JAVA NUMBERS, CHARS AND STRINGS

It turned out that all Workstation in the classroom are NOT set equally. This is why I wil demonstrate all examples using an on-line web tool http://www.browxy.com/

Please consider using this tool as an alternative to your NetBeans.

NUMBERS AS OBJECTS

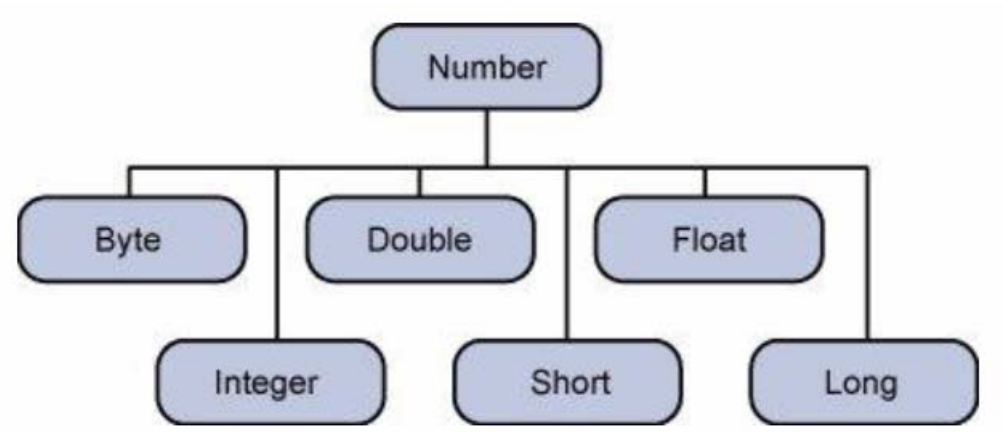
Normally, when we work with Numbers, we use **primitive data types such as byte, int, long, double**, etc.

Example:

```
int i = 5000;
float gpa = 13.65;
byte mask = 0xaf;
```

However, in development, we come across situations where we need to use **objects** instead of primitive data types. In order to achieve this, Java provides wrapper classes for each primitive data type.

All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number:



This wrapping is taken care of by the compiler, the process is called **boxing**. So when a primitive is used when an object is required, the compiler boxes the primitive type in its wrapper class. Similarly, the compiler unboxes the object to a primitive as well. The Number is part of the **java.lang** package.

Example: class Integer

For example, an object of type **Integer** contains a field whose type is int. Namely, it contains the following fields:

- static int MAX_VALUE -- This is a constant holding the maximum value an int can have, 231-1.

- static int MIN_VALUE -- This is a constant holding the minimum value an int can have, -231.
- static int SIZE -- This is the number of bits used to represent an int value in two's complement binary form.
- static Class<Integer> TYPE -- This is the class instance representing the primitive type int.

Also, it contains two constructors:

- Integer(int value) This constructs a newly allocated Integer object that represents the specified int value.
- Integer(String s) This constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

…and a number of class methods like

static int bitCount(int i)

This method returns the number of one-bits in the two's complement binary representation of the specified int value.

byte byteValue() This method returns the value of this Integer as a byte.

int compareTo(Integer anotherInteger) This method compares two Integer objects numerically.

static Integer decode(String nm) This method decodes a String into an Integer.

double doubleValue() This method returns the value of this Integer as a double.

boolean equals(Object obj) This method compares this object to the specified object.

static Integer getInteger(String nm) This method determines the integer value of the system property with the specified name.

int hashCode() This method returns a hash code for this Integer.

static int highestOneBit(int i) This method returns an int value with at most a single one-bit, in the position of the highest-order ("leftmost") one-bit in the specified int value.

static int lowestOneBit(int i) This method returns an int value with at most a single one-bit, in the position of the lowest-order ("rightmost") one-bit in the specified int value.

static int numberOfLeadingZeros(int i) This method returns the number of zero bits preceding the highest-order ("leftmost") one-bit in the two's complement binary representation of the specified int value.

static int parseInt(String s) This method parses the string argument as a signed decimal integer.

static int signum(int i) This method returns the signum function of the specified int value.

static String toBinaryString(int i) This method returns a string representation of the integer argument as an unsigned integer in base 2.

static String toHexString(int i) This method returns a string representation of the integer argument as an unsigned integer in base 16.

static String toOctalString(int i) This method returns a string representation of the integer argument as an unsigned integer in base 8.

static String toString(int i) This method returns a String object representing the specified integer.

static String toString(int i, int radix) This method returns a string representation of the first argument in the radix specified by the second argument.

static Integer valueOf(String s) This method returns an Integer object holding the value of the specified String.

