String Handling

Dr. Partha Pratim Sarangi
School of Computer Engineering

Outline

1. Fundamentals of Characters and Strings
2. Class String: String Constructors
3. String Methods: String extractions, string comparison, Searching strings, modifying a string, ToString() and valueOf() methods
4. Class StringBuffer
   - StringBuffer Constructors
   - StringBuffer Methods: length, capacity, setLength, ensureCapacity
   - StringBuffer Methods: charAt, setCharAt, getChars, and reverse
   - StringBuffer Methods: append, insertion, deletion
5. Class StringTokenizer

Fundamentals of Characters and Strings

- Characters
  - “Building blocks” of non-numeric data
  - ‘A’, ‘a’, ‘5’, ‘@’

- String
  - Sequence of characters treated as single unit or literal
  - May include letters, digits, etc.
  - In java String is a non-primitive datatype or reference datatype
  - Strings are objects of predefined class String
  - Unlike C/C++, Strings in Java do not have a null character at the end

Introduction

- In Java, Strings are a sequence of Unicode characters
- String is a predefined class present in java.lang package
- It is a final class, that means String class cannot be inherited
- String is a child class of an Object class
- It implements the CharSequence, Comparable, and Serializable interfaces
- String constants or iterals are stored in the heap area named as “String Constant Pool” (SCP) or “String Literal Pool” (SLP)
- String objects are immutable

Declaration of String class

- **public final class String extends Object implemts CharSequence, Serializable, Comparable** { … }
- String class provides nine constructors
- Null constructor String( ) has no characters and a length of zero
- String(array, start index, number of characters)
- String may be constant sequence of character or literal, array of characters, and array of bytes
  - String s = “Hello”; //Literal “Hello” is stored in String Constant Pool and referred by reference variable s
  - String s1 = new String(“Hello”); // Literal “Hello” is stored in heap memory and reference variable s1 is stored in stack memory

```
class MyStringDemo
{
        public static void main(String args[])
        {
                char charArray[] = {'K','I','I','T'};
                byte byteArray[] = {(byte)'K',
                 (byte)'I', (byte)'I', (byte)'T'};

          String s = new String();
         //String default constructor
         //instantiates empty string

        String s1 = new String(charArray);
        String s2 = new String(charArray,1,3);
        String s3 = new String(byteArray);
        String s4 = new String(byteArray,1,3);
```

```
String s5 = s1.concat(" JAVA");
//Concatenate a string to a string constant
 //and stored in another string variable
String s6 = s1+" JAVA";

//Concatenate a string to a string constant
//and stored in another string variable

                System.out.println(s);
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
                System.out.println(s4);
                System.out.println(s5);
                System.out.println(s6);
        }
}
```

Creating a String in Java

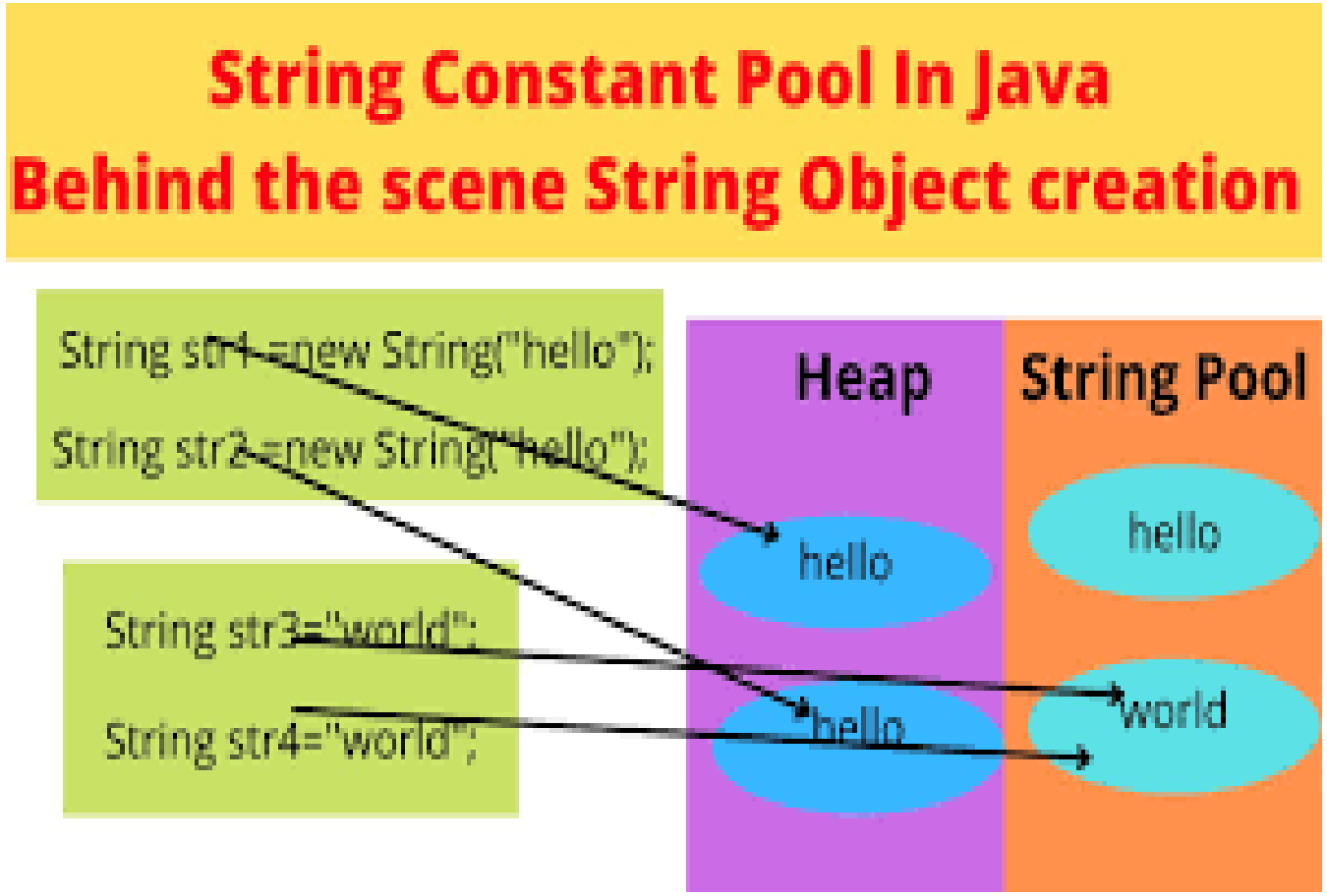
- There are two ways to create a String in Java
  - ➢ String literal
  - ➢ Using new keyword

