=new stringBuffer(‘abcdefg”);

ab.delete(2,5);

sopln(sb)

1. public stringBuffer **deletecharAt**(index number)

stringBuffer sb=new stringBuffer(‘abcdefg’);

sb.deletecharAt(3);

sopln(sb); //abcefg

1. public **stringBuffer** reverse();

stringBuffer sb=new stringBuffer(“Durga”);

sb.reverse();

sopln(sb) //agruD

1. public void **setLength**(int length);

stringBuffer sb=new stringBuffer(“Aishwarya”);

sb.setLegth(8);

sopln(sb); //Aishwar

1. public void **ensureCapacity**(int capacity);

stringBuffer sb=new stringBuffer();

sopln(sb.capacity()); //16

sb.ensureCapacity(100);

sopln(sb.capacity()); //100

1. public void **trimTosize**();

stringBuffer sb =new stringBuffer(1000);

sb.append(‘abc’);

sopln(ab.capacity()); //1000

sb.trimTosize();

sopln(sb.Capacity()); //3

1. **Need of stringBuilder and differences with stringBuffer**

-every method present inside stringBuffer is **synchronized,**

Therefore at a time only one thread is allowed to operate

To overcome this issue stringBuilder is introduced which is **not synchronized**

-in multithreaded environment stringBuffer can’t be used.

Therefore **stringBuffer** introduced in **1.5 v**

1. **String vs stringBuffer vs stringBuilder**
2. **String**

If content is fixed then we should go for string

1. **stringBuffer**

if content is changeable and want thread safety then should go for stringBuffer

1. **stringBuilder**

if content is changeable and don’t want thread safety or want multiple thread to be execute simultaneously then we should go for stringBuilder

**Note**: all immutable objects are thread safe.

1. **Method chaining**

sb.m1().m2() ----------mn();

stringBuilder sb=new stringBuilder();

sb.append(“Durga”).append(“solution”).reverse().insert(2,’xyz’).delete(3,7);

sopln(sb);

1. **toString() method**

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the String representation of the object.

If you print any object, Java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depending on your implementation.

Note: whenever we are trying to print any object reference then internally toString() method is invoked

If we did not define the toString() method in your class then the Object class toString() method is invoked otherwise our implemented or overridden toString() method will be called.

***Note:****Default behavior of toString() is to print class name, then @, then unsigned hexadecimal representation of the hash code of the object.*

public String toString() {

return getClass().getName()+"@"+Integer.toHexString(hashCode());

}

class Best\_Friend {

    // Member attributes of this class

    String name;

    int age;

    String college;

    String course;

    String address;

    // Constructor of this class

    Best\_Friend(String name, int age, String college,

                String course, String address)

    {

        // This variable refers to current instance itself

        this.name = name;

        this.age = age;

        this.college = college;

        this.course = course;

        this.address = address;

    }

    // Method of this class

    // Main driver method

    public static void main(String[] args)

    {

        // Creating an object of this class

        // Custom attributes been passed as in arguments

        Best\_Friend b = new Best\_Friend(

            "Gulpreet Kaur", 21, "BIT MESRA", "M.TECH",

            "Kiriburu");

        // Print and display commands to illustrate

        // toString() method as both will print the same

        // Print the object

        System.out.println(b);

        // Print the object but implicitly using toString()

        // method

        System.out.println(b.toString());

    }

}

**O/P**

Best\_Friend@3d075dc0

Best\_Friend@3d075dc0

In the above program, we create an Object of Best\_Friend class and provide all the information of a friend. But when we try to print the Object, then we are getting some output which is in the form of classname@HashCode\_in\_Hexadeciaml\_form. If we want the proper information about the Best\_friend object, then we have to override the toString() method of the Object class in our Best\_Friend class.

**By overriding toString() method**

class Best\_Friend {

    // Member attributes of this class

    String name;

    int age;

    String college;

    String course;

    String address;

    // Constructor of this class

    Best\_Friend(String name, int age, String college,

                String course, String address)

    {

        // This keyword refers to current instance itself

        this.name = name;

        this.age = age;

        this.college = college;

        this.course = course;

        this.address = address;

    }

    // Method 1

    // Creating our own toString() method

    public String toString()

    {

        return name + " " + age + " " + college + " "

            + course + " " + address;

    }

    // Method 2

    // Main driver method

    public static void main(String[] args)

    {

        // Creating object of class inside main() method

        Best\_Friend b = new Best\_Friend(

            "Gulpreet Kaur", 21, "BIT MESRA", "M.TECH",

            "Kiriburu");

        // Print and display commands to illustrate

        // toString() method as both will print the same

        // Print the object

        System.out.println(b);

        // Printing object but using toString() method

        System.out.println(b.toString());

    }

}

O/P

Gulpreet Kaur 21 BIT MESRA M.TECH Kiriburu

Gulpreet Kaur 21 BIT MESRA M.TECH Kiriburu

***Note:****In all wrapper classes, all collection classes, String class, StringBuffer, StringBuilder classes toString() method is overridden for meaningful String representation. Hence, it is highly recommended to override toString() method in our class also.*