LAB # 01

INTRODUCTION TO STRING POOL, LITERALS, AND WRAPPER CLASSES

OBJECTIVE: To study the concepts of String Constant Pool, String literals, String immutability and Wrapper classes.

In Java, String is the most important topic. There is a number of concepts related to String but the string pooling concept is one of them.

➢ String Constant Pool

A string pool is nothing but a storage area in a Java heap where string literals are stored. It is also known as String Intern Pool or String Constant Pool. It is just like object allocation. By default, it is empty and privately maintained by the Java String class.

The JVM performs some steps during the initialization of string literals that increase the performance and decrease the memory load. To decrease the number of String objects created in the JVM the String class keeps a pool of strings.

When we create a string literal, the JVM first checks that literal in the String pool. If the literal is already present in the pool, it returns a reference to the pooled instance. If the literal is not present in the pool, a new String object takes place in the String pool.

- **Creating String in Java**

There are two ways to create a string in Java:

1. **Using String Literal**

```
String str1 = "Python";

String str2 = "Data Science";

String str3 = "Python";
```

2. **Using new Keyword**

In Java, a new keyword is also used to create String, as follows:

```
String str1 = new String ("Java");

String str2 = new String ("C++");

String str3 = new String ("Data Science");
```

Let's understand what is the difference between them. Let's compare the string literals' references.

```
s1==s3

s2==s3
```

Let's see how we found that equal or not.

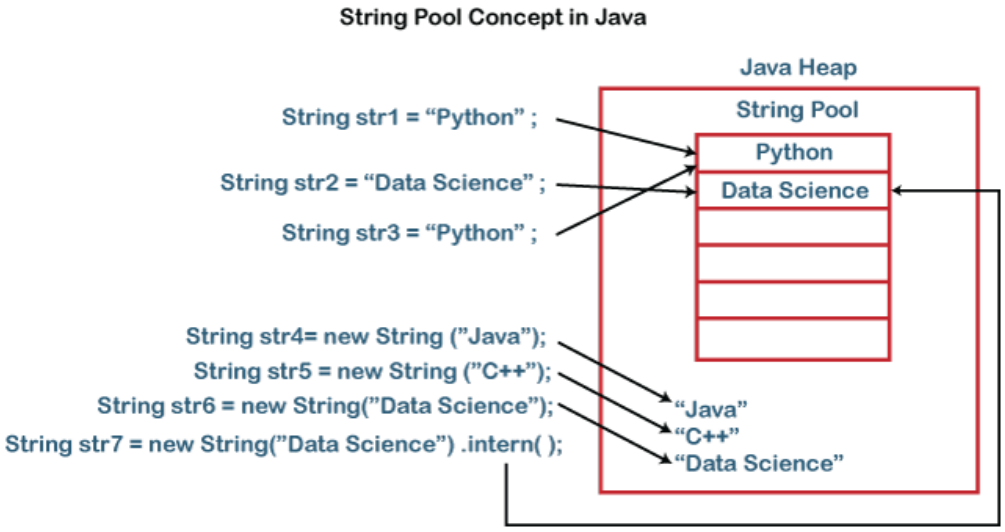

Fig.1 String Constant Pool concept in JAVA

Sample Program#1

```
public class StringPoolExample  {
public static void main(String[] args)  {
String s1 = "Java";
String s2 = "Java";
String s3 = new String("Java");
String s4 = new String("Java").intern();
System.out.println((s1 == s2)+", String are equal."); // true
```

```
System.out.println((s1 == s3)+", String are not equal."); // false
System.out.println((s1 == s4)+", String are equal."); // true
}
}
```

Output

```
true, String are equal.
false, String are not equal.
true, String are equal.
```

In the above example, we have seen that whenever we use a 'new' operator to create a string, it creates a new string object in the Java heap. We can forcefully stop this feature by using the intern() method of the String class.

- **Java String.intern() Method**

The String.intern() Method puts the string in the String pool or refers to another String object from the string pool having the same value. It returns a string from the pool if the string pool already contains a string equal to the String object. It determines the string by using the String.equals(Object) method. If the string is not already existing, the String object is added to the pool, and a reference to this String object is returned.

Syntax:

```
public String intern()
```

For example:

```
String str1 = new String ("Javaprogram");
```

The above statement creates the string in the Java heap. If we want to store the string literal in the String pool we should use the intern() method.

```
str1.intern(); or String s1 = new String("Javaprogram").intern();
```

Sample Program#2

```
public class StringInternExample {
```

```
public static void main(String args[]) {
String str1 = "Python";
String str2 = "Data Science";
String str3 = "Python";
String str4 = "C";
String str5 = new String ("Java");
String str6 = new String ("C++");
String str7 = new String ("Data Science");
String str8 = new String ("C").intern();
System.out.println((str1 == str5)+", Strings are not equal.");
System.out.println((str2 == str7)+", Strings are not equal.");
System.out.println((str4 == str8)+", Strings are equal.");
} }
```

Output

```
false, Strings are not equal.
false, Strings are not equal.
true, Strings are equal.
```

➢ Why Java Strings are Immutable?

These are some more reasons for making String immutable in Java. These are:

- The String pool cannot be possible if String is not immutable in Java. A lot of heap space is saved by 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 immutable.
- If we don't make the String immutable, it will pose a serious security threat to the application. For example, database usernames, and passwords are passed as strings to receive database connections.
- 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.

Before proceeding further with the fuss of *immutability*, let's just take a look into the String class and its functionality a little before coming to any conclusion.

This is how a String works:

```
String str = "knowledge";
```

This, as usual, creates a string containing “knowledge” and assigns it to reference *str*. Simple enough? Let us perform some more functions:

```
// assigns a new reference to the
// same string "knowledge"
String s = str;
str = str.concat(" base");
```

This appends a string “base” to *str*. But wait, how is this possible, since String objects are immutable? Well to your surprise, it is.

