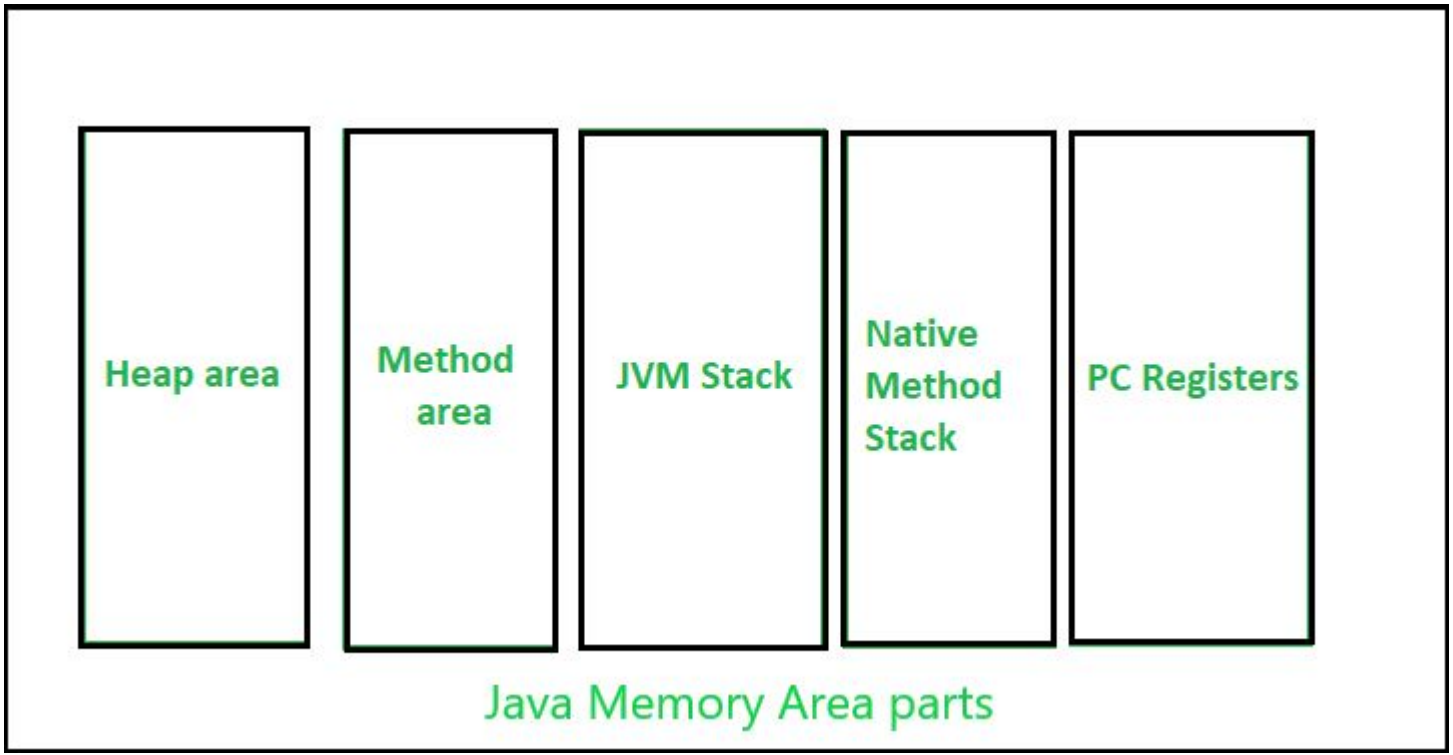
String Handling

Java Memory Management

Java Memory Management (Run-time memory areas in JVM)

- Heap Area
- Method Area
- JVM Stack
- Native Method Stack
- PC register

Java Memory Management



Java Memory Management (Run-time memory areas in JVM)

Heap Area

Used to store objects of classes and arrays. Heap memory is common and shared across multiple threads. This is where the garbage collector comes into picture. Heap area is created at the JVM startup.

Method Area

Method area is a logical part of heap area. This is an storage area for compiled class files. Method area has per class structures and fields. Nothing but static fields and structures. It also includes the method data, method and constructor code, **run-time constant pool.** Method area is created at JVM startup and shared among all the threads. JVM will throw OutOfMemoryError if the allocated memory area is not sufficient during the run-time.

Java Memory Management (Run-time memory areas in JVM)

JVM Stack: A stack is created at the same time when a thread is created and is used to store local variables, data and partial results. It contains references to heap objects.