(etc.)

Here is an example of boxing and unboxing:

```
public class Test{

   public static void main(String args[]){
      Integer x = 5; // boxes int to an Integer object
      x =  x + 10;   // unboxes the Integer to a int
      System.out.println(x);
   }
}
```

This would produce the following result:

```
15
```

When x is assigned integer values, the compiler boxes the integer because x is integer objects. Later, x is unboxed so that they can be added as integers.

NUMBER METHODS

Here is the list of the instance methods that all the subclasses of the Number class implement:

xxxValue()

Converts the value of this Number object to the xxx data type and returns it. E.g.:

byte	byteValue()
short	shortValue()
int	intValue()
long	longValue()
float	floatValue()
double	doubleValue()

EXAMPLE

```
public class Test{

   public static void main(String args[]){
      Integer x = 5;
      // Returns byte primitive data type
      System.out.println( x.byteValue() );
```

```
            // Returns double primitive data type
            System.out.println(x.doubleValue());

            // Returns long primitive data type
            System.out.println( x.longValue() );
         }
      }
```

Output is as follows:

```
5
5.0
5
```

compareTo()

Compares this Number object to the argument. Return value is:

- If the Integer is equal to the argument then 0 is returned.
- If the Integer is less than the argument then -1 is returned.
- If the Integer is greater than the argument then 1 is returned.

EXAMPLE:

```
public class Test{

   public static void main(String args[]){
      Integer x = 5;
      System.out.println(x.compareTo(3));
      System.out.println(x.compareTo(5));
      System.out.println(x.compareTo(8));
     }
}
```

This produces the following result:

```
1
0
-1
```

equals()

Determines whether this number object is equal to the argument. The methods returns True if the argument is not null and is an object of the same type and with the same numeric value.

(There are some extra requirements for Double and Float objects that are described in the Java API documentation.)

EXAMPLE:

```
public class Test{

   public static void main(String args[]){
```

```
        Integer x = 5;
        Integer y = 10;
        Integer z =5;
        Short a = 5;

        System.out.println(x.equals(y));
        System.out.println(x.equals(z));
        System.out.println(x.equals(a));
       }
}
```

This produces the following result:

```
false
true
false
```

valueOf()

All the variants of this method are given below:

```
static Integer valueOf(int i)
static Integer valueOf(String s)
static Integer valueOf(String s, int radix)
```

Where parameters are:

- i -- An int for which Integer representation would be returned.
- s -- A String for which Integer representation would be returned.
- radix -- This would be used to decide the value of returned Integer based on passed String.

For these three representations, return values are:

- valueOf(int i): This returns an Integer object holding the value of the specified primitive.
- valueOf(String s): This returns an Integer object holding the value of the specified string representation.
- valueOf(String s, int radix): This returns an Integer object holding the integer value of the specified string representation, parsed with the value of radix.

EXAMPLE

```
public class Test{

   public static void main(String args[]){

      Integer x =Integer.valueOf(9);
      Double c = Double.valueOf(5);
      Float a = Float.valueOf("80");

      Integer b = Integer.valueOf("444",16);
```

```
            System.out.println(x);
            System.out.println(c);
            System.out.println(a);
            System.out.println(b);
         }
      }
```

This produces the following result:

```
      9
      5.0
      80.0
      1092
```

toString()

The method is used to get a String object representing the value of the Number Object.

If the method takes two arguments, then a String representation of the first argument in the radix specified by the second argument will be returned.

