تو اپنے پروردگار کے لیے نماز پڑھا کرو اور قربانی دیا کرو۔ الکوثر آیت ۲

Lecture 05

Library Functions and More Variable Types

In this lesson we will learn use of some library functions and some more variable types namely **char** and **String**. Talking specifically with respect to mathematical calculations if we have operators to perform some operation we use them for calculations but to perform many calculations we don't have operators but library functions available for them. Five things are important to use library functions:

1. **Location:** It is important to know class and package of library function. Sometimes we have to use import statement to use function like we have written import java.util.* to use input functions that is while using Scanner class
2. **Name:** name is very important as Java is case sensitive therefore we must know exact spellings with case of each alphabet to use them, though it will not be too difficult as mostly names are according to functionality like length, **set**, **size**, **sort** but many a times we have short names like **pow** for power, **sqrt** for square root. Secondly many a times function name is not a single word rather multiple words like **getExponenet** or **toDegrees**. Students must remember ***camel*** case convention used for variable and function names that is they start with small letter but every next word starts with capital letter like humps of camel. Therefore **count** is fine if a single word variable but **countPositive** for two words function's name.
3. **Arguments:** Many functions required arguments to perform functionality. Like **abs** function requires one argument that is number and return same or positive number if argument is a negative number. Similarly **pow** function requires two arguments. To use a function one has to give same number of arguments and compatible type of arguments as required by the function.
4. **Return type:** If we have to save value returned by function in some variable, we must know the return type of function. As Java is strict type therefore we can assign data to compatible type of variable only. Therefore if a function is returning double type like **sqrt** we have to store it in double type of variable only.
5. **Static or Non-Static:** Functions are of two types static (if mentioned) otherwise non-static. For the time being without going into details of static and non-static, we will here learn how to call both of them. In case of **static** functions we have to call functions by writing their class name followed by . (period) followed by function name like **Math.random or Math.abs**. In case of non-static functions we have first to create variable/ object than to call function we will write variable followed by . (period) followed by function name. See example for further explanation:

```
Scanner scan = new Scanner (System.in);
int n = scan.nextInt(); //or
double d = scan.nextDouble();
```

Let's now discuss some functions from Math library/ class.

`static double`	**abs**`(double a)`	Returns the absolute value of a `double` value.

abs is static function as written on left hand side, return type is double. This function requires single argument of type double.

`static double`	**pow**`(double a, double b)` Returns the value of the first argument raised to the power of the second argument.

Again **pow** is static function has double return type and two arguments of type double. Ofcourse this function is used to take power for example to calculate square of 7, use Math.pow(7, 2)

`static double`	**sqrt**`(double a)` Returns the correctly rounded positive square root of a `double` value.

Again **sqrt** is static function has double return type and one arguments of type double. Ofcourse this function is used to take under hood.

I am giving complete list of functions of Math and String library at the end of this lecture. Some of them you may need for Homework 2, others may be useful for future. Now let's consider some programs.

```
class TestMathFunctions{
   public static void main(String []args){
      //replace S.o.p with System.out.println
      S.o.p ("2 power 5 is "+Math.pow(2,5));
      S.o.p ("3 power 4 is "+Math.pow(3,4));
      S.o.p ("Under hood of 64 is "+Math.sqrt(64));
      S.o.p ("Absolute value of -7 is:"+Math.abs(-7));
   }
}
```

Output:

```
2 power 5 is 32.0
3 power 4 is 81.0
Under hood of 64 is 8.0
Absolute of -7 is:7
```

Formula to calculate area of circle is $A = \pi r^2$. To compute this input is r; whereas; π is constant and we can write Math.PI in Java programs. Here is a program to calculate area of circle. Radius is input of this program. Therefore, you can see different runs of same programs on right hand side. For example a different output for different values of radius. Student can try more values.

```
import java.util.*;

class CircleArea{
   public static void main(String []args){
      Scanner in=new Scanner(System.in);
      System.out.print("Enter Radius:");
      int r=in.nextInt();
      double area=Math.PI*Math.pow(r,2);
      System.out.println ("Area:"+area);
   }
}
```

Output:

```
Enter Radius: 5
Area: 78.5398
----------
Enter Radius: 10
Area: 314.1593
----------
Enter Radius: 4
Area: 50.2655
```

Another program to calculate area of sphere using formula $A = \frac{4}{3}\pi r^3$. Again r is input of program.

```
import java.util.*;

class Sphere{
   public static void main(String []args){
      Scanner in=new Scanner(System.in);
      System.out.print("Enter Radius:");
      int r=in.nextInt();
      double area=4/3.0*Math.PI*Math.pow(r,3);
      System.out.println ("Area:"+area);
   }
```

Output:

```
Enter Radius: 2
Area: 33.5103
----------
Enter Radius: 3
Area: 113.0973
```

```
}
```

In above program student should note that 4/3.0 is written instead of 4/3, only to avoid integer division to get correct results.

