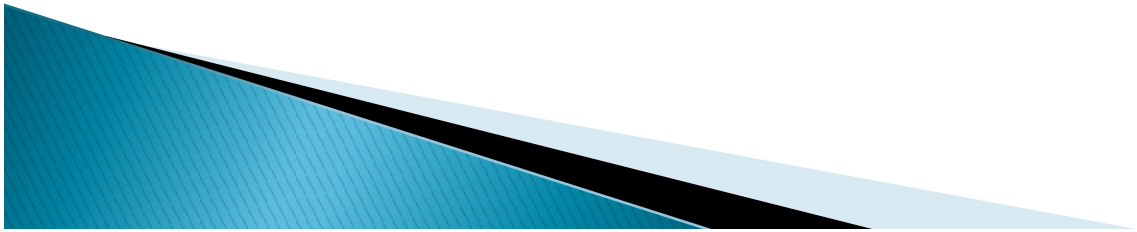
String Class in Java

By
Sunil Kumar(Master of Sc.)
Bangalore, India

1. Agenda

- String Class
- StringBuilder class
- Primitive wrapper classes
- Final fields
- Enumeration types



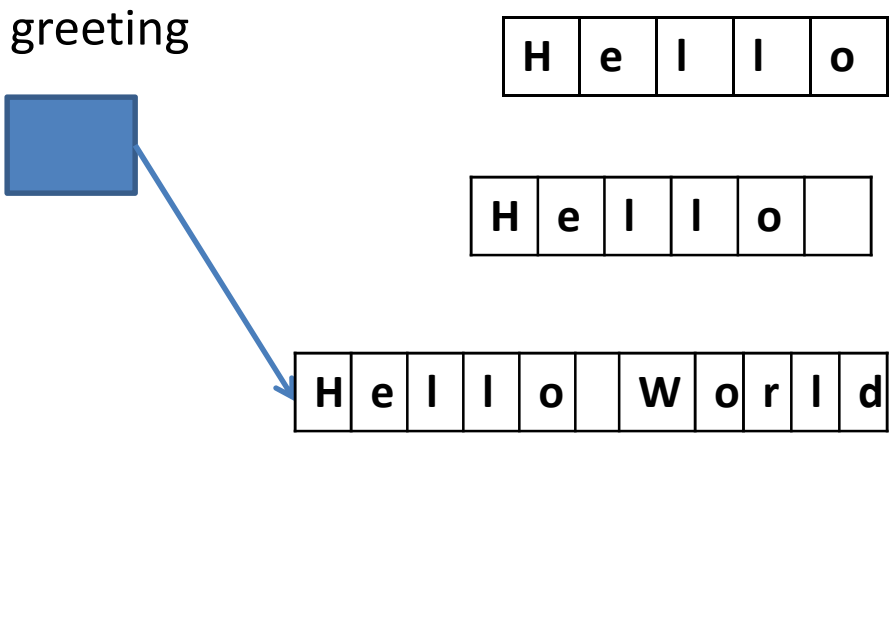
String Class

- In our earlier classes, we have talked about primitive data type and in that char type which allows us to work deal with individual characters. In general, we usually work with sequence of characters and so String class comes in.
- The String class stores a sequence of Unicode characters
  - Stored using UTF-16 encoding (Unicode Transformation Format).
  - Literals are enclosed in double quotes (“ “)
  - Values can be concatenated using + and +=
  - String object are immutable

String Class

- What does this immutable means.
  - Suppose we have declared the java program as below.

```
String greeting = “Hello”;
greeting += greeting + “ ”
greeting +=
greeting+”World”;
```



String Class

- After 1st line of executing the code, a reference is created with the name greeting and memory is allocated as shown in the figure.
- After 2nd line, another new memory area is created and now greeting is re-pointing to the newly created memory.
- Similarly after the 3rd line.

String Class Methods

Operation	Methods
Length	length
String for non-string	valueOf
Create new string(s) from existing	concat, replace, toLowerCase, toUpperCase, trim, split
Formatting	format
Extract substring	chatAt
Test substring	contains, endsWith, startWith, indexOf, lastIndexOf
comparison	compareTo, compareToIgnoreCase, isEmpty, equals, equalsIgnoreCase

String Class Methods

- These string class methods whatever are discussed in the last slides are just the common methods used frequently during the program.
- There are lot of other methods which can be found at official Java site. http://bit.ly/javastringclass

