Introduction to String Class

BY
Apparao G

What is String ?

- String is a sequence of characters.

In Java programming language, strings are treated as objects.

- The Java platform provides the String class to create and manipulate strings.

String objects are immutable! - That means once a string object is created it cannot be altered.

For mutable string, you can use StringBuffer and StringBuilder classes.

[An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper class's objects are immutable.]

How to create String object?

There are two ways to create String object:

1. By string literal

2. By new keyword

String literal :

String literal is created by double quote. For Example: String s="Hello";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.

For example:

String s1="Welcome";

String s2="Welcome";//no new object will be created

In the above example only one object will be created. First time JVM will find no string object with the name "Welcome" in string constant pool , so it will create a new object. Second time it will find the string with the name "Welcome" in string constant pool , so it will not create new object whether will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool inside the Heap memory.

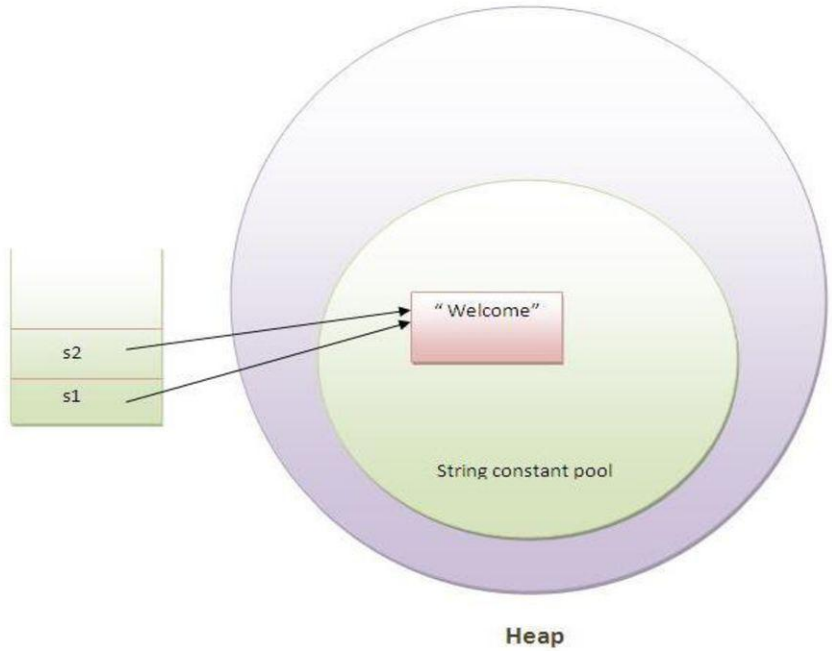
Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

By new keyword :

String s=new String("Welcome"); //creates two objects and one reference variable

In such case, JVM will create a new String object in normal(nonpool) Heap memory and the literal "Welcome" will be placed in the string constant pool.The variable s will refer to the object in Heap(nonpool).


s2
s1
" Welcome"
String constant pool
Heap

Immutable String in Java:

In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

```
Public class ImmutableTest
{
 public static void main(String args[])
    String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string  at the end
    System.out.println(s);//will print Sachin because strings are Immutable objects
  }
}
```