String and char Data Types

In this part first we will discuss two more data types than we will learn some functions related to String. **char** data type is used to store a single character and takes 2 bytes. Character should be given in single quotes otherwise they will be considered as variables or String if given in double quotes. For example char ch='A' or ch='0'. We will shortly see the use of char data type.

String is actually not a variable type rather a class in Java, students will slowly learn the difference between variable type and a class. Anyhow String is used to store set of characters. Their count can be 0 to onwards. Doube quotes must be used to start and close the set like "MIT" or "December". Just consider program String_Char:

```
class String_Char{
   public static void main(String []args){
      char f1='-';
      char f2=':';
      System.out.println("12"+f1+"07"+f1+"2012");
      System.out.println("12"+f2+"07"+f2+"2012");
      String degreeName="MIT";
      System.out.println("My degree name is:"+
                                          degreeName);
   }
}
```

Output:

```
12-07-2012
12:07:2012

My degree name is:MIT
```

Here f1 & f2 are two char type variables used, say format 1 and format 2. In first print statement f1 is used; whereas; in second print statement f2 is used and difference can be seen in output. Lastly degreeName is String type initialized by "MIT" and later used in print statement. Now consider some functions of String class.

`char`	**charAt**`(int index))`	Returns the `char` value at the specified index.
`String`	**substring**`(int beginIndex, int endIndex)`	Returns a new string that is a substring of this string.
`int`	**length**`()`	Returns the length of this string.

Functions are self-explanatory and ambiguities can be removed by observing code and its output. One thing is index starts from 0 that is first character is at 0 index, second is substring function returns characters that is difference of end and begin.

```
class StringFunctions{
   public static void main(String []args){
      String batch="MIT Fall 2012";
      //replace S.o.p with System.out.println
      S.o.p ("Length:"+batch.length());
      S.o.p ("First character is:"+batch.charAt(0));
      S.o.p ("Fifth character is:"+batch.charAt(4));
      S.o.p ("Year is:"+batch.substring(9,13));
   }
}
```

Output:

```
Length:13
First character is:M
Fifth character is:F
Year is:2012
```

Here is the last program showing you just a trick to animate a String:

```
import java.util.*;

class TestSubstring{
   public static void main(String []args){
      String name="Welcome";
      System.out.println(name.substring(0,1));
      System.out.println(name.substring(0,2));
      System.out.println(name.substring(0,3));
      System.out.println(name.substring(0,4));
      System.out.println(name.substring(0,5));
      System.out.println(name.substring(0,6));
      System.out.println(name.substring(0,7));

   }
}
```

Output:

```
W
We
Wel
Welc
Welco
Welcom
Welcome
```

Documentation of Math & String class

Math class documentation

You can notice that very first line is java.lang means Math library/ class is stored in ***lang*** package, which is by default included in every Java program and reason why we don't import it in our program. **Field Summary** shows constants defined in the class. **Method Summary** shows list of functions/ methods of Math class. On left hand side column two things are given. First whether function is static or non-static (if not mentioned), second return type of function. You can see double, int, long etc. in return type. On right hand side three things are given. First is name of function. Second is list of arguments used by function. Third is a very short description of function. If you are using actual documentation you may see a description about class on top, which I have omitted here. Secondly all the function names are hyperlinks so if you click them it will show you more detail about same function. Student must not feel fear to see very long list of functions. I have used hardly 25% of them and there are more than 60% I don't know names even. Finally from next line documentation of Math class is given:

java.lang

Class Math

java.lang.Object
  └ **java.lang.Math**

```
public final class Math
extends Object
```

Field Summary

static double	**E** The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Method Summary

static double	**abs**(double a) Returns the absolute value of a `double` value.
static float	**abs**(float a) Returns the absolute value of a `float` value.
static int	**abs**(int a) Returns the absolute value of an `int` value.
static long	**abs**(long a) Returns the absolute value of a `long` value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (`x`, `y`) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a `double` value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**cosh**(double x) Returns the hyperbolic cosine of a `double` value.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a `double` value.
static double	**expm1**(double x) Returns e^x -1.
static double	**floor**(double a) Returns the largest (closest to positive infinity) `double` value that is less than or equal to the argument and is equal to a mathematical integer.
static int	**getExponent**(double d) Returns the unbiased exponent used in the representation of a `double`.
static int	**getExponent**(float f) Returns the unbiased exponent used in the representation of a `float`.
static double	**hypot**(double x, double y) Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.

