String Manipulation

Java String

Conceptually, Java strings are sequences of Unicode characters. String is basically an object that represents a sequence of char values. java.lang.String class is used to create a string object.

An array of characters works same as a java string. For example:

char[] ch={'j','a','v','a'};
String s=new String(ch);

is same as:
String s="java";
String e = ""; // an empty string

There are two ways to create String object:
By string literal: String s="java";
By new keyword : String s=new String(ch);

Java String literal is created by using double quotes.
The String class supports several constructors.
To create an empty String, call the default constructor.
For example,
String s = new String();
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);

String Manipulation

String class methods

The java.lang.String class provides many useful methods to perform operations on the sequence of char values.

1. **char charAt(int index)** returns char value for the particular index
2. **int length()** returns string length
3. **static String format(String format, Object... args)** returns formatted string
4. **static String format(Locale l, String format, Object... args)** returns formatted string with given locale
5. **String substring(int beginIndex)** returns substring for given begin index
6. **String substring(int beginIndex, int endIndex)** returns substring for given begin index and end index
7. **boolean contains(CharSequence s)** returns true or false after matching the sequence of char value
8. **static String join(CharSequence delimiter, CharSequence... elements)** returns a joined string
9. **static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)** returns a joined string
10. **boolean equals(Object another)** checks the equality of string with object
11. **boolean isEmpty()** checks if string is empty
12. **String concat(String str)** concatenates specified string
13. **String replace(char old, char new)** replaces all occurrences of specified char value
14. **String replace(CharSequence old, CharSequence new)** replaces all occurrences of specified CharSequence
15. **String trim()** returns trimmed string omitting leading and trailing spaces
16. **String split(String regex)** returns splitted string matching regex 17 String split(String regex, int limit) returns splitted string matching regex and limit

String Manipulation

17. **int indexOf(int ch)** returns specified char value index
18. **int indexOf(int ch, int fromIndex)** returns specified char value index starting with given index
19. **int indexOf(String substring)** returns specified substring index
20. **int indexOf(String substring, int fromIndex)** returns specified substring index starting with given index
21. **String toLowerCase()** returns string in lowercase.
22. **String toLowerCase(Locale l)** returns string in lowercase using specified locale.
23. **String toUpperCase()** returns string in uppercase.
24. **String toUpperCase(Locale l)** returns string in uppercase using specified locale.

Substrings

You can extract a substring from a larger string with the substring method of the String class.
For example,
String greeting = "Hello";
String s = greeting.substring(0, 3);
creates a string consisting of the characters "Hel".

The second parameter of substring is the first position that you do not want to copy. In our case, we want to copy positions 0, 1, and 2 (from position 0 to position 2 inclusive). As substring counts it, this means from position **0 inclusive** to position **3 exclusive**. There is one advantage to the way substring works: Computing the length of the substring is easy.

The **string** s.substring(a, b) always has length b - a.
For example, the substring "Hel" has length 3 – 0 = 3.

String Manipulation

Strings Are Immutable

In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. Once string object is created its data or state can't be changed but a new string object is created. The String class gives no methods that let you change a character in an existing string.

Concatenation

Java, like most programming languages, allows you to use + to join (concatenate) two strings.

```
String expletive = "Expletive";
String PG13 = "deleted";
String message = expletive + PG13; // "Expletivedeleted"
int age = 13;
String rating = "PG" + age; //sets rating to the string "PG13".
```

Testing Strings for Equality

To test whether two strings are equal, use the equals method. The expression

s.equals(t)

returns true if the strings s and t are equal, false otherwise. Note that s and t can be string variables or string constants.

```
String h = “Hello”;
String j = “Java”;
h.equals(j) //return false
"Hello".equals(j) //return false
"Hello".equalsIgnoreCase("hello") //return true
"Hello".equalsIgnoreCase(h) //return true
h.equalsIgnoreCase(“Hello”)// return true
```

String Manipulation

Do not use the **==** operator to test whether two strings are equal!
It only determines whether or not the strings are stored in the same location. Sure, if the strings are in the same location, they must be equal. But it is entirely possible to store multiple copies of identical strings in different places.

If the virtual machine always arranges for equal strings to be shared, then you could use the == operator for testing equality. But only string constants are shared, not strings that are the result of operations like + or substring. Therefore, never use == to compare strings lest you end up with a program with the worst kind of bug—an intermittent one that seems to occur randomly.
String h = “Hello”;
H == “Hello” //may be true

Empty and Null Strings
The empty string "" is a string of length 0. You can test whether a string is empty by calling
if (str.length() == 0)
or
if (str.equals(""))

However, a String variable can also hold a special value,called null, that indicates that no object is currently associated with the variable.
str = null;
To test whether a string is null, use the condition
if (str == null)
Sometimes, you need to test that a string is neither null nor empty. Then use the condition
if (str != null && str.length() != 0)

String Manipulation

int compareTo(String other): returns a negative value if the string comes before other in dictionary order, a positive value if the string comes after other in dictionary order, or 0 if the strings are equal.

toString():

If you want to represent any object as a string, toString() method comes into existence. The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of Java toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

For those cases in which a modifiable string is desired, Java provides two options: **StringBuffer and StringBuilder**. Both hold strings that can be modified after they are created.