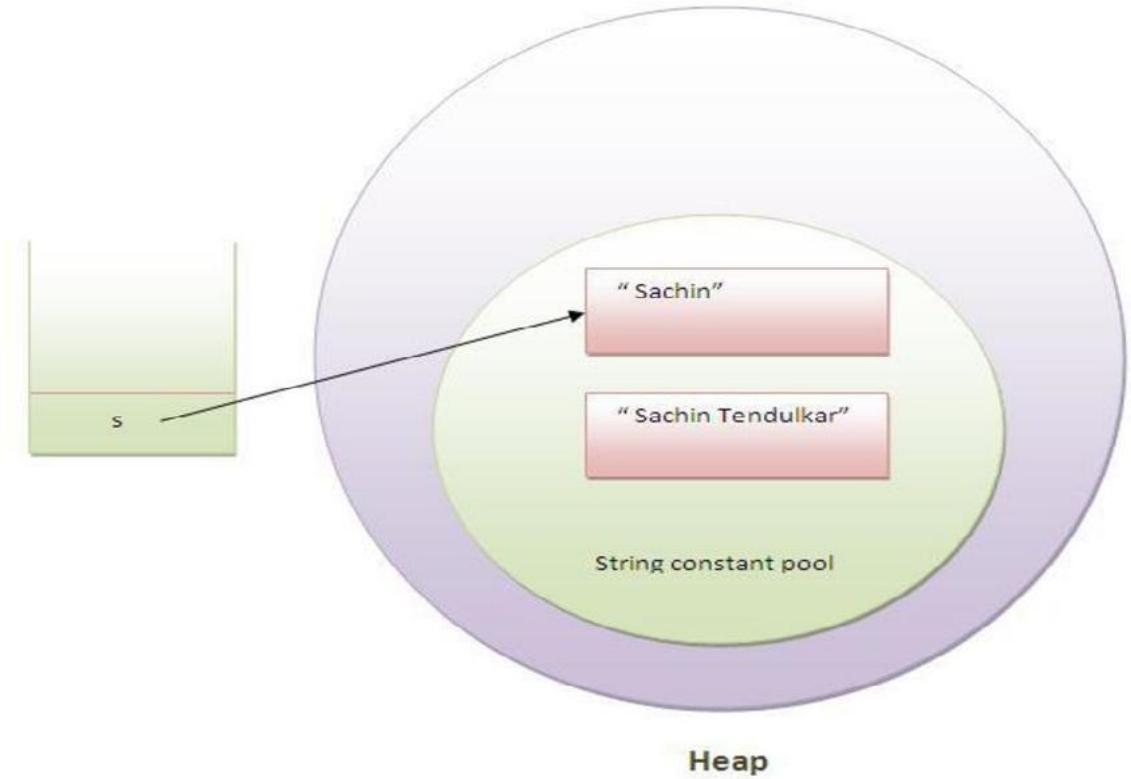
As you can see in the above figure that two objects are created but s reference

variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

For example:

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



String comparison in Java:

We can compare two given strings on the basis of content and reference.

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

There are three ways to compare String objects:

By equals() method

By = = operator

By compareTo() method

1) By equals() method

equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

public boolean equals(Object another){} compares this string to the specified object.

public boolean equalsIgnoreCase(String another){} compares this String to another String, ignoring case.

```
public class StringEqualsMethodTest {
 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 of equalsIgnoreCase(String) method :

```
Public class EqualsIgnoreExample {
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";
   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s3));//true
 }
}
```

2. By == operator The = = operator compares references not values.