Return values are:

```
      toString(): This returns a String object representing the value of
      this Integer.
      toString(int i): This returns a String object representing the
      specified integer.
```

EXAMPLE:

```
      public class Test{

         public static void main(String args[]){
            Integer x = 5;

            System.out.println(x.toString());
            System.out.println(Integer.toString(12));
         }
      }
```

This produces the following result:

```
      5
      12
```

parseInt()

This method is used to get the primitive data type of a certain String. Return values are:

- parseInt(String s): This returns an integer (decimal only).
- parseInt(int i): This returns an integer, given a string representation of decimal, binary, octal, or hexadecimal (radix equals 10, 2, 8, or 16 respectively) numbers as input.

EXAMPLE:

```
public class Test{

   public static void main(String args[]){
      int x =Integer.parseInt("9");
      double c = Double.parseDouble("5");
      int b = Integer.parseInt("444",16);

      System.out.println(x);
      System.out.println(c);
      System.out.println(b);
   }
}
```

This produces the following result:

```
9
5.0
1092
```

abs()

Returns the absolute value of the argument.

ceil()

Returns the smallest integer that is greater than or equal to the argument. Returned as a double.

floor()

Returns the largest integer that is less than or equal to the argument. Returned as a double.

rint()

Returns the integer that is closest in value to the argument. Returned as a double.

round()

Returns the closest long or int, as indicated by the method's return type, to the argument.

EXAMPLE:

```
public class Test{

   public static void main(String args[]){
      double d = 100.675;
      double e = 100.500;
      float f = 100;
      float g = 90f;

      System.out.println(Math.round(d));
      System.out.println(Math.round(e));
      System.out.println(Math.round(f));
```

```
            System.out.println(Math.round(g));
        }
    }
```

This produces the following result:

```
101
101
100
90
```

min()

Returns the smaller of the two arguments.

max()

Returns the larger of the two arguments.

exp()

Returns the base of the natural logarithms, e, to the power of the argument.

log()

Returns the natural logarithm of the argument.

pow()

Returns the value of the first argument raised to the power of the second argument.

sqrt()

Returns the square root of the argument.

sin()

Returns the sine of the specified double value.

cos()

Returns the cosine of the specified double value.

tan()

Returns the tangent of the specified double value.

asin()

Returns the arcsine of the specified double value.

acos()

Returns the arccosine of the specified double value.

atan()

Returns the arctangent of the specified double value.

atan2()

Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.

EXAMPLE:

```
public class Test{

   public static void main(String args[]){
     double x = 45.0;
     double y = 30.0;

     System.out.println( Math.atan2(x, y) );
   }
}
```

This produces the following result:

```
0.982793723247329
```

toDegrees()

Converts the argument to degrees

toRadians()

Converts the argument to radians.

random()

Returns a random number.

CHARS AS OBJECTS

Normally, when we work with characters, we use primitive data types char.

EXAMPLE:

```
char ch = 'a';

// Unicode for uppercase Greek omega character
char uniChar = '\u039A';
```

```
// an array of chars
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
```

However in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides wrapper class Character for primitive data type char.

The Character class offers a number of useful class (i.e., static) methods for manipulating characters. You can create a Character object with the Character constructor:

```
Character ch = new Character('a');
```

The Java compiler will also create a Character object for you under some circumstances. For example, if you pass a primitive char into a method that expects an object, the compiler automatically converts the char to a Character for you. This feature is called autoboxing or unboxing, if the conversion goes the other way.

EXAMPLE:

```
// Here following primitive char 'a'
// is boxed into the Character object ch
Character ch = 'a';

// Here primitive 'x' is boxed[1] for method test,
// return is unboxed to char 'c'
char c = test('x');
```

Escape Sequences:

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler.

The newline character (\n) is used frequently in System.out.println() statements to advance to the next line after the string is printed.

Following table shows the Java escape sequences:

\t Inserts a tab in the text at this point.

\b Inserts a backspace in the text at this point.

\n Inserts a newline in the text at this point.

\r Inserts a carriage return in the text at this point.

\f Inserts a form feed in the text at this point.

\' Inserts a single quote character in the text at this point.

\" Inserts a double quote character in the text at this point.

