FAQ's on Strings, StringBuffer and StringTokenizer

1. What is the difference between String and String Buffer in java?

The main difference between String and StringBuffer in Java is that the String is immutable while StringBuffer is mutable. This means you can modify a StringBuffer object once you created it without creating any new object.

2. What is the difference between StringBuilder and StringBuffer in java?

Both StringBuffer and StringBuilder in Java have the same methods with one difference, and that is of synchronization.

StringBuffer is synchronized, which means it is thread-safe, and hence you can use it when you implement threads for your methods, whereas, StringBuilder is not synchronized, and it is not thread safe.

So, if you aren't going to use threading then use the **StringBuilder** class as it will be more **efficient** than **StringBuffer** due to the **absence** of **synchronization**.

3. How many ways are there to create a String?

In Java, you can create a string from byte array, char array, another string, from StringBuffer or from StringBuilder. Java String class provides constructor for all of these.

4. Why String is immutable in java? Mention at least two reasons?

String has been widely used as parameter for many java classes, e.g. for opening network connection you can pass hostname and port number as string:

- You can pass database URL as string for opening database connection.
- You can open any file by passing name of file as argument to File I/O classes.

Since string is immutable, it can be safely shared between many threads. This is very important for multi threaded programming. String immutability also helps in avoiding any synchronization issues in java.

5. State the difference between new operator and without new operator in string?

String string=”talent sprint”

Whenever we create an object of String object with double quote it allocates the memory in string literal pool in the indexing manner.

String string=new String(“talent sprint”);

Whenever we create the String object with new operator it allocates memory for String object in heap area in the indexing manner.

6. What is String Constant Pool Area or String Literal Pool?

It is a separate block of memory where the string objects are held by JVM. If a string object is created directly, using assignment operator, then it is stored in string constant pool area. For example:

```
public class Program{
    public static void main(String[] args){
       String str1 = "Hello";
       String str2 = "Hello";
       System.out.print(str1 == str2);
    }
}
The result is: true
```

7. Predict the output of the following code snippet? Explain why?

```
class StringEx{
     public static void main(String args[]) {
          String s1=”talent sprint”;
          String s2=new String(“talent sprint”);
          if(s1 == s2){
               System.out.println(“Strings  s1 and s2 both
     are same”);
          }
          else{
               System.out.println(“Strings  s1 and s2 both
     are  not same”);
          }
     }//main()close
} //StringEx close
```

Output: Strings s1 and s2 both are not same

Reason: The “==” operator always checks the address of the objects. As, s1 and s2 has different addresses, it only executes else block.

8. What is the difference between equals() and "==" in java?

equals()	"==" operator
It is a method, can only applied on any java object by passing another java object.	It is a comparison operator and it is applicable for both primitive and object type also.
Always it compares content present the objects	When we apply on primitive types it compares values of primitive type. and when we apply on object type it compares address of the objects.

9. Compile and run the below code? Provide the reason for the output that it gives?

```
package com.talentsprint.strings;
public class StringBufferDemo {
      public static void main(String[] args) {
            StringBuffer sb1 = new StringBuffer("talent");
            StringBuffer sb2 = new StringBuffer("talent");
            System.out.println("using == :" + (sb1 == sb2));
            System.out.println("using equals() method: " +
sb1.equals(sb2));

      }
}
 Prints: using ==: false
         using equals() method: false
```

Reason:

Using "==": It always compares the address of the objects so here we have used the new operator to create string object it creates two different objects with two different address.

Using equals() method: If we apply equals() method on StringBuffer objects it always prints false because, in StringBuffer class they did not override equals() method. So it uses Object class equals() method.

10. Compile and run the below code? Provide the reason for the output that it gives?

```
package com.talentsprint.strings;

public class StringBufferDemo {
      public static void main(String[] args) {
                  Test t1=new Test();
                  System.out.println(t1);
                  String s2="talent";
                  System.out.println(s2);
      }
}
```

Output: com.talentsprint.strings.Test@1ad086a
Talent

Reason: In Test class they didn't override the toString() method in println() method it uses Object class toString() method hence it produces hash code of Test class.

Where as in String class they override the toString() method to return the String.

11. What is the output of the following code? Provide the reason for the output that it gives?

```
Class Test{
      Public static void main(String ar[]){
            Integer i1=new Integer(123);
            String s1=new String(i1);
            System.out.println(s1);
      }
}
```

Output: compilation error, because there is no constructor in String class accepting to Integer object.

12. What is the output of the following code? Provide the reason for the output that it gives?

"Factory Method" is a static method of a class, which produces an object of the same class or at least its child class:

```
class Ex{
      public static void main(String ar[]){
            String s1=new String();
            String s2=new String("");
            if(s1.equals("")){
                  System.out.println("s1 and s2 are same");
            }
            else{
                  System.out.println("s1 and s2 are not same");
            }
      }
}
```

Output: s1 and s2 are same. Because, s1 holding the empty string object is exactly matches with the object String s1=new String("");

13. What is the output of the following code?

Sample code1:

```
class Ex{
```

```
        public static void main(String ar[]){
              String s1=null;
              if(null==s1){
                    System.out.println("s1 is having null");
              }
              else{
                    System.out.println("s1 is not having null");
              }
        }
  }
```

Sample code2:

```
  class Ex1{
        public static void main(String ar[]){
              String s1=null;
              if(null.equals(s1)){
                    System.out.println("s1 is having null");
              }
              else{
                    System.out.println("s1 is not having null");
              }
        }
  }
```

Output of sample Code 1: It executes the 'if' block because s1 can be initialized with the value null and == operator compares the content present in the s1 with null both are same. Hence it executes 'if' block and prints "s1 is having null".