- <u>String literal</u>
- In java, Strings can be created using String literal: Assigning a String literal to a String instance:
  - String str1 = "Welcome";
  - String str2 = "Welcome";
- String constant pool is special memory location present in Heap area which is used to store String literal.

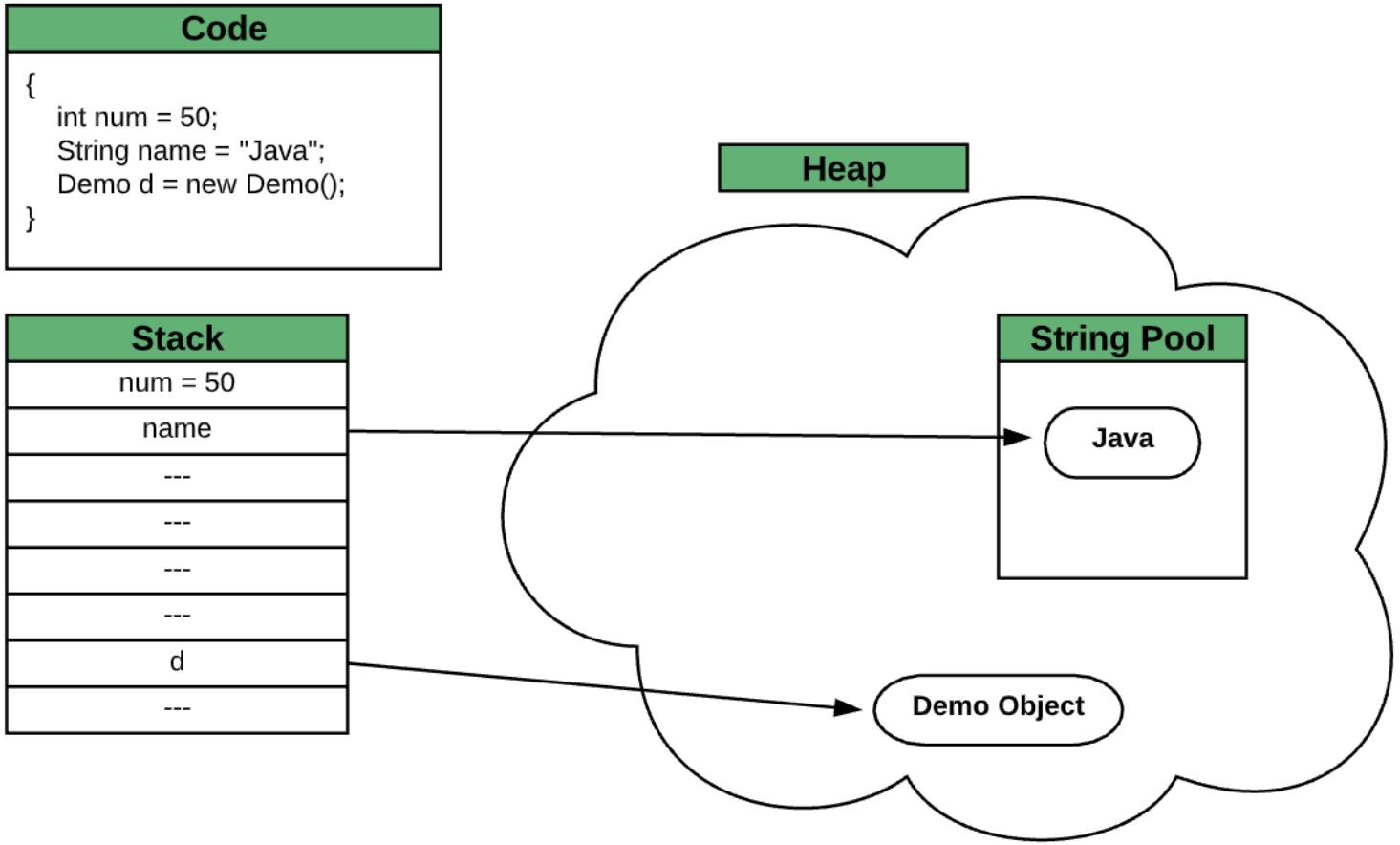
String Constant Pool or String Literal Pool