static double	**IEEEremainder**(double f1, double f2) Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
static double	**log**(double a) Returns the natural logarithm (base *e*) of a double value.
static double	**log10**(double a) Returns the base 10 logarithm of a double value.
static double	**log1p**(double x) Returns the natural logarithm of the sum of the argument and 1.
static double	**max**(double a, double b) Returns the greater of two double values.
static float	**max**(float a, float b) Returns the greater of two float values.
static int	**max**(int a, int b) Returns the greater of two int values.
static long	**max**(long a, long b) Returns the greater of two long values.
static double	**min**(double a, double b) Returns the smaller of two double values.
static float	**min**(float a, float b) Returns the smaller of two float values.
static int	**min**(int a, int b) Returns the smaller of two int values.
static long	**min**(long a, long b) Returns the smaller of two long values.
static double	**nextAfter**(double start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static float	**nextAfter**(float start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static double	**nextUp**(double d) Returns the floating-point value adjacent to d in the direction of positive infinity.
static float	**nextUp**(float f) Returns the floating-point value adjacent to f in the direction of positive infinity.
static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	**rint**(double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	**round**(double a) Returns the closest long to the argument.
static int	**round**(float a) Returns the closest int to the argument.

static double	**scalb**(double d, int scaleFactor) Return d × $2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
static float	**scalb**(float f, int scaleFactor) Return f × $2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
static double	**signum**(double d) Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
static float	**signum**(float f) Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
static double	**sin**(double a) Returns the trigonometric sine of an angle.
static double	**sinh**(double x) Returns the hyperbolic sine of a double value.
static double	**sqrt**(double a) Returns the correctly rounded positive square root of a double value.
static double	**tan**(double a) Returns the trigonometric tangent of an angle.
static double	**tanh**(double x) Returns the hyperbolic tangent of a double value.
static double	**toDegrees**(double angrad) Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
static double	**toRadians**(double angdeg) Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
static double	**ulp**(double d) Returns the size of an ulp of the argument.
static float	**ulp**(float f) Returns the size of an ulp of the argument.

String class documentation

Again very first line is java.lang means String library/ class is also stored in ***lang*** package. In String class most of the functions are non-static as static is not mentioned with them. Still there are some static functions as well like copyValueOf or format or valueOf. Therefore, students must not think that a class can either static functions or non-static functions rather class may have both. Actually it depends on the nature of function that whether it is to be static or non-static. Similarly you can see a variety of return type in functions of this class namely char, String, boolean, int, void etc. Finally from next line documentation of String class is given:

java.lang

Class String

java.lang.Object
  └ **java.lang.String**

Field Summary

static Comparator<String>	**CASE_INSENSITIVE_ORDER** A Comparator that orders String objects as by compareToIgnoreCase.

Constructor Summary

String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] bytes, Charset charset) Constructs a new String by decoding the specified array of bytes using the specified charset.
String(byte[] ascii, int hibyte) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a Charset, charset name, or that use the platform's default charset.*
String(byte[] bytes, int offset, int length) Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.
String(byte[] bytes, int offset, int length, Charset charset) Constructs a new String by decoding the specified subarray of bytes using the specified charset.
String(byte[] ascii, int hibyte, int offset, int count) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a Charset, charset name, or that use the platform's default charset.*
String(byte[] bytes, int offset, int length, String charsetName) Constructs a new String by decoding the specified subarray of bytes using the specified charset.
String(byte[] bytes, String charsetName) Constructs a new String by decoding the specified array of bytes using the specified charset.
String(char[] value) Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count) Allocates a new String that contains characters from a subarray of the character array argument.
String(int[] codePoints, int offset, int count) Allocates a new String that contains characters from a subarray of the Unicode code point array argument.
String(String original) Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
String(StringBuilder builder) Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

Method Summary

char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
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	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(StringBuffer sb) Compares this string to 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.
static String	**format**(Locale l, String format, Object... args) Returns a formatted string using the specified locale, format string, and arguments.
static String	**format**(String format, Object... args) Returns a formatted string using the specified format string and arguments.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
byte[]	**getBytes**(Charset charset) Encodes this String into a sequence of bytes using the given charset, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the getBytes() method, which uses the platform's default charset.*

byte[]	**getBytes**(String charsetName) Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hash code for this string.
int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
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.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
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.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, length() is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the rightmost occurrence of the specified substring.
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.
boolean	**matches**(String regex) Tells whether or not this string matches the given regular expression.
int	**offsetByCodePoints**(int index, int codePointOffset) Returns the index within this String that is offset from the given index by codePointOffset code points.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

String	**replace**(CharSequence target, CharSequence replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
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 the substring of this string beginning at the specified index starts with the specified prefix.
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**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static	**valueOf**(char[] data)

String	Returns the string representation of the char array argument.
static String	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	**valueOf**(double d) Returns the string representation of the double argument.
static String	**valueOf**(float f) Returns the string representation of the float argument.
static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.
static String	**valueOf**(Object obj) Returns the string representation of the Object argument.

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

The methods given at the end are not in String class but its parent class therefore they are also used like functions of String class