The **String, StringBuffer,** and **StringBuilder** classes are defined in **java.lang**. Thus, they are available to all programs automatically. All are declared final, which means that none of these classes may be subclassed. This allows certain optimizations that increase performance to take place on common string operations. All three implement the **CharSequence** interface.

One last point: To say that the strings within objects of type String are unchangeable means that the contents of the **String** instance cannot be changed after it has been created. However, a variable declared as a **String** reference can be changed to point at some other String object at any time.

String Manipulation

StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Important Constructors of StringBuffer

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index):** is used to return the character at the specified position.
9. **public int length():** is used to return the length of the string i.e. total number of characters.

String Manipulation

10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
11. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

- **StringBuffer append() method**

  The append() method concatenates the given argument with this string.

```
class A{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}
```

- **StringBuffer insert() method**

  The insert() method inserts the given string with this string at the given position.

- **StringBuffer replace() method**

  The replace() method replaces the given string from the specified beginIndex and endIndex(exclude).

- **StringBuffer delete() method**

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

- **StringBuffer reverse() method**

  The reverse() method of StringBuilder class reverses the current string.

String Manipulation

StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is the same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important Constructors of StringBuilder class

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

1. **public StringBuilder append(String s)** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public StringBuilder insert(int offset, String s)** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public StringBuilder replace(int startIndex, int endIndex, String str)** is used to replace the string from specified startIndex and endIndex.
4. **public StringBuilder delete(int startIndex, int endIndex)** is used to delete the string from specified startIndex and endIndex.
5. **public StringBuilder reverse()** is used to reverse the string.
6. **public int capacity()** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity)** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index)** is used to return the character at the specified position.

String Manipulation

9. **public int length()** is used to return the length of the string i.e. total number of characters.
10. **public String substring(int beginIndex)** is used to return the substring from the specified beginIndex.
11. **public String substring(int beginIndex, int endIndex)** is used to return the substring from the specified beginIndex and endIndex.

StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is a simple way to break the string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class.

Constructors of StringTokenizer class

- **StringTokenizer(String str)** creates StringTokenizer with specified string.
- **StringTokenizer(String str, String delim)** creates StringTokenizer with specified string and delimiter.
- **StringTokenizer(String str, String delim, boolean returnValue)** creates StringTokenizer with specified string, delimiter and returnValue. If the return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of StringTokenizer class

- **boolean hasMoreTokens()** checks if there are more tokens available.
- **String nextToken()** returns the next token from the StringTokenizer object.
- **String nextToken(String delim)** returns the next token based on the delimiter.
- **boolean hasMoreElements()** same as hasMoreTokens() method.
- **Object nextElement()** same as nextToken() but its return type is Object.
- **int countTokens()** returns the total number of tokens.

String Manipulation

StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).

Regular Expressions in Java

Regular Expressions or Regex (in short) is an API for defining String patterns that can be used for searching, manipulating and editing a string in Java. Email validation and passwords are few areas of strings where Regex are widely used to define the constraints. Regular Expressions are provided under java.util.regex package. This consists of 3 classes and 1 interface.

The java.util.regex package primarily consists of the following three classes:

util.regex.Pattern: Used for defining patterns

util.regex.Matcher : Used for performing match operations on text using patterns

PatternSyntaxException: Used for indicating syntax error in a regular expression pattern

MatchResult interface: This interface is used to determine the result of a match operation for a regular expression. It must be noted that although the match boundaries, groups and group boundaries can be seen, the modification is not allowed through a MatchResult

String Manipulation

import java.util.regex.Pattern;

```
class Demo{
   public static void main(String args[]){
         // Following line prints "true" because the whole
         // text "JavaDevelper" matches pattern "JavaD*veloper"
         System.out.println (Pattern.matches("JavaD*veloper", "JavaDevelper"));
         // Following line prints "false" because the whole
         //text "javafor" doesn't match pattern "f*java*or"
         System.out.println(Pattern.matches("f*java*or","javafor"));
   }
}
```

Java Program to demonstrate simple pattern searching

```
// A Simple Java program to demonstrate working of
// String matching in Java
import java.util.regex.Matcher;
import java.util.regex.Pattern;
class Demo {
   public static void main(String args[])  {
      // Create a pattern to be searched
      Pattern pattern = Pattern.compile("good");
      // Search above pattern in "goodforgood.org"
      Matcher m = pattern.matcher("goodforgood.org");
      // Print starting and ending indexes of the pattern  in text
      while (m.find())
         System.out.println("Pattern found from " + m.start() +  " to " + (m.end()-1));
   }
}
```

String Manipulation

Important Observations/Facts:

- We create a pattern object by calling Pattern.compile(), there is no constructor. compile() is a static method in Pattern class.
- Like above, we create a Matcher object using matcher() on objects of Pattern class.
- Pattern.matches() is also a static method that is used to check if given text as a whole matches pattern or not.
- find() is used to find multiple occurrences of patterns in text.

