Java & JEE Fast Track Training

Day 5 – Handling Strings in Java

MindsMapped Consulting

Memory Allocation & Garbage Collection

MindsMapped Consulting

Memory Allocation in Java and Garbage Collection

- **Java Heap Space**
  - Used by Java runtime to allocate memory to Objects and JRE classes.
  - Any new Object is always created in Heap Space.
  - Garbage Collection runs on the heap memory to free the memory used by objects that doesn't have any reference.
  - All instance and class variables are also stored in the heap.

- **Java Stack Memory**
  - Used for execution of a thread.
  - Store method specific values, and "references" to Objects being used in the method.
  - Stack memory is LIFO (Last-In-First-Out)
  - Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as method ends, the block becomes unused and become available for next method.
  - Stack memory size is very less compared to Heap memory.

Memory Allocation in Java and Garbage Collection

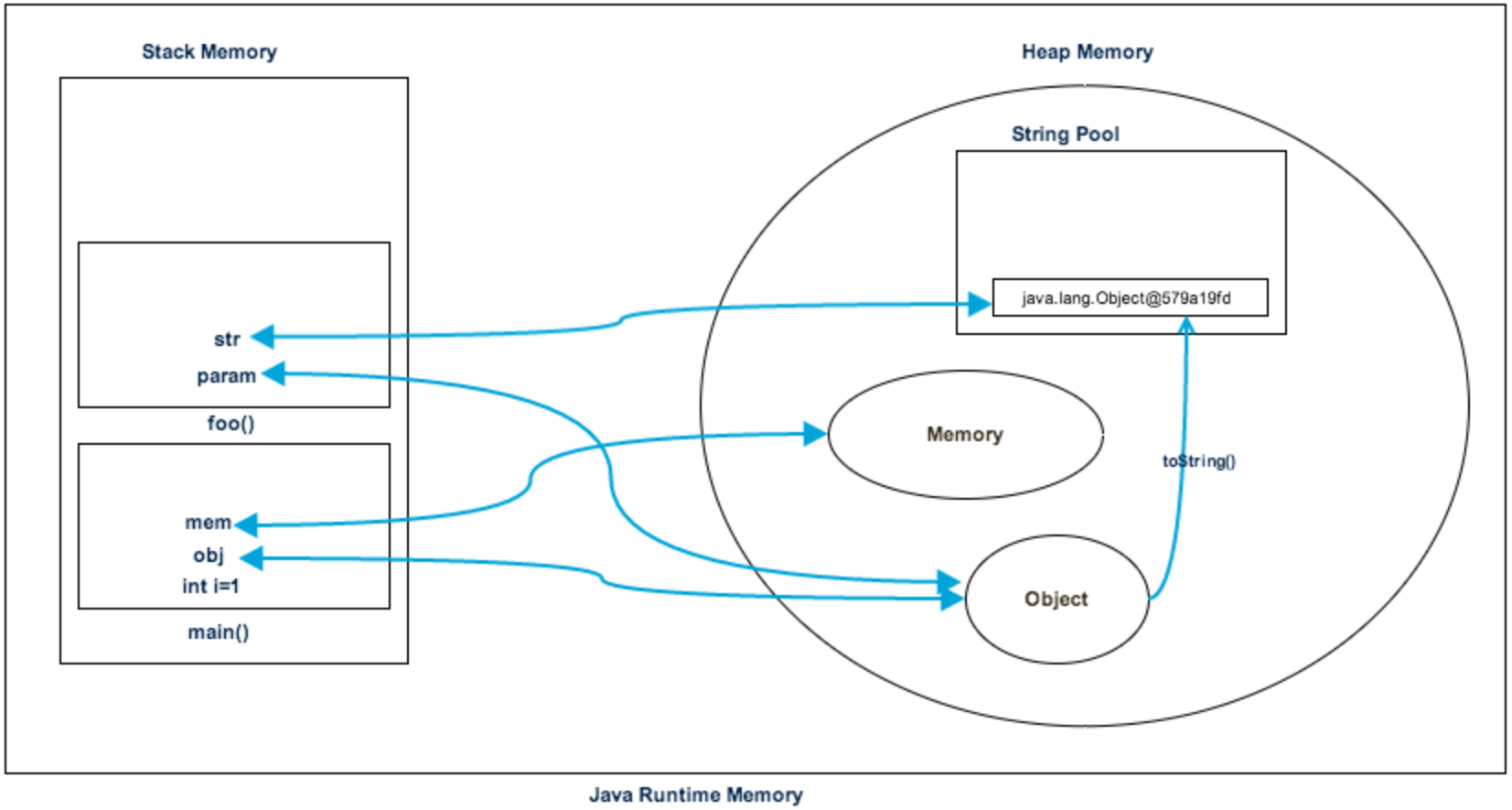
```
public class Memory {

        public static void main(String[] args) { // Line 1
                int i=1; // Line 2
                Object obj = new Object(); // Line 3
                Memory mem = new Memory(); // Line 4
                mem.foo(obj); // Line 5
        } // Line 9

        private void foo(Object param) { // Line 6
                String str = param.toString(); //// Line 7
                System.out.println(str);
        } // Line 8

}
```