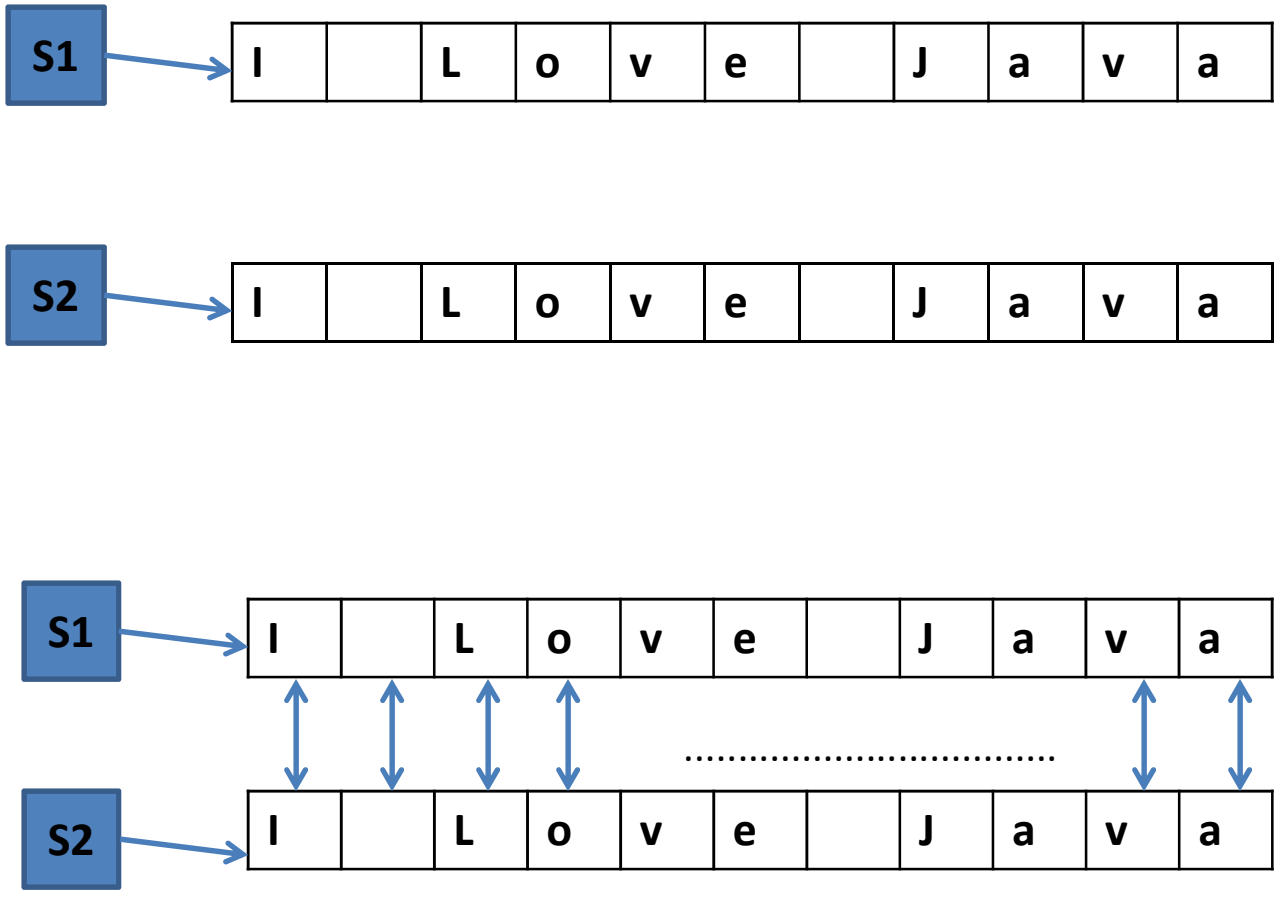
String Equality

```
String s1 = “I Love”;
s1 += “Java”;

String s2 = “I Love”;
s2 += “Java”;

if(s1==s2)   //always false
      //do something

if(s1.equals(s2))  //True
      //do something
```



String Equality

- Normal == method, see if the variables are pointing to the same object or not. Here s1 is pointing to different object and s2 is pointing to some other object, though values in these 2 are exactly same.
- String equals method try to compare the character by character and if values are same then it returns true.
- Here we should note that reference based comparison is very quick and inexpensive but character by character comparison is quite expensive specially in case of very long strings. So in case of very long string comparison, we should take advantage of inexpensive nature of reference based comparison. That's where the intern method comes in.