\\ Inserts a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

EXAMPLE:

If you want to put quotes within quotes, you must use the escape sequence, \", on the interior quotes:

[1]) i.e. converted into class

```
public class Test {

   public static void main(String args[]) {
      System.out.println("She said \"Hello!\" to me.");
   }
}
```

This would produce the following result:

```
She said "Hello!" to me.
```

CHARACTER METHODS

Here is the list of the important instance methods that all the subclasses of the Character class implement:

isLetter()

Determines whether the specified char value is a letter.

isDigit()

Determines whether the specified char value is a digit.

isWhitespace()

Determines whether the specified char value is white space.

isUpperCase()

Determines whether the specified char value is uppercase.

isLowerCase()

Determines whether the specified char is lowercase.

toUpperCase()

Returns the uppercase form of the specified char value.

toLowerCase()

Returns the lowercase form of the specified char value.

toString()

Returns a String object representing the specified character valuethat is, a one-character string.

REGULAR EXPRESSIONS

In computing, a regular expression (abbreviated regex or regexp) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations.

A very simple use of a regular expression would be to locate the same word spelled two different ways in a text editor, for example seriali[sz]e. A wildcard match can also achieve this, but wildcard matches differ from regular expressions in that wildcards are limited to what they can pattern (having fewer metacharacters and a simple language-base), whereas regular expressions are not.

STRING CLASS

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, **strings are objects**.

The Java platform provides the String class to create and manipulate strings.

Creating Strings:

The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has eleven constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

```
public class StringDemo{

   public static void main(String args[]){
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

This would produce the following result:

```
hello.
```

Important !

The String class is immutable, so that once it is created a String object **cannot be changed**. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

String Length

Methods used to obtain information about an object are known as **accessor methods**. One accessor method that you can use with strings is the `length()` method, which returns the number of characters contained in the string object.

After the following two lines of code have been executed, `len` equals 17:

```
public class StringDemo {

   public static void main(String args[]) {
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      System.out.println( "String Length is : " + len );
   }
}
```

This would produce the following result:

```
String Length is : 17
```

Concatenating Strings

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns **a new string** that is string1 with string2 added to it at the end. You can also use the `concat()` method with string literals, as in:

```
"My name is ".concat("Zara");
```

Strings are more commonly concatenated with the + operator, as in:

```
"Hello," + " world" + "!"
```

which results in:

```
"Hello, world!"
```

Let us look at the following example:

```
public class StringDemo {

   public static void main(String args[]) {
      String string1 = "saw I was ";
      System.out.println("Dot " + string1 + "Tod");
   }
}
```

This would produce the following result:

```
Dot saw I was Tod
```

Creating Format Strings

You have **`printf()`** and **`format()`** methods to print output with formatted numbers. The String class has an equivalent class method, **`format()`**, that returns a String object rather than a PrintStream object.

Using String's static **`format()`** method allows you to create a formatted string that you can reuse, as opposed to a one-time print statement. For example, instead of:

```
System.out.printf("The value of the float variable is " +
                  "%f, while the value of the integer " +
                  "variable is %d, and the string " +
                  "is %s", floatVar, intVar, stringVar);
```

you can write:

```
String fs;
fs = String.format("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

STRING METHODS

Here is the list of some (not all!) methods supported by String class:

char charAt(int index)

Returns the character at the specified index.

EXAMPLE:

```
public class Test {

   public static void main(String args[]) {
      String s = "Strings are immutable";
      char result = s.charAt(8);
      System.out.println(result);
   }
}
```

This produces the following result:

```
a
```

int compareTo(Object o)

Compares this String to another Object. The return value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

EXAMPLE:

```
public class Test {

   public static void main(String args[]) {
      String str1 = "Strings are immutable";
      String str2 = "Strings are immutable";
      String str3 = "Integers are not immutable";

      int result = str1.compareTo( str2 );
      System.out.println(result);

      result = str2.compareTo( str3 );
      System.out.println(result);

      result = str3.compareTo( str1 );
      System.out.println(result);
   }
}
```

This produces the following result:

```
0
10
-10
```

int compareTo(String anotherString)

Compares two strings lexicographically.

int compareToIgnoreCase(String str)

Compares two strings lexicographically, ignoring case differences.