- The String constant pool is a special memory area in the heap memory. When we declare a String literal, the JVM creates the string object in the pool and stores its reference on the stack. Before creating each String object in memory, the JVM performs some steps to decrease the memory overhead.
- But if the string object already exist in the SCP JVM does not create a new object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created on string object (having the value “Welcome”) and assigned the same to both the instances. For example there are 10 string instances that have same value, it means that in memory there is only one object having the value and all the 10 string instances would be pointing to the same object.
- The variables created on the stack are deallocated as soon as the method completes execution. In contrast, a garbage collector reclaims the resources in the heap. Similarly, the garbage collector collects the un-referenced items from the pool.

How strings are stored in memory



Storing variables and objects in Java program



Some widely used String Methods

- length(): Returns the length of the string.
  - String str = "Hello, World!";
  - int len = str.length(); // len will be 13
- charAt(): Returns the character at the specified index in the string.
  - String str = "Hello, World!";
  - char ch = str.charAt(0); // ch will be 'H'
- substring(): Returns a new string that is a substring of the original string.
  - String str = "Hello, World!";
  - String subStr1 = str.subString(0, 5); // subStr1 will be "Hello"
  - String subStr2 = str.subString(7); // subStr2 will be "World!"

Contd…

- toUpperCase() and toLowerCase(): Returns a new string with all characters in upper or lower case.
  - String str = "Hello, World!";
  - String upperStr = str.toUpperCase(); // upperStr will be "HELLO, WORLD!"
  - String lowerStr = str.toLowerCase(); // lowerStr will be "hello, world!“
- The getChars() method in Java is used to extract a sequence of characters from a String and store them in a character array.
  - void getChars(int srcBegin, int srcEnd, char[] destination, int destBegin)
- The getBytes() method in Java is used to convert a String to an array of bytes using the platform's default character encoding.

```
String str = "Hello, World!";
// Get the byte representation of the string
byte[] bytes = str.getBytes();
// Print the byte representation of the string
for (byte b : bytes) {
    System.out.print(b + " ");
}
```

Contd…

- equals() and equalsIgnoreCase(): Compares the current string with another string and returns true if they are equal.
- startsWith() and endsWith(): Returns true if the current string starts or ends with the specified character or sequence.
  - String str = "Hello, World!";
  - boolean result1 = str.startsWith("Hello"); // result1 will be true
  - boolean result2 = str.endsWith("World!"); // result2 will be true
- The indexOf() method in Java is used to find the index of the first occurrence of a specified substring within a String.
  - public int indexOf(char ch, int fromIndex)
- The concat() method in Java is used to concatenate one String to the end of another String.

Contd...

- The toCharArray() method in Java is used to convert a String to a character array.

```
String str = "Hello, World!";
char[] chars = str.toCharArray();
// Print the contents of the character array
for (char c : chars) {
    System.out.print(c + " ");
}
```

- The trim() method in Java is used to remove leading and trailing white space characters from a String.

```
String str = "   Hello, World!   ";
String trimmed = str.trim();
// Print the trimmed string
System.out.println(trimmed); // Output: "Hello, World!"
```

Contd...