String Equality & intern Method

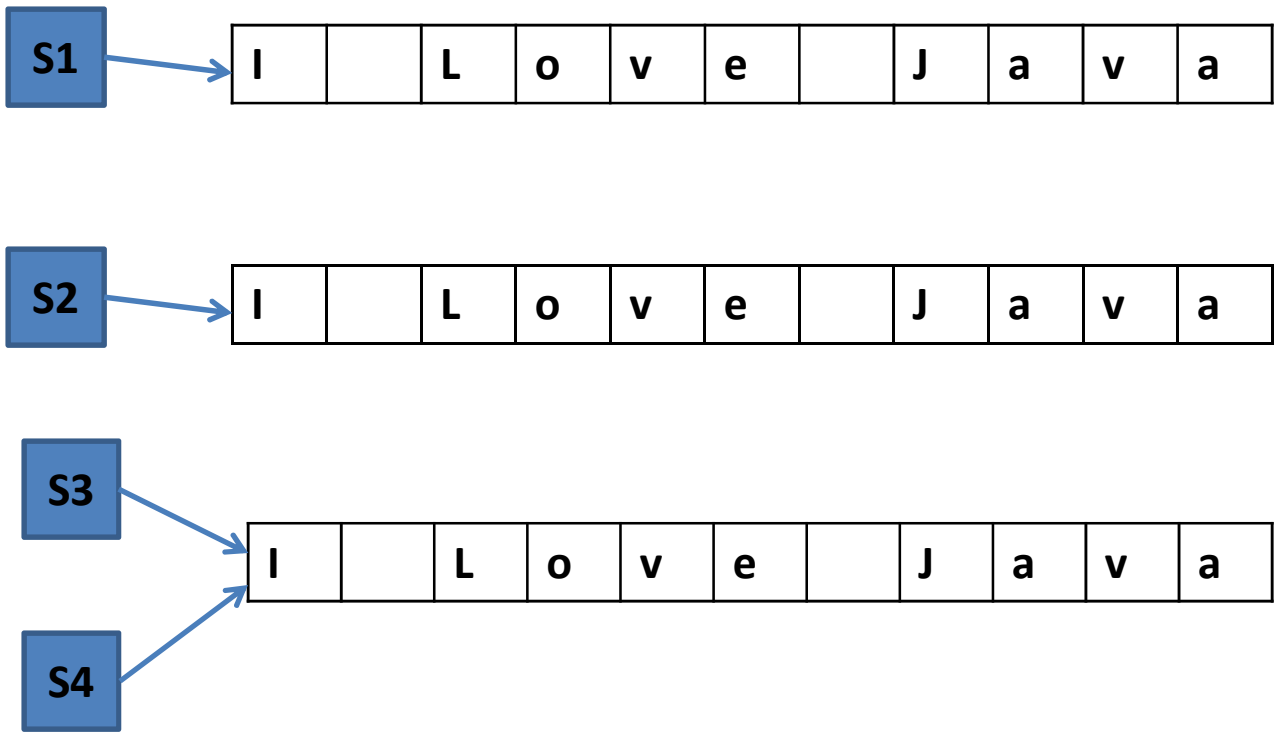
```
String s1 = “I Love”;
s1 += “Java”;

String s2 = “I Love”;
s2 += “Java”;

if(s1==s2)   //always false
      //do something

if(s1.equals(s2))  //True
      //do something

String s3 = s1.intern();
String s4 = s2.intern();
if(s1==s2)   //True
      //do something
```



String Equality & intern Method

- An intern method gives us back a canonicalized reference of a string value.
- It means that, when we will declare another string variable as in case of s3, it will get the same string value as s1. Also, in case of s4, it will get the same string value as s2 but it will not create a new instance. It will 1st check if the string with same exact value exist in the memory block or not and then the intern assures that the 2 string with same value, will reference the exact same object.
- Thus it allows us to do the very inexpensive == comparison by reference. But there is a bit of overhead to interning a string. So the thumb rule is that when String are small then we will use equals method and when there is a series of value in an array or other collection, then we will use intern method to do the comparison.

Converting Non-string Types to Strings

- We often need to convert non-string types into strings having a value of message makes it easy to do things like build a message or display output to a user.
  - String.valueOf provides overrides to handle most types.
  - Conversion often happen implicitly in most of the cases. Like primitive type or so.
  - Class conversions controlled by the class toString method.

