Java Lang Package

Fresher Learning Program
January, 2013

People matter, results count.

Objectives of Java Lang Package

- Purpose:
  - To understand important classes under Java Lang package like String, Math, Wrapper classes, StringBuffer / StringBuilder, and System Classes.

- Product:
  - Understanding of String class and its immutableness
  - Understanding of all Wrapper classes, their usage and how to use
  - Understanding of System class, its use and how to use
  - Understanding of Math class, its use and how to use

- Process:
  - Theory Sessions along with assignments
  - A recap at the end of the session in the form of Quiz.

Table of Contents

The *Math Class*

- Provides predefined constants and methods for performing different mathematical operations

- Methods:

Math Methods
`public static double abs(double a)`
Returns the positive value of the parameter. An overloaded method. Can also take in a float or an integer or a long integer as a parameter, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double random()`
Returns a random postive value greater than or equal to 0.0 but less than 1.0.
`public static double max(double a, double b)`
Returns the larger value between two *double* values, *a* and *b*. An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double min(double a, double b)`
Returns the smaller value between two *double* values, *a* and *b*. An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
`public static double ceil(double a)`
Returns the smallest integer that is greater than or equal to the specified parameter *a*.
`public static double floor(double a)`
Returns the largest integer that is lesser than or equal to the specified parameter *a*.

The *Math Class: Methods*

Math Methods
`public static double exp(double a)`
Returns Euler's number *e* raised to the power of the passed argument *a*.
`public static double log(double a)`
Returns the natural logarithm (base e) of *a*, the *double* value parameter.
`public static double pow(double a, double b)`
Returns the *double* value of *a* raised to the *double* value of *b*.
`public static long round(double a)`
Returns the nearest *long* to the given argument. An overloaded method. Can also take in a *float* as an argument and returns the nearest *int* in this case.
`public static double sqrt(double a)`
Returns the square root of the argument *a*.
`public static double sin(double a)`
Returns the trigonometric sine of the given angle *a*.
`public static double toDegrees(double angrad)`
Returns the degree value approximately equivalent to the given radian value.
`public static double toRadians(double angdeg)`
Returns the radian value approximately equivalent to the given degree value.

The *Math Class: Example*

```
class MathDemo {
        public static void main(St ring args[]) {
            System.out.println("absolute value of -5: " + Math.abs(-5));
            System.out.println("absolute value of 5: " + Math.abs(-5));
            System.out.println("random number(max is 10):" + Math.random()*10) ;
            System.out.println("max of 3.5 and 1.2: " + Math.max(3.5,1.2));
            System.out.println("min of 3.5 and 1.2: " + Math.min(3.5,1.2));
            System.out.println("ceiling of 3.5: " +  Math.ceil(3.5));
            System.out.println("floor of 3.5: " +  Math.floor(3.5));
            System.out.println("e raised to 1: " +  Math.exp(1));
            System.out.println("log 10: " + Math.log(10));
            System.out.println("10 raised to 3: " +  Math.pow(10,3));
            System.out.println("rounded off value of pi: " + Math.round(Math.PI));
            System.out.println("square root of 5 = " + Math.sqrt(5));
            System.out.println("10 radian = " + Math.toDegrees(10) + " degrees");
            System.out.println("sin(90): " +  Math.sin(Math.toRadians(90)));
        }
}
```

String Class

- Definition:
  - Represents combinations of character literals
  - Using Java, strings can be represented using:

    Array of characters

    The *String class*

– Note: A *String object is different from an array of* characters!

- String constructors
  - 13 constructors

The *String Class: Constructors*

```
class StringConstructorsDemo {
    public static void main(String args[]) {
        String s1 = new String(); //empty string
        char chars[] = { 'h', 'e', 'l', 'l', 'o'};
        String s2 = new String(chars); //s2="hello";
        byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
        String s3 = new String(bytes); //s3="world"
        String s4 = new String(chars, 1, 3);
        String s5 = new String(s2);
        String s6 = s2;
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
        System.out.println(s6);
    }
}
```