String concat(String str)

Concatenates the specified string to the end of this string.

boolean contentEquals(StringBuffer sb)

Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.

static String copyValueOf(char[] data)

Returns a String that represents the character sequence in the array specified.

static String copyValueOf(char[] data, int offset, int count)

Returns a String that represents the character sequence in the array specified.

boolean endsWith(String suffix)

Tests if this string ends with the specified suffix.

boolean equals(Object anObject)

Compares this string to the specified object.

boolean equalsIgnoreCase(String anotherString)

Compares this String to another String, ignoring case considerations.

byte getBytes()

byte[] getBytes(String charsetName)

Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

EXAMPLE:

```
import java.io.*;

public class Test{

   public static void main(String args[]){
      String Str1 = new String("Welcome to Tutorialspoint.com");

      try{
         byte[] Str2 = Str1.getBytes();
         System.out.println("Returned  Value " + Str2 );

         Str2 = Str1.getBytes( "UTF-8" );
         System.out.println("Returned  Value " + Str2 );

         Str2 = Str1.getBytes( "ISO-8859-1" );
         System.out.println("Returned  Value " + Str2 );
      }catch( UnsupportedEncodingException e){
         System.out.println("Unsupported character set");
      }
   }
}
```

This produces the following result:

```
Returned  Value [B@192d342
Returned  Value [B@15ff48b
Returned  Value [B@1b90b39
```

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

Copies characters from this string into the destination character array. Here is the detail of parameters:

- srcBegin -- index of the first character in the string to copy.
- srcEnd -- index after the last character in the string to copy.
- dst -- the destination array.
- dstBegin -- the start offset in the destination array.

It does not return any value but on error it throws IndexOutOfBoundsException.

EXAMPLE 1

```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str1 = new String("Welcome to Tutorialspoint.com");
      char[] Str2 = new char[7];

      try{   // this catches possible errors
         Str1.getChars(2, 9, Str2, 0);
         System.out.print("Copied Value = " );
         System.out.println(Str2 );

      }catch( Exception ex){  //on error continues here
         System.out.println("Raised exception...");
      }
   }
}
```

This produces the following result:

```
Copied Value = lcome t
```

EXAMPLE 2

Now, consider very similar case, but with errorneous parameter dstBegin:

```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str1 = new String("Welcome to Tutorialspoint.com");
      char[] Str2 = new char[7];

      try{   // this catches possible errors
         Str1.getChars(2, 9, Str2, 100);
         System.out.print("Copied Value = " );
         System.out.println(Str2 );
```

```
            }catch( Exception ex){  //on error continues here
               System.out.println("Raised exception...");
            }
         }
      }
```

This produces the following result:

```
Raised exception...
```

int hashCode()

Returns a hash code for this string. The hash code for a String object is computed as:

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

Using int arithmetic, where s[i] is the i-th character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

int indexOf(int ch)

int indexOf(int ch, int fromIndex)

int indexOf(String str)

int indexOf(String str, int fromIndex)

Returns the index within this string of the first occurrence of the specified substring. This method has following different variants:

- public int indexOf(int ch): Returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.
- public int indexOf(int ch, int fromIndex): Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index or -1 if the character does not occur.
- int indexOf(String str): Returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.
- int indexOf(String str, int fromIndex): Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

EXAMPLE:

```
import java.io.*;

public class Test {

   public static void main(String args[]) {
      String Str = new String("Welcome to Tutorialspoint.com");
      String SubStr1 = new String("Tutorials");
      String SubStr2 = new String("Sutorials");

      System.out.print("Found Index :" );
```

```
            System.out.println(Str.indexOf( 'o' ));
            System.out.print("Found Index :" );
            System.out.println(Str.indexOf( 'o', 5 ));
            System.out.print("Found Index :" );
            System.out.println( Str.indexOf( SubStr1 ));
            System.out.print("Found Index :" );
            System.out.println( Str.indexOf( SubStr1, 15 ));
            System.out.print("Found Index :" );
            System.out.println(Str.indexOf( SubStr2 ));
         }
      }
```