Converting Non-string Types to Strings

```
int iVal = 100;

String sVal = String.valueOf(iVal);
//gives o/p as “100”

int i = 2, j=3;
int result = i* j;
System.out.println(i+” ”+j+” = ”+result);
/*Fany o/p for better understanding to
      user “2 * 3 = 6” */
```

```
Flight myFlight = new Flight(175);
System.out.println(“My flight is ”+myFlight);
//o/p “My flight is Javaapplication1@74a1448b”
//o/p after below: My Flight is Flight # 175
```

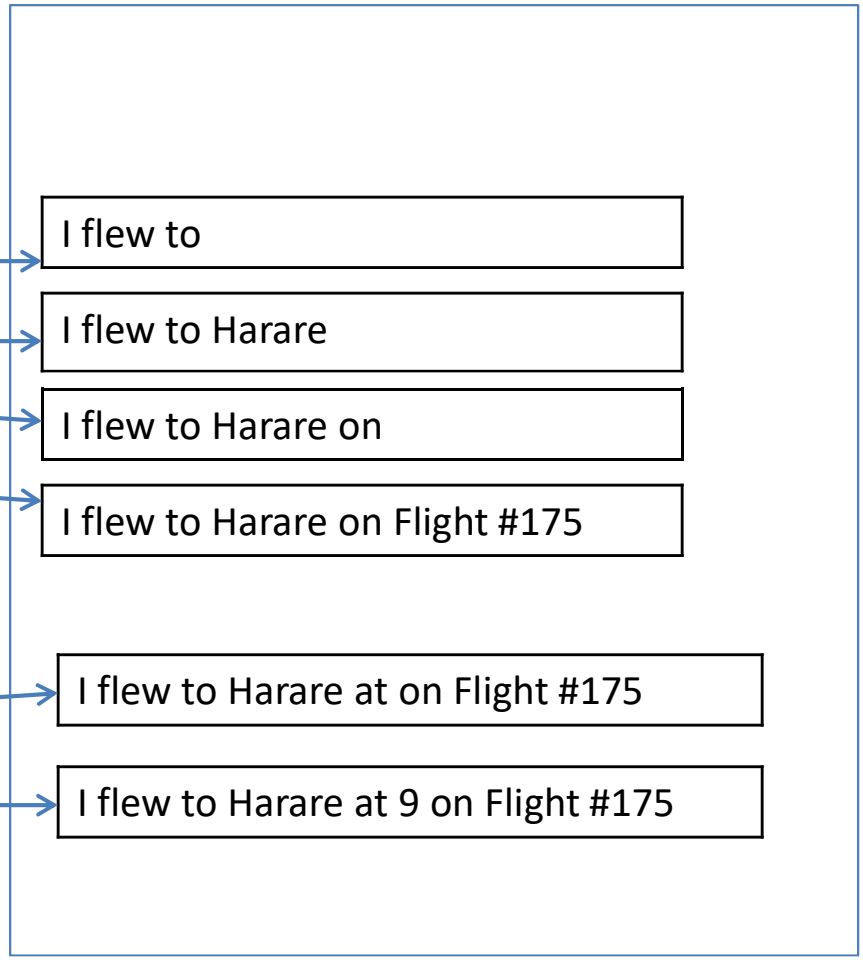
```
Public class Flight{
      int flightNumber;
      char flightClass;
//other method/members elided for clarity
      @override
      public String toString(){
      if(flightNumber > 0)
            return “Flight # ”+flightNumber;
      else
            return “Flight # ”+flightClass;
      }
}
```

StringBuilder

- Since Strings are immutable, that means that any modification of the strings results to a creation of new string and that is not efficient way of programming if it requires so and thats where StringBuilder comes into picture.
- StringBuilder provides mutable string buffer, providing us with an efficient way to manipulate strings.
  - For best performance pre-size buffer.
  - Will grow automatically if needed. But we should minimize this as there is a little bit of overhead each time the system has to grow a StringBuilder instance.
- StringBuilder has number of methods and most common:
  - append : allow us to add new content to the end of StringBuilder
  - insert: allows us to add new content anywhere in StringBuilder

StringBuilder

```
StringBuilder sb = new StringBuilder(40);
Flight myFlight = new Flight(175);
String location = “Harare”;
sb.append(“I flew to ”);
sb.append(“location”);
sb.append(“ on ”);
sb.append(myFlight);
int time = 9;
int pos = sb.lenght() – “ on ”.lenght() –
      myFlight.toString().lenght;
sb.insert(pos, “ at ”);
sb.insert(pos + 4, time);
String message = sb.toString();
```



Classes vs Primitives

- Classes provide convenience
  - Common interaction through object class
  - Fields and methods specific to the type
  - Incurs an overhead cost
    - Every class instance has a certain amount of memory that's always taken up just by the fact that it's a class and that's before it even has its own specific values inside of there.
- Primitives provide efficiency
  - Lightweight
  - Cannot expose fields or methods
  - Cannot be treated as object