Native Method Stack : Used for native methods, and created per thread. It helps in executing native methods (methods written in languages other than Java)

PC (Program Counter) register : Keeps track of the current instruction executing at any moment. It Stores the address of the next instruction to be executed.

** Static methods (in fact all methods) as well as static variables are stored in the PermGen (Permanent Generation) section of the heap. In java 8 “Metaspace” introduced in place of PermGen.

Shallow Comparision & Deep Comparision

Shallow Comparision : Comparision based on object references. This is the default behaviour of JVM while comparing two objects using equals( ) method.

Deep Comparision : Comparision based on data/state of the object. This we need to implement, if required.

Note: **Equal objects must produce the same hash code as long as they are equal, however unequal objects need not produce distinct hash codes.**

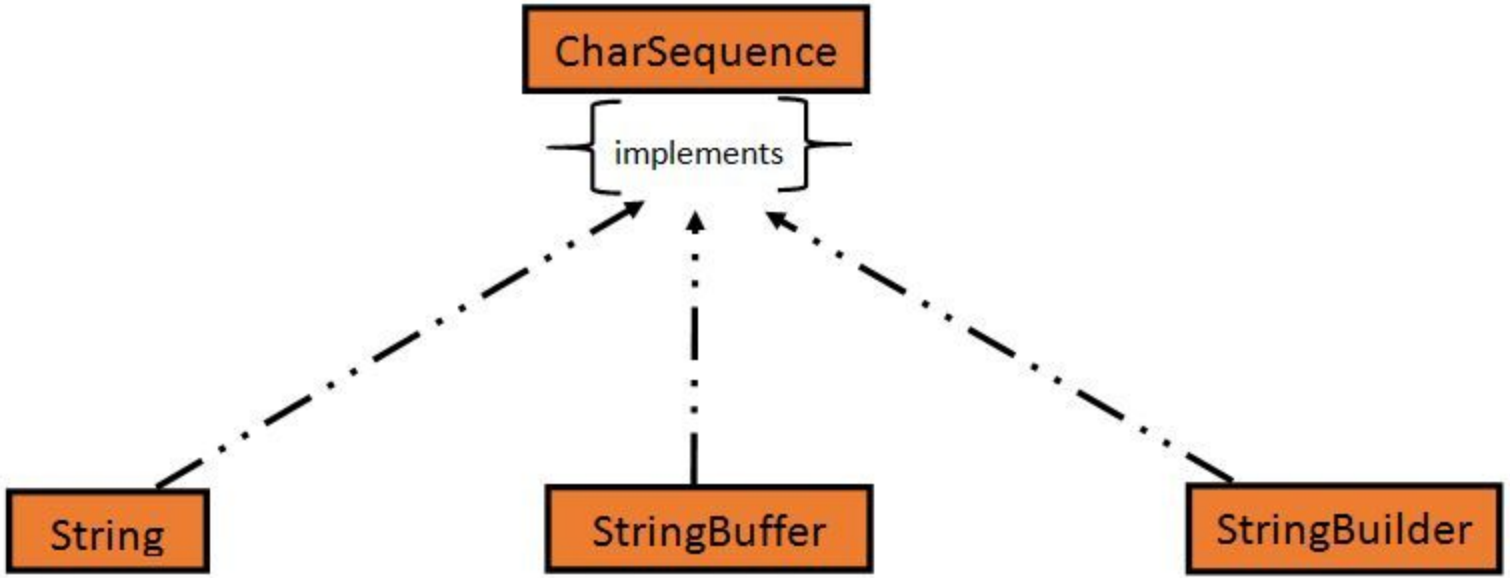
We can override equals() and hashCode() method to perform deep comparision. Remember if two objects are equals then it is our responsibility to keep their hash code also same.

== compares only references and not the object data/state.

String Handling

- Basically, string is a sequence of characters but it’s not a primitive type.
- In Java, CharSequence Interface is used for representing a sequence of characters.
- CharSequence interface is implemented by String, StringBuffer and StringBuilder classes. These three classes can be used for creating strings in java.