Memory Allocation in Java and Garbage Collection



Strings in Java

MindsMapped Consulting

Creating String in Java

There are two ways to create a String in Java

- String literal

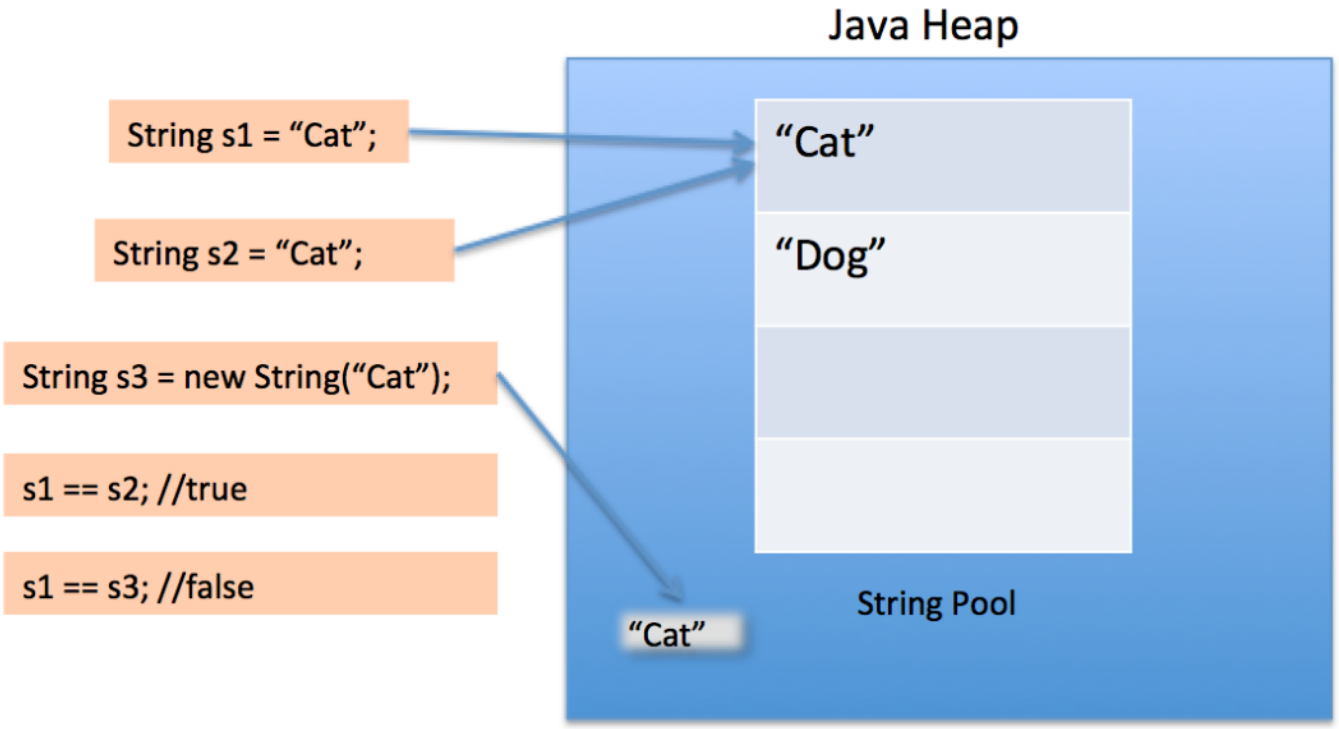
```
String str1 = "Welcome";
String str2 = "Welcome";
```

- Using “new” keyword

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

Does it make any difference? Well, yes!