Output of Sample Code 2: It gives compilation error mentioned that "null cannot be dereference ".

14. What is the functionality of java.lang.String.intern()?

The *java.lang.String.intern()* returns an interned string, that is, one that has an entry in the global string pool. If the string is not already in the global string pool, then it will be added.

```
  public class Program{
        public static void main(String[] args){
           // Create three strings in three different ways.
              String s1 = "Hello";
              String s2 = new
              StringBuffer("He").append("llo").toString();
              String s3 = s2.intern();
           // Determine which strings are equivalent using the ==
           // operator
              System.out.println("s1 == s2? " + (s1 == s2));
              System.out.println("s1 == s3? " + (s1 == s3));
       }
  }
```

The output is: s1 == s2? False
s1 == s3? True

15. What is the output of the following?

```
package testPackage;
class Test {
      public static void main(String[] args) {
            String hello = "Hello", lo = "lo";
          System.out.print((hello == "Hello") + " ");
          System.out.print((Other.hello == hello) + " ");
          System.out.print((other.Other.hello == hello) + " ");
          System.out.print((hello == ("Hel"+"lo")) + " ");
          System.out.print((hello == ("Hel"+lo)) + " ");
          System.out.println(hello == ("Hel"+lo).intern());
      }
}

class Other {
      static String hello = "Hello";
}
package other;
public class Other {
      static String hello = "Hello";
}
```

Produces the output:

```
true true true true false true
```

This example illustrates six points:

- Literal strings within the same class in the same package represent the references to the same String object.
- Literal strings within different classes in the same package represent the references to the same String object.
- Literal strings within different classes in different packages likewise represent the references to the same String object.
- Strings computed by constant expressions are computed at compile time and then treated as if they were literals.
- Strings computed by concatenation at run time are newly created and therefore distinct.

16. Can we pass more than one delimiter to a StringTokenizer?

Yes. We can pass any number of delimiters as an argument to a String Tokenizer. In order to work this we have to separate each delimiter by "//" (double forward slash) or directly we can represent them.

Example:

```
package com.talentsprint.stringtokenizer;
import java.util.Scanner;
import java.util.StringTokenizer;
public class Tokens {
 public static void main(String[] args) {
      String sentence="10,10@10@10,10.10/10&10";
      String temp;
      int k, total = 0;
      StringTokenizer s1 = new
      StringTokenizer(sentence,",/.@&");
     //StringTokenizer s1 = new
      StringTokenizer(sentence,",/////.//@//&");
      System.out.println("Total Number of tokens:" +
s1.countTokens());
      while (s1.hasMoreTokens()) {
            temp = s1.nextToken();
            k = Integer.parseInt(temp);
            total += k;
            System.out.print(k + "\t");
      }
      System.out.println("Sum of tokens :" + total);
 }
}
```

Output:

Total Number of tokens: 8

10 10 10 10 10 10 10 10

Sum of tokens: 80

Note: We can also pass "/" (single slash as a delimiter to the StringTokenizer) followed by "//".

17. From the following input string: "this is apple 10 this is apple 20 this is apple 30 this is apple 40" Print the following output.

This is apple
This is apple
This is apple
This is apple

Program:

```
public class Tokens {
      public static void main(String[] args) {
            String sentence=" this is apple 10 this is apple 20
this is apple 30 this is apple 40";
            String arr[]=sentence.split("\\d+"); /* "\" for
      representing the \ d (d for digits) and + to
      allow  multiple numbers in regular expression */
            for(String token: arr)
```

```
                System.out.println(token);
        }
    }
```

18. What is the difference between declaring a variable and defining a variable?

In declaration we just mention the type of the variable and its name. We do not initialize it. But defining means declaration + initialization.

Example: String s; It is just a declaration while String s = new String ("abcd"); Or String s = "abcd"; is the definition.

19. What is the default value of an object reference declared as an instance variable?

The default value will be null unless we define it explicitly.

20. What is the difference between the String and StringBuffer classes?

String objects are constants whereas, StringBuffer objects are not constants.

21. If a variable is declared as private, from where the variable can be accessed?

A private variable may only be accessed within the class in which it is declared.

22. Is "abc" a primitive value?

The String literal "abc" is not a primitive value. It is a String object.

23. What happens when you add a double value to a String?

When we add a double value to a string, the result is a String object.

24. What is your platform's default character encoding?

If you are running Java on English Windows platforms, it is probably Cp1252. If you are running Java on English Solaris platforms, it is most likely 8859_1.