This produces the following result:

```
Found Index :4
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

int lastIndexOf(int ch)

int lastIndexOf(int ch, int fromIndex)

int lastIndexOf(String str)

int lastIndexOf(String str, int fromIndex)

Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

int length()

Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

boolean matches(String regex)

Tells whether or not this string matches the given regular expression.

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)

Tests if two string regions are equal. Here is the detail of parameters:

- toffset -- the starting offset of the subregion in this string.
- other -- the string argument.
- ooffset -- the starting offset of the subregion in the string argument.
- len -- the number of characters to compare.
- ignoreCase -- if true, ignore case when comparing characters.

Return value is true if the specified subregion of this string matches the specified subregion of the string argument; false otherwise. Whether the matching is exact or case insensitive depends on the ignoreCase argument.

EXAMPLE

```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str1 = new String("Welcome to Tutorialspoint.com");
      String Str2 = new String("Tutorials");
      String Str3 = new String("TUTORIALS");

      System.out.print("Return Value :" );
      System.out.println(Str1.regionMatches(11, Str2, 0, 9));

      System.out.print("Return Value :" );
      System.out.println(Str1.regionMatches(11, Str3, 0, 9));

      System.out.print("Return Value :" );
      System.out.println(Str1.regionMatches(true, 11, Str3, 0, 9));
   }
}
```

This produces the following result:

```
Return Value :true
Return Value :false
Return Value :true
```

String replace(char oldChar, char newChar)

Returns a new string resulting from replacing **all occurrences** of oldChar in this string with newChar.

String replaceAll(String regex, String replacement)

Replaces each substring of this string that matches the given regular expression with the given replacement.

String replaceFirst(String regex, String replacement)

Replaces the first substring of this string that matches the given regular expression with the given replacement.

String[] split(String regex)

Splits this string around matches of the given regular expression.

String[] split(String regex, int limit)

Splits this string around matches of the given regular expression.

boolean startsWith(String prefix)

Tests if this string starts with the specified prefix.

boolean startsWith(String prefix, int toffset)

Tests if this string starts with the specified prefix beginning a specified index.

CharSequence subSequence(int beginIndex, int endIndex)

Returns a new character sequence that is a subsequence of this sequence.

String substring(int beginIndex)

Returns a new string that is a substring of this string.

String substring(int beginIndex, int endIndex)

Returns a new string that is a substring of this string.

char[] toCharArray()

Converts this string to a new character array.

String toLowerCase()

Converts all of the characters in this String to lower case using the rules of the default locale.

String toLowerCase(Locale locale)

Converts all of the characters in this String to lower case using the rules of the given Locale.

String toString()

This object (which is already a string!) is itself returned.

String toUpperCase()

Converts all of the characters in this String to upper case using the rules of the default locale.

String toUpperCase(Locale locale)

Converts all of the characters in this String to upper case using the rules of the given Locale.

String trim()

Returns a copy of the string, with leading and trailing whitespace omitted.

static String valueOf(primitive data type x)

This method has followings variants, which depend on the passed parameters. This method returns the string representation of the passed argument.

- valueOf(boolean b): Returns the string representation of the boolean argument.
- valueOf(char c): Returns the string representation of the char argument.
- valueOf(char[] data): Returns the string representation of the char array argument.
- valueOf(char[] data, int offset, int count): Returns the string representation of a specific subarray of the char array argument.
- valueOf(double d): Returns the string representation of the double argument.
- valueOf(float f): Returns the string representation of the float argument.
- valueOf(int i): Returns the string representation of the int argument.
- valueOf(long l): Returns the string representation of the long argument.
- valueOf(Object obj): Returns the string representation of the Object argument.

EXAMPLE

```
import java.io.*;

public class Test{
   public static void main(String args[]){
      double d = 102939939.939;
      boolean b = true;
      long l = 1232874;
      char[] arr = {'a', 'b', 'c', 'd', 'e', 'f','g' };

      System.out.println("Return Value : " + String.valueOf(d) );
      System.out.println("Return Value : " + String.valueOf(b) );
      System.out.println("Return Value : " + String.valueOf(l) );
      System.out.println("Return Value : " + String.valueOf(arr) );
   }
}
```

This produces the following result:

```
Return Value : 1.02939939939E8
Return Value : true
Return Value : 1232874
Return Value : abcdefg
```