When the above statement is executed, the VM takes the value of String *str*, i.e. “knowledge” and appends “base”, giving us the value “knowledge base”. Now, since Strings are immutable, the VM can’t assign this value to *str*, so it creates a new String object, gives it a value “knowledge base”, and gives it reference *str*.

An important point to note here is that, while the String object is immutable, its reference variable is not. So that’s why, in the above example, the reference was made to refer to a newly formed String object.

At this point in the example above, we have two String objects: the first one we created with value “knowledge”, pointed to by s, and the second one “knowledge base”, pointed to by *str*. But technically, we have three String objects, the third one being the literal “base” in the *concat* statement.

Sample program#3

```
import java.io.*;

class GFG {
    public static void main(String[] args)
    {
        String s1 = "java";
        s1.concat(" rules");

        // Yes, s1 still refers to "java"
        System.out.println("s1 refers to " + s1);
    }
}
```

Output

```
s1 refers to java
```

➢ String Literals and Wrapper Classes

- **LITERALS**

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable.

Example

```
byte a = 68;
char a = 'A'
```

Types of Literals in Java

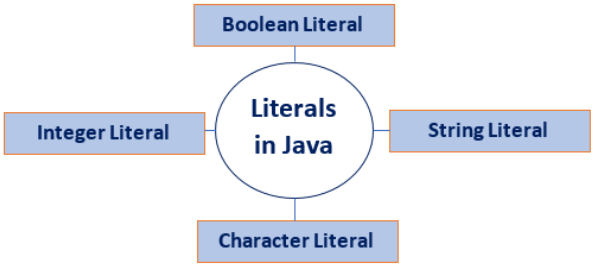

Fig.2 Types of String Literals

How to use literals?

A literal in Java can be identified with the prefix =, followed by a specific value.

Let's create a Java program and use above mentioned java literals.

Sample Program#4

```
public class LiteralsExample {
public static void main(String args[]){
int count = 987;
float floatVal = 4534.99f;
double cost = 19765.567;
```

```
int hexaVal = 0x7e4;
int binary = 0b11010;
char alpha = 'p';
String str = "Java";
boolean boolVal = true;
int octalVal = 067;
String stuName = null;
char ch1 = '\u0021';
char ch2 = 1456;
System.out.println(count);
System.out.println(floatVal);
System.out.println(cost);
System.out.println(hexaVal);
System.out.println(binary);
System.out.println(alpha);
System.out.println(str);
System.out.println(boolVal);
System.out.println(octalVal);
System.out.println(stuName);
System.out.println(ch1);
System.out.println("\t" +"backslash literal");
System.out.println(ch2);
}
}
```

Output

```
987
4534.99
19765.567
2020
26
p
Java
true
55
null
!
        backslash literal
?
```

- **WRAPPER CLASSES**

A wrapper class is an object that encapsulates a primitive type. Each primitive type has a corresponding wrapper:

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Fig.3 Corresponding Wrapper classes of primitive types

Each wrapper class has Object as a superclass. Byte, Short, Integer, Long Float and Double have Number as their direct superclass. This means that each wrapper class can implement the methods of the Object class such as hashCode(), equals(Object obj), clone(), and toString().

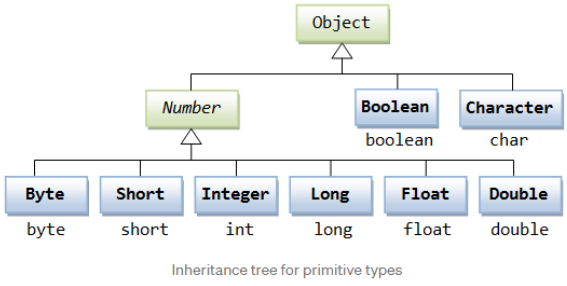

Fig.4 Tree representation of primitive types

Wrapper Objects are Immutable

All primitive wrapper objects in Java are final, which means they are immutable. When a wrapper object gets its value modified, the compiler must create a new object and then reassign that object to the original.
Sometimes you must use wrapper classes, for example when working with Collection objects, such as ArrayList, where primitive types cannot be used (the list can only store objects):

Example

```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid

ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

Creating Wrapper Objects

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

Sample Program #5

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Output

```
5
5.99
A
```

LAB TASKS

1. Write a program that initialize five different strings using all the above mentioned ways, i.e.,

   a) string literals

   b) new keyword

also use intern method and show string immutability.

2. Write a program to convert primitive data type Double into its respective wrapper object.

3. Write a program that initialize five different strings and perform the following operations.

   a. Concatenate all five stings.
   b. Convert fourth string to uppercase.
   c. Find the substring from the concatenated string from 8 to onward

4. You are given two strings word1 and word2. Merge the strings by adding letters in alternating order, starting with word1. If a string is longer than the other, append the additional letters onto the end of the merged string. Return *the merged string.*

Example:

Input: word1 = "abc", word2 = "pqr"
Output: "apbqcr"
Explanation: The merged string will be merged as so:
word1: a b c
word2: p q r
merged: a p b q c r

5. Write a Java program to find the minimum and maximum values of Integer**,** Float**,** and Double using the respective wrapper class constants.

.

HOME TASKS

1. Write a JAVA program to perform Autoboxing and also implement different methods of wrapper class.
2. Write a Java program to count the number of even and odd digits in a given integer using Autoboxing and Unboxing.
3. Write a Java program to find the absolute value, square root, and power of a number using Math class methods, while utilizing Autoboxing and Wrapper classes**.**
4. Write a Java program to **reverse only the vowels** in a string.
5. Write a Java program to **find the longest word** in a sentence.