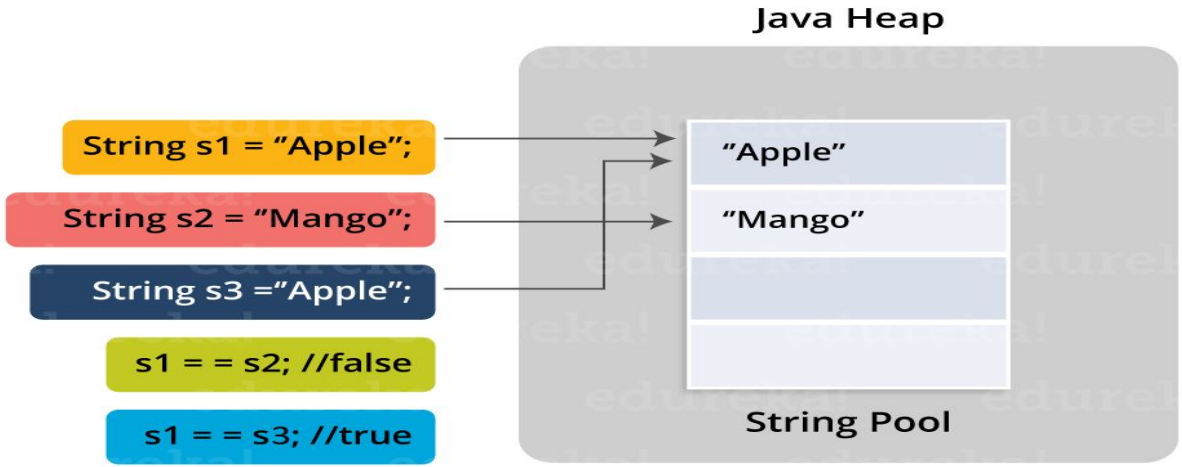
String


CharSequence
implements
String
StringBuffer
StringBuilder

String Handling

Creation of String using String literal (String Constant Pool)

String Constant Pool is a pool of Strings stored in Java heap memory

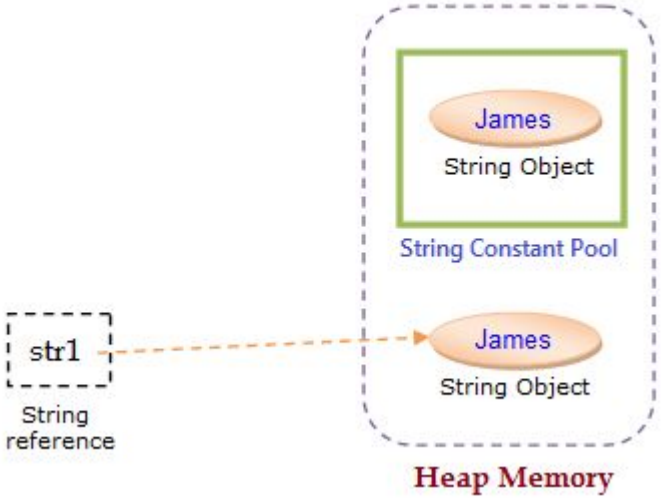


Note : == compares object references, it checks to see if the two operands point to the same object.

String Handling

Creation of String using new keyword

String str1 = new String(“James);



String Handling

Strings are immutable objects.

Once a String object is created, it can never be changed so it is an immutable object but its reference variable is not.

String str = “Hello Friends”;

str.concat(“ Welcome”);

// New String object is created with value Hello Friends Welcome

System.out.println(str); // will print - Hello Friends

String Handling

String Methods

public char charAt(int index)

Returns character at a particular index. Specified index value should be between '0' to 'length() -1'. It throws IndexOutOfBoundsException if index is invalid/ out of range.

public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

This method is used to copy set of characters of the invoking string into the specified character array.

public byte[] getBytes()

Returns sequence of bytes for the given string.

byte[]b=str.getBytes(); //byte array having all charactes with ascii values

String Handling

String Methods

boolean equals(Object anObject)

Used to compare two strings. It returns true if both string represents same sequence of characters else false.

String s1="hello";

String s2="welcome";

s1.equals(s2);// returns false

s1.equals("hello");// returns true.

boolean equalsIgnoreCase(String anotherString)

Used to compare two strings, ignoring the case(upper or lower case).

String Handling

String Methods

public char[] toCharArray()

This method converts this string to a new character array.

char[]ch=str.toCharArray();//array contains the all characters of the string

boolean startsWith(String prefix)

Checks if a string starts with the string represented by prefix.

String s1="refresh java";

s1.startsWith("refresh");// returns true.

String Handling

String Methods

public boolean endsWith(String suffix)

This method checks whether the String ends with a specified suffix.

String str1 = new String("This is a test String");

boolean var1 = str1.endsWith("String");

int compareTo(String str)

The Java String compareTo() method is used for comparing two strings lexicographically. Each character of both the strings is converted into a Unicode value for comparison. If both the strings are equal then this method returns 0 else it returns positive or negative value. The result is positive if the first string is lexicographically greater than the second string else the result would be negative.