String Pool Concept in Java (String Interning)



- String Intern Pool maintained in Java Heap Space

String Pool Concept in Java (String Interning)

- String is immutable in Java
- All Strings are stored in String Pool (also called String Intern Pool) allocated within Java Heap Space
- It is implementation of String Interning Concept.
- **String interning** is a method of storing only one copy of each distinct string value, which must be immutable.
- Interning strings makes some string processing tasks **more time- or space-efficient** at the **cost of requiring more time** when the string is created or interned.
- The distinct values are stored in a **string intern pool**.
- String pool is an example of **Flyweight Design Pattern.**
- Using new operator, we force String class to create a new String object in heap space.

Discussion: How many Strings are getting created here?

```
String str = new String("Cat");
```

java.lang.String API – Important methods

public class String		
	String(String s)	*create a string with the same value as* `s`
int	length()	*number of characters*
char	charAt(int i)	*the character at index* `i`
String	substring(int i, int j)	*characters at indices* `i` *through* `(j-1)`
boolean	contains(String substring)	*does this string contain* `substring`*?*
boolean	startsWith(String pre)	*does this string start with* `pre`*?*
boolean	endsWith(String post)	*does this string end with* `post`*?*
int	indexOf(String pattern)	*index of first occurrence of* `pattern`
int	indexOf(String pattern, int i)	*index of first occurrence of* `pattern` *after* `i`
String	concat(String t)	*this string with* `t` *appended*
int	compareTo(String t)	*string comparison*
String	toLowerCase()	*this string, with lowercase letters*
String	toUpperCase()	*this string, with uppercase letters*
String	replaceAll(String a, String b)	*this string, with* `a`*s replaced by* `b`*s*
String[]	split(String delimiter)	*strings between occurrences of* `delimiter`
boolean	equals(Object t)	*is this string's value the same as* `t`*'s?*
int	hashCode()	*an integer hash code*

java.lang.String API – Examples

```
String a = new String("now is");
String b = new String("the time");
String c = new String(" the");
```

instance method call	*return type*	*return value*
a.length()	int	6
a.charAt(4)	char	'i'
a.substring(2, 5)	String	"w i"
b.startsWith("the")	boolean	true
a.indexOf("is")	int	4
a.concat(c)	String	"now is the"
b.replace("t","T")	String	"The Tim"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false

Converting String to numbers and vice versa

- **String to Number**

  int i = Integer.parseInt(*str*);

  Integer i = Integer.valueOf(*str*);

  double d = Double.parseDouble(*str*);

  Double d = Double.valueOf(*str*);

  Note: Both throw ***NumberFormatException*** *If the String is not valid for conversion*

- **String to Boolean**

  boolean b = Boolean.parseBoolean(str);

- **Any Type to String**

  String s = String.valueOf(*value*);

String vs StringBuffer/StringBuilder

	String	*StringBuffer*	*StringBuilder*
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Performance	Fast	Very slow	Fast

Prefer StringBuffer or StringBuilder when performing multiple concatenations in your code.

Exercise: What is the output?

```
public static String someMethod( String str) //”Hello  “
{
  if( str == null)
    return null;
  int len = str.length();
  for( ; len > 0; len--)
  {
    if( ! Character.isWhitespace( str.charAt( len - 1)))
       break;
  }
  return str.substring( 0, len);
}
```

Exercise

- Write a program to input 2 strings – string1 and string2, and concatenate the first 5 characters of string1 with last 5 characters of string2.

Exercise

- Write a program to input 3 strings and check if at least any two strings are equal.

Exercise

- Write a program to sort an array of Strings in ascending order.