```
class Simple{
 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 nonpool)
 }
}
```

3. By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than. Suppose s1 and s2 are two string variables. If:

s1 == s2 :0

s1 > s2 :positive value

s1 < s2 :negative value

Some of the important methods in String class:

char charAt(int index) : returns char value for the particular index

int length() : returns string length

String substring(int beginIndex): returns substring for given begin index.

String substring(int beginIndex, int endIndex): returns substring for given begin index and end index.

boolean contains(CharSequence s): returns true or false after matching the sequence of char value.

static String join(CharSequence delimiter, CharSequence... elements): returns a joined string.

static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements): returns a joined string.

boolean equals(Object another): checks the equality of string with the given object.

boolean isEmpty(): checks if string is empty.

String concat(String str): concatenates the specified string.

String replace(char old, char new): replaces all occurrences of the specified char value.

String replace(CharSequence old, CharSequence new): replaces all occurrences of the specified CharSequence.

static String equalsIgnoreCase(String another): compares another string. It doesn't check case.

String[] split(String regex): returns a split string matching regex.

String[] split(String regex, int limit): returns a split string matching regex and limit.

String intern(): returns an interned string.

int indexOf(int ch): returns the specified char value index.

int indexOf(int ch, int fromIndex): returns the specified char value index starting with given index.

int indexOf(String substring): returns the specified substring index.

int indexOf(String substring, int fromIndex): returns the specified substring index starting with given index.

String toLowerCase(): returns a string in lowercase.

String toLowerCase(Locale l): returns a string in lowercase using specified locale.

String toUpperCase(): returns a string in uppercase.

String toUpperCase(Locale l): returns a string in uppercase using specified locale.

String trim(): removes beginning and ending spaces of this string.

static String valueOf(int value): converts given type into string. It is an overloaded method.

StringBuffer class :

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

  StringBuffer is synchronized (it is thread-safe i.e multiple threads cannot access it simultaneously.)

- Important Constructors of StringBuffer class:

  StringBuffer(): creates an empty string buffer with the initial capacity of 16.

  StringBuffer(String str): creates a string buffer with the specified string.

  StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

- Important methods of StringBuffer class:

append(String s): is used to append the specified string with this string.

The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

insert(int offset, String s): is used to insert the specified string with this string at the specified position.

The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

replace(int startIndex, int endIndex, String str): is used to replace the string from specified startIndex and endIndex.

delete(int startIndex, int endIndex): is used to delete the string from specified startIndex and endIndex.

reverse(): is used to reverse the string.

capacity(): is used to return the current capacity.

StringBuilder class:

- Java StringBuilder class is used to create mutable (modifiable) string.
- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.

Example:

```
class StringBuilderExample {
  public static void main(String args[]){
    StringBuilder sb=new StringBuilder("Hello ");
    sb.append("World");
    System.out.println(sb.toString());
  }
}
```

Difference between String and StringBuffer:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Difference between StringBuffer and StringBuilder :

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

Why to use char[] array over a string for storing passwords in Java?

Strings are immutable: Strings are immutable in Java and therefore if a password is stored as plain text it will be available in memory until Garbage collector clears it and as Strings are used in String pool for re-usability there are high chances that it will remain in memory for long duration, which is a security threat. Strings are immutable and there is no way that the content of Strings can be changed because any change will produce new String.

With an array, the data can be wiped explicitly data after its work is complete. The array can be overwritten and and the password won't be present anywhere in the system, even before garbage collection.

Security: Any one who has access to memory dump can find the password in clear text and that's another reason to use encrypted password than plain text. So Storing password in character array clearly mitigates security risk of stealing password.

Log file safety: With an array, one can explicitly wipe the data , overwrite the array and the password won't be present anywhere in the system.

With plain String, there are much higher chances of accidentally printing the password to logs, monitors or some other insecure place. char[] is less vulnerable.

Example:

```
Public class PasswordTest {

 public static void main(String[] args)

   {

      String strPwd = "password";

      char[] charPwd = new char[] {'p','a','s','s','w','o','r','d'};


      System.out.println("String password: " + strPwd );

      System.out.println("Character password: " + charPwd );

   }

}
```

Output:

```
 String password: password

 Character password: [C@15db9742
```

How to create user defined immutable class:

- Declare class ***final,*** so that no other classes can extend it.
- Declare all fields as ***final,*** so that they're initialized only once inside the constructor and never modified afterward.
- Don't expose setter methods.
- When exposing methods which modify the state of the class, you must always return a new instance of the class.
- If the class holds a mutable object:
  - Inside the constructor, make sure to use a clone copy of the passed argument and never set your mutable field to the real instance passed through constructor, this is to prevent the clients who pass the object from modifying it afterwards.
  - Make sure to always return a clone copy of the field and never return the real object instance.

Example:

```
public final class ImmutableEmployee {
    private final int id;
    private final String name;
    public ImmutableEmployee (int id, String name) {
        this.name = name;
        this.id = id;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
}
```

Passing Mutable Objects to Immutable Class:

```
public class Age {
   private int day;
   private int month;
   private int year;
   public int getDay() {
      return day;
   }
   public void setDay(int day) {
   this.day = day;
   }
   public int getMonth() {
   return month;
   }
   public void setMonth(int month) {
   this.month = month;
   }
   public int getYear() {
   return year;
   }
   public void setYear(int year) {
   this.year = year;
   }
}
```

```
public final class ImmutableEmployee {
    private final int id;
    private final String name;
    private final Age age;
    public ImmutableEmployee(int id, String name, Age age) {
    this.name = name;
    this.id = id;
    this.age = age;
    }
    public int getId() {
    return id;
    }
    public String getName() {
    return name;
    }
    public Age getAge() {
    return age;
    }
}
```

Now, we added a new mutable field of type Age to our immutable class and assign it as normal inside the constructor.

Let’s create a simple test class and verify that ImmutableEmployee is no more immutable:

```
public static void main(String[] args) {
    Age age = new Age();
    age.setDay(1);
    age.setMonth(1);
    age.setYear(1992);
    ImmutableEmployee employee = new ImmutableEmployee(1, "Bob", age);
    System.out.println("Bob age year before modification = " +
            employee.getAge().getYear());
    age.setYear(1993);
    System.out.println("Bob age year after modification = " +
            employee.getAge().getYear());
}
```

Output:
Bob age year before modification = 1992

Bob age year after modification = 1993

In order to fix this and make our class again immutable, we follow step #5 from the steps that we mention above for creating an immutable class. So we modify the constructor in order to clone the passed argument of Age and use a clone instance of it.

```
public ImmutableEmployee(int id, String name, Age age) {
    this.name = name;
    this.id = id;
    Age cloneAge = new Age();
    cloneAge.setDay(age.getDay());
    cloneAge.setMonth(age.getMonth());
    cloneAge.setYear(age.getYear());
    this.age = cloneAge;
}
```

Now, if we run our test, we get the following output:

```
Alex age year before modification = 1992
Alex age year after modification = 1992
```

Returning Mutable Objects From Immutable Class:

However, our class still has a leak and is not fully immutable, let’s take the following test scenario:

```
public static void main(String[] args) {
    Age age = new Age();
    age.setDay(1);
    age.setMonth(1);
    age.setYear(1992);
    ImmutableStudent student = new ImmutableStudent(1, "Alex", age);
    System.out.println("Alex age year before modification = " + student.getAge().getYear());
    student.getAge().setYear(1993);
    System.out.println("Alex age year after modification = " + student.getAge().getYear());
}
```

output:

Alex age year before modification = 1992

Alex age year after modification = 1993

when returning mutable fields from immutable object, you should return a clone instance of them and not the real instance of the field.

So we modify getAge() in order to return a clone of the object's age:

```
public Age getAge() {
    Age cloneAge = new Age();
    cloneAge.setDay(this.age.getDay());
    cloneAge.setMonth(this.age.getMonth());
    cloneAge.setYear(this.age.getYear());
    return cloneAge;
}
```

Now the class becomes fully immutable and provides no way or method for other objects to modify its state.

Now run the test case again.

output:

Alex age year before modification = 1992

Alex age year after modification = 1992

Advantages of Immutable class:

1. By default thread safe
2. It uses HashMap keys
3. Proper memory utilization.

Problems:

1. Check given two Strings are anagrams of each other?
2. Revere of a string using iteration.
3. Find all permutations of given String
4. How to reverse words in a sentence.
5. Check given String is Palindrome or not.
6. How to remove duplicate characters from String
7. Write a program to remove a given character from String
8. Write a program to find the longest palindrome in a string
9. Write a program to print Even length words in a String
10. Write a program to add two binary strings.

   Example: input strings: 0101, 0100

   output string: 1001