String Handling

String Methods

substring() method

This method is used to get the substring of a given string based on the passed indexes.

public String substring(int beginIndex)

public String substring(int beginIndex, int endIndex) //end index is exclusive

"Welcome".substring(3) would return "come".

"Welcome".substring(3,5) would return "co".

String Handling

String Methods

concat() method

public String concat(String str)

This method concatenates the string str at the end of the current string. This method can be called multiple times in a single statement.

String str = “Hello”;

str = str.concat(“ Friends”).concat(“ ,Welcome”);

String Handling

String Methods

replace() method

String replace(char oldChar, char newChar)

It replaces all the occurrences of a oldChar character with newChar character.

For example, "jiya joy".replace('j', 't') would return tiya toy.

replaceFirst() method

String replaceFirst(String regex, String replacement)

It replaces the first substring that fits the specified regular expression with the replacement String.

String str = new String("cdac.in");

System.out.println(str.replaceFirst("in", "net")); //Print cdac.net

String Handling

String Methods

replaceAll() method

String replaceAll(String regex, String replacement)

It replaces all the substrings that fits the given regular expression with the replacement String.

String str = new String("My .in site is cdac.in");

System.out.println(str.replaceAll("in", "net"));

Output is My.net site is cdac.net

String Handling

String Methods

indexOf()method

This method is used to find the index of a specified character or a substring in a given String.

int indexOf(int ch) : It returns the index of the first occurrence of character ch in a given String.

int indexOf(int ch, int fromIndex) : It returns the index of the first occurrence of character ch in a given String. Starts searching from the 'from index'.

int indexOf(String str): It returns the index of the first occurrence of string str in a given String.

int indexOf(String str, int fromIndex) : It returns the index of the first occurrence of string str in a given String. Starts searching from the 'from index'.

Simailarly, there is a lastIndexOf( ) method which returns the index of the last occurrence of the character or string.

String Handling

String Methods

boolean contains() method

This method checks whether a particular sequence of characters is part of a given string or not.

Boolean isEmpty()method

This method checks whether a String is empty or not.

static String join(CharSequence delimiter, CharSequence... elements)

The first argument of this method specifies the delimiter that is used to join multiple strings.

String message = String.join("-", "This", "is", "a", "String");

Output is: "This-is-a-String"

String Handling

String Methods

split() method

This method used for splitting a String into its substrings based on the given delimiter or regular expression.

String[] split(String regex)

String[] split(String regex, int limit) // Would return only the array of strings specified by limit

Ex.

String str = new String("30/08/2020");

String array1[]= str.split("/");

String array2[]= str.split("/", 2);

String Handling

String Methods

split(String regex, int limit) method

String str = "pq@rccc@xy@zccc";

Limit = 0 : Splits as many times as possible. Discards trailing empty strings.

Limit < 0 : Splits as many times as possible. Does not discard trailing empty strings.

Limit > 0 : Splits limit-1 times

str5.split("c",0)	{ “pq@r" , " " , " " , "@xy@z" }
str5.split("c",-1)	{ "pq@r" , " " , " " , "@xy@z" , " " , " " , " " }
str5.split("@",3)	{"pq" , "rccc" , "xy@zccc"}

String Handling

String Methods

public static String format(String format, Object... args)

Returns a formatted string using the specified format string and arguments.

Java String Format Specifiers

%c – Character

%d – Integer

%s – String

%o – Octal

%x – Hexadecimal

%f – Floating number

%h – hash code of a value

String Handling

```
String str = "just a string";

//concatenating string using format

String formattedString = String.format("My String is %s", str);

// %.6f is for having 6 digits in the fractional part

String formattedString2 = String.format("My String is %.6f",12.121);
```

String Handling

We can specify the argument positions using %1$, %2$,..format specifiers. Here %1$ represents first argument, %2$ second argument and so on.

String str1 = "cool string";

String str2 = "88";

String fstr = String.format("My String is: %1$s, %1$s and %2$s", str1, str2);

System.out.println(fstr);

Result is:

My String is: cool string, cool string and 88

String Handling

Left Padding with Zero

```
int str = 88;

String formattedString = String.format("%05d", str);

System.out.println(formattedString);// 00088
```

String Handling

Left Padding with Zero

```
int str = 88;

String formattedString = String.format("%05d", str);

System.out.println(formattedString);// 00088
```

String Handling

Displaying String, int, hexadecimal, float, char, octal value using format() method

```
String str1 = String.format("%d", 15); // Integer value

String str2 = String.format("%s", "cdac.in"); // String

String str3 = String.format("%f", 16.10); // Float value

String str4 = String.format("%x", 189); // Hexadecimal value

String str5 = String.format("%c", 'P'); // Char value

String str6 = String.format("%o", 189); // Octal value
```