- It's worth noting that the trim() method only removes white space characters from the beginning and end of the String. It does not remove white space characters that appear within the String. If you need to remove white space characters from within a String, you can use other methods such as replaceAll() or replace().
- contains(): Returns true if the current string contains the specified character or sequence.

```
String str = "Hello, World!";
boolean s1 = str.contains("Hello");
boolean s2 = str.contains("Java");
// Print the results
System.out.println(s1); // Output: true
System.out.println(s2); // Output: false
```

  - It's worth noting that the contains() method is case-sensitive, so it will only match sequences of characters that have the same case as the parameter. If you need to perform a case-insensitive search, you can convert both the String and the parameter to lowercase or uppercase using the toLowerCase() or toUpperCase() method before calling contains().

Java program that counts the number of vowels present in a given String

```
import java.util.Scanner;
public class CountVowels {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = input.nextLine();
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
                ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
                count++;
                str.setCharAt(i, Character.toUpperCase(ch));
            }
        }
        System.out.println("Number of vowels: " + count);
        System.out.println(“Modified String: " + str.toString());
    }
}
```

Contd…

- join(CharSequence delimiter, CharSequence... elements): Concatenates a sequence of strings with the specified delimiter.

```
String[] words = {"The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"};
String str = String.join(" ", words);
System.out.println(“Modified String: " + str.toString());
```

- valueOf() methods: The String class provides valueOf() methods to convert primitive data types into Strings.

```
int i = 42;
double d = 3.14159;
String str1 = String.valueOf(i); // "42"
String str2 = String.valueOf(d); // "3.14159"
```

StringBuffer Class

- The StringBuffer class provides for strings that will be modified; you use string buffers when you know that the value of the character data will change.
- A StringBuffer object is like a String, but it can be changed or modified.
- StringBuffer class is present in java.lang package.
- It is a final class and thus it cannot be inherited.
- A StringBuffer class in Java is a subclass of Java.lang.Object class and implements the interfaces, CharSequence and Serializable.
- An instance of StringBuffer class represents a string that can be dynamically modified.
- The objects of String class are immutable, whereas the objects of a String Buffer class are mutable.

Use the following guidelines for deciding which class to use:

- If your text is not going to change, use a **String class**.
- If your text will change, and will only be accessed from a single thread, use a **StringBuilder class**.
- If your text will change, but will be accessed from multiple threads, use a **StringBuffer class** because StringBuffer class is thread-safe.

Constructor of StringBuffer

StringBuffer ( )	This constructs an empty StringBuffer
StringBuffer (int capacity)	This constructs an empty StringBuffer with the specified intial capacity
StringBuffer (String s)	This constructs a StringBuffer that initially contains the special string

The StringBuilder class, introduced in JDK 5.0, is a faster, drop-in replacement for string buffers. You use a string builder in the same way as a string buffer, but only if it's going to be accessed by a single thread.

Comparison of String, StringBuffer and StringBuilder

String	StringBuffer	StringBuilder
Storage: Heap area SCP	Heap area	Heap area
Object: Immutable	Mutable	Mutable
Memory: If we change the value of String a lot of times, it will allocates more space.	Consumes less memory	Consumes less memory
Thread safe: Not thread safe	All methods are synchronized and thread safe	All methods are non-synchronized and not thread safe.
Performance: Slow	Fast as compared to String	Faster that StringBuffer

Methods of StringBuffer

- **public synchronized int length()** : This method returns the length of the StringBuffer
- **public synchronized int capacity()** : This method returns the capacity of the amount of space allocated rather than the amount of space used.
- **public synchronized void setLength(int length)** : This method is used to set the length of the StringBuffer.
- **public synchronized void ensureCapacity(int capacity)** : This method is used to set the capacity of the StringBuffer.
- **public synchronized char charAt(int index)** : This method returns a character from the StringBuffer.
- **public synchronized void getChars(int start, int end, char c[], int index)** : This method extracts more than one character from StringBuffer.

Contd...

- **public synchronized void setCharAt(int index, char ch)** : This method sets a character in the StringBuffer.
- **public synchronized StringBuffer append(Object o)** : This method calls toString() on Object o and appends the result to the current StringBuffer.
- **public synchronized StringBuffer append(String s)** : This method appends a string in the StringBuffer.
- **public synchronized StringBuffer append(StringBuffer sb)** : This method appends a StringBuffer object to the existing StringBuffer.
- **public synchronized StringBuffer append(char c)** : This method appends a character to the existing StringBuffer.
- **public synchronized StringBuffer delete(int index, int length)** : This method is used to delete more than one character from the StringBuffer.

Contd...

- **public synchronized StringBuffer deleteCharAt(int index)** : This method is used to delete a character from the StringBuffer.
- **public synchronized StringBuffer replace(int index, int length, String s)** : This method is used to replace a string in the StringBuffer.
- **public synchronized StringBuffer insert(int index, String s)** : This method is used to replace a string in the StringBuffer.
- **public synchronized StringBuffer reverse()** : This method is used to reverse the StringBuffer.
- **public String toString()** : This method is used to convert a string to a StringBuffer.