Reflection Array Class in Java

The Array class in java.lang.reflect package is a part of the Java Reflection. This class provides static methods to dynamically create and access Java arrays. It is a final class, which means it can't be instantiated, or changed. Only the methods of this class can be used by the class name itself.

The java.util.Arrays class contains various methods for manipulating arrays (such as sorting and searching) whereas this java.lang.reflect.Array class provides static methods to dynamically create and access Java arrays. This Array class keeps the array to be type-safe.

Array declaration in Array Class:

X[] arrayOfXType = (X[]) Array.newInstance(X.class, size);
where X is to be replaced by
the type of the array like int, double, etc.

Methods in Java Array:

- **static Object get(Object array, int index)**: This method returns the value of the indexed component in the specified array object.

String Manipulation

- **static boolean getBoolean(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a boolean.
- **static byte getByte(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a byte.
- **static char getChar(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a char.
- **static double getDouble(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a double.
- **static float getFloat(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a float.
- **static int getInt(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as an int.
- **static int getLength(Object array)**: This method returns the length of the specified array object, as an int.
- **static long getLong(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a long.
- **static short getShort(Object array, int index)**: This method returns the value of the indexed component in the specified array object, as a short.
- **static Object newInstance(Class<E> componentType, int length)**: This method creates a new array with the specified component type and length.
- **static Object newInstance(Class<E> componentType, int… dimensions)**: This method creates a new array with the specified component type and dimensions.
- **static void set(Object array, int index, Object value)**: This method sets the value of the indexed component of the specified array object to the specified new value.
- **static void setBoolean(Object array, int index, boolean z)**: This method sets the value of the indexed component of the specified array object to the specified boolean value.

String Manipulation

- **static void setByte(Object array, int index, byte b)**: This method sets the value of the indexed component of the specified array object to the specified byte value.
- **static void setChar(Object array, int index, char c)**: This method sets the value of the indexed component of the specified array object to the specified char value.
- **static void setDouble(Object array, int index, double d)**: This method sets the value of the indexed component of the specified array object to the specified double value.
- **static void setFloat(Object array, int index, float f)**: This method sets the value of the indexed component of the specified array object to the specified float value.
- **static void setInt(Object array, int index, int i)**: This method sets the value of the indexed component of the specified array object to the specified int value.
- **static void setLong(Object array, int index, long l)**: This method sets the value of the indexed component of the specified array object to the specified long value.
- **static void setShort(Object array, int index, short s)**: This method sets the value of the indexed component of the specified array object to the specified short value.

Arrays class in Java

The Arrays class in java.util package is a part of the Java Collection Framework. This class provides static methods to dynamically create and access Java arrays. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.

Need for the Java-Arrays Class:

There are often times when loops are used to do some tasks on an array like:

Fill an array with a particular value.

- Sort an Arrays.
- Search in an Arrays.
- And many more.

String Manipulation

Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops.

Some Methods in Java Array:

The Arrays class of the java.util package contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

- **static <T> List<T> asList(T… a)**: This method returns a fixed-size list backed by the specified Arrays.
- **static int binarySearch(elementToBeSearched):** These methods search for the specified element in the array with the help of Binary Search algorithm.
- **static <T> int binarySearch(T[] a, int fromIndex, int toIndex , T key, Comparator<T> c)**: This method searches a range of the specified array for the specified object using the binary search algorithm.
- **compare(array 1, array 2)**: This method compares two arrays passed as parameters lexicographically.
- **compareUnsigned(array 1, array 2)**: This method compares two arrays lexicographically, numerically treating elements as unsigned.
- **copyOf(originalArray, newLength)**: This method copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.
- **copyOfRange(originalArray, fromIndex, endIndex)**: This method copies the specified range of the specified array into a new Arrays.
- **static boolean deepEquals(Object[] a1, Object[] a2)**: This method returns true if the two specified arrays are deeply equal to one another
- **static String deepToString(Object[] a)**: This method returns a string representation of the “deep contents” of the specified Arrays.
- **equals(array1, array2)**: This method checks if both the arrays are equal or not.

String Manipulation

- **fill(originalArray, fillValue)**: This method assigns this fillValue to each index of this Arrays.
- **mismatch(array1, array2)**: This method finds and returns the index of the first unmatched element between the two specified arrays.
- **setAll(originalArray, functionalGenerator)**: This method sets all the elements of the specified array using the generator function provided.
- **sort(originalArray)**: This method sorts the complete array in ascending order
- **sort(originalArray, fromIndex, endIndex)**: This method sorts the specified range of array in ascending order.
- **toString(originalArray)**: This method returns a String representation of the contents of this Arrays. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters a comma followed by a space. Elements are converted to strings as by String.valueOf() function