String Handling

StringBuffer Class

1. As we know that String objects are immutable, so if we do a lot of modifications to String objects, we may end up with a memory leak. To overcome this we use StringBuffer class.
2. Java StringBuffer class is used to create mutable (modifiable) string object.
3. StringBuffer class represents growable and writable character sequence. It is also thread-safe i.e. multiple threads cannot access it simultaneously.
4. Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

String Handling

Constructors of StringBuffer class

1. StringBuffer ( ) : Creates an empty string buffer with the initial capacity of 16.
2. StringBuffer (int capacity) : Creates an empty string buffer with the specified capacity as length.
3. StringBuffer (String str) : Creates a string buffer initialized to the contents of the specified string.
4. StringBuffer (charSequence[] ch) : Creates a string buffer that contains the same characters as the specified CharSequence.

String Handling

StringBuffer Methods

append() method

The append() method concatenates the given argument(string representation) to the end of the invoking StringBuffer object.

StringBuffer append(String str)

StringBuffer append(int n)

Ex.

```
StringBuffer strBuffer = new StringBuffer("Hello ");

strBuffer.append("Friends");

System.out.println(strBuffer);              //Print Hello Friends
```

String Handling

StringBuffer Methods

insert() method

The insert() method inserts the given argument(string representation) into the invoking StringBuffer object at the given position.

StringBuffer strBuffer=new StringBuffer("Core");

strBuffer.insert(1,"Java");

System.out.println(strBuffer); // CJavaore

String Handling

StringBuffer Methods

replace() method

The replace() method replaces the string from specified start index to the end index.

```
StringBuffer strBuffer=new StringBuffer("Core");

strBuffer.replace( 2, 4, "Java");

System.out.println(strBuffer);              //CoJava
```

String Handling

StringBuffer Methods

reverse() method

This method reverses the characters within a StringBuffer object.

```
StringBuffer strBuffer=new StringBuffer("Core");

strBuffer.reverse();

System.out.println(strBuffer);  // eroC
```

String Handling

StringBuffer Methods

delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
StringBuffer strBuffer=new StringBuffer("Core");
strBuffer.delete( 2, 4);
System.out.println(strBuffer);                    //Co
```

String Handling

StringBuffer Methods

capacity() method

The capacity() method returns the current capacity of StringBuffer object. The capacity is the amount of storage available for newly inserted characters, beyond which an allocation will occur.

```
StringBuffer strBuffer=new StringBuffer();

System.out.println(strBuffer.capacity());

strBuffer.append("1234");

System.out.println(strBuffer.capacity());

strBuffer.append("123456789112");

System.out.println(strBuffer.capacity());

strBuffer.append("1");

System.out.println(strBuffer.capacity()); //(oldcapacity*2)+2
```

String Handling

Difference between String and StringBuffer

Some of differences between String and StringBuffer are given below:

String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String are stored in Sting pool.	StringBuffers are stored in heap.
While performing concatenations, String is slow because you are actually creating new object(internally) every time since String is immutable.	StringBuffer is fast and consumes less memory when you cancat strings.
You can compare the contents of two strings by equals() method as String class overrides the equals() method of Object class.	StringBuffer class doesn't override the equals() method.

String Handling

StringBuilder Class

1. StringBuilder objects are like String objects, except that they can be modified. Hence Java StringBuilder class is also used to create mutable (modifiable) string object.
2. StringBuilder is same as StringBuffer except for one important difference. **StringBuilder is not synchronized, which means it is not thread safe.**
3. This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread.
4. Instances of StringBuilder are not safe for use by multiple threads. If such synchronization is required then it is recommended that StringBuffer be used.

String Handling

Constructors of StringBuilder class

1. StringBuilder ( ) : Constructs a string builder with no characters in it and an initial capacity of 16 characters.
2. StringBuilder (int capacity) : Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.
3. StringBuilder (String str) : Constructs a string builder initialized to the contents of the specified string. The initial capacity of the string builder is 16 plus the length of the string argument.

String Handling

Difference between StringBuffer and StringBuilder

Some of differences between StringBuffer and StringBuilder are given below:

StringBuffer	StringBuilder
StringBuffer is synchronized i.e. thread safe.	StringBuilder is non-synchronized i.e. not thread safe.
StringBuffer is less efficient and slower than StringBuilder as StringBuffer is synchronized.	StringBuilder is more efficient and faster than StringBuffer.
StringBuffer is old, its there in JDK from very first release.	StringBuilder is introduced much later in release of JDK 1.5