The *String Class: Methods*

String Methods
`public char charAt(int index)`
Returns the character located in the specified *index*.
`public int compareTo(String anotherString)`
Compares this string with the specified parameter. Returns a negative value if this string comes lexicographically before the other string, 0 if both of the strings have the same value and a postive value if this string comes after the other string lexicographically.
`public int compareToIgnoreCase(String str)`
Like compareTo but ignores the case used in this string and the specified string.
`public boolean equals(Object anObject)`
Returns true if this string has the same sequence of characters as that of the *Object* specified, which should be a *String* object. Otherwise, if the specified parameter is not a *String* object or if it doesn't match the sequence of symbols in this string, the method will return false.
`public boolean equalsIgnoreCase(String anotherString)`
Like equals but ignores the case used in this string and the specified string.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Gets the characters from this string starting at the *srcBegin* index up to the *srcEnd* index and copies these characters to the *dst* array starting at the *dstBegin* index.

The *String Class: Methods*

String Methods
`public int length()`
Returns the length of this string.
`public String replace(char oldChar, char newChar)`
Returns the string wherein all occurrences of the *oldChar* in this string is replaced with *newChar*.
`public String substring(int beginIndex, int endIndex)`
Returns the substring of this string starting at the specified *beginIndex* index up to the *endIndex* index.
`public char[] toCharArray()`
Returns the character array equivalent of this string.
`public String trim()`
Returns a modified copy of this string wherein the leading and trailing white space are removed.
`public static String valueOf(-)`
Takes in a simple data type such as boolean, integer or character, or it takes in an object as a parameter and returns the *String* equivalent of the specified parameter.

The *StringBuffer Class*

- Problem with *String objects:*
  – Once created, can no longer be modified (It is a final class)

- A StringBuffer object
  – Similar to a String object
  – But, mutable or can be modified
    - Unlike *String in this aspect*
    - Length and content may changed through some method calls

The *StringBuffer Class: Methods*

StringBuffer Methods
`public int capacity()`
Returns the current capacity of this *StringBuffer* object.
`public StringBuffer append(-)`
Appends the string representation of the argument to this *StringBuffer* object. Takes in a single parameter which may be of these data types: *boolean, char, char [], double, float, int, long, Object, String and StringBuffer*. Still has another overloaded version.
`public char charAt(int index)`
Returns the character located in the specified *index*.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Gets the characters from this object starting at the *srcBegin* index up to the *srcEnd* index and copies these characters to the *dst* array starting at the *dstBegin* index.
`public StringBuffer delete(int start, int end)`
Deletes the characters within the specified range.
`public StringBuffer insert(int offset, -)`
Inserts the string representation of the second argument at the specified offset. An overloaded method. Possible data types for the second argument: *boolean, char, char [], double, float, int, long, Object and String*. Still has another overloaded version.

The *StringBuilder Class*

- StringBuilder class is same as StringBuffer.
- Only difference is that StringBuffer is thread Safe.
- In general – it is better to use StringBuilder, until thread safety is concerned.

The Wrapper Classes

Some Facts:

- Primitive data types are not objects
- Cannot access methods of the *Object class*
- Only actual objects can access methods of the *Object* class

Why wrapper classes?

Need an object representation for the primitive type variables to use Java built-in methods

Definition: Object representations of simple non-object variables

The *Wrapper Class*

- Wrapper classes are very similar to their primitive equivalents.
  - Capitalized
  - Spelled out versions of the primitive data types

Wrapper Classes

- It converts primitive values to objects.
- Need of Wrapper class:
- Primitive values in java are not objects. So in order to include the primitives in the activities reserved for objects for ex-to be used as elements of Collections or to be returned as objects from a method we need Wrapper classes.
- There is a Wrapper class for every primitive in java.
- All Wrapper classes are **final**.
- The objects of all Wrapper classes are **immutable** i.e their state can not be changed

The *Wrapper Class: Example*

PrimitiveType	Corresponding Wrapper class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Numeric Wrapper Classes

- Each numeric Wrapper classdefines minimum and maximum value of the corresponding primitive data type.

  <Wrapperclassname>.MIN_VALUE

  <Wrapperclassname>.MAX_VALUE

  Example -

  int max = Integer.MAX_VALUE

  double max = Double.MIN_VALUE

Creating Wrapper Objects

By using Constructors : (converting primitive values to wrapper objects)

```
Character     c1 = new Character('c');

Boolean b1 = new Boolean(true);

Integer i1 = new Integer(8000);

Double d1 = new Double(8.34);
```

Creating Wrapper Objects

B(converting Strings to Wrapper objects)

Boolean b2 = new Boolean(“TrUe”); // true

Boolean b3 = new Boolean(“xx”); //false

Integer i2 = new Integer(“8000”);

Double d2 = new Double(“8.45”);

Long long1 = new Long(8);

Using Wrapper conversion utilities

1. Converting Wrapper objects to Strings

String toString()

Each Wrapper classoverrides the toString() method from the Object class.

```
Character c1 = new Character('g');
String s1 = c1.toString();
Boolean b1 = new Boolean(true);
String s2 = b1.toString();
```

Using Wrapper conversion…

2.Converting primitive values to Strings

Each wrapper class defines a static method toString(type v) that returns the String corresponding to the primitive value .

static String toString(type v)

```
String s1 = Character.toString('\n');
String s2 = Boolean.toString(false);
String s3 = Integer.toString(2000);// base 10
String s4 = Double.toString(1.55);
```

Using Wrapper conversion…

3. Converting integer values to Strings in different Notations

static String toBinaryString(int i) //base 2

static String toHexString(int i) //base 16

static String toOctalString(int i) //base 8

These three methods return an unsigned String value of base 2,16,8 with no leading zeros.

static String toString(int i, int base)

This method returns the minus sign('-') as the first character if i is negative.

static String toString(int i)

Using Wrapper conversion…

4. Converting Wrapper objects to primitive values

Each Wrapper classexcept Boolean and Character defines a typeValue() which returns the primitive value in the wrapper object.

type typeValue()

```
double d = d1.doubleValue();
int i = i1.intValue();
```

Using Wrapper conversion…

5. Converting any numeric Wrapper objects into any numeric primitive values

```
Byte b1 = new Byte((byte)17);
Integer i1 = new Integer(2005);
Double d1 = new Double(1.48);

short s1 = i1.shortValue();
long  l1 = b1.longValue();
int i2 = d1.intValue(); //truncation
double d2 = i1.doubleValue();
```

Using Wrapper conversion…

6. Converting Strings to Numeric values

Each numeric wrapper classdefines a static method parseType(String s) that returns the primitive numeric value of String it contains.

This method will throw a NumberFormatException if the String parameter is not a valid argument.

Using Wrapper conversion…

1. static type parseType(String s)

```
byte b1 = Byte.parseByte("18");
int i1 = Integer.parseInt("2007");
int i2 = Integer.parseInt("abc");//NFE
```

2. static typeparseType(String s, int base)

```
byte b1 = Byte.parseByte("1010",2);
short s1 = Short.parseShort("013",8);
long l1 = Long.parseLong("-a",16);
int i1 = Integer.parseInt("500",16);
```

The Wrapper Classes: Boolean Example

```
class BooleanWrapper {
    public static void main(String args[]) {
        boolean booleanVar = 1>2;
        Boolean booleanObj = new Boolean("True");
        /* primitive to object; can also use valueOf method */
        Boolean booleanObj2 = new Boolean(booleanVar);
        System.out.println("booleanVar = " + booleanVar);
        System.out.println("booleanObj = " + booleanObj);
        System.out.println("booleanObj2 = " +
        booleanObj2);
       System.out.println("compare 2 wrapper objects: "
        + booleanObj.equals(booleanObj2));
        /* object to primitive */
        booleanVar = booleanObj.booleanValue();
        System.out.println("booleanVar = " + booleanVar);
       }
 }
```

The *System Class*

- Provides many useful fields and methods
  - Standard input
  - Standard output
  - Utility method for fast copying of a part of an array

The *System Class: Methods*

System Methods
`public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
Copies *length* items from the source array *src* starting at *srcPos* to *dest* starting at index *destPos*. Faster than manually programming the code for this yourself.
`public static long currentTimeMillis()`
Returns the difference between the current time and January 1, 1970 UTC. Time returned is measured in milliseconds.
`public static void exit(int status)`
Kills the Java Virtual Machine (JVM) running currently. A non-zero value for status by convention indicates an abnormal exit.
`public static void gc()`
Runs the garbage collector, which reclaims unused memory space for recycling.
`public static void setIn(InputStream in)`
Changes the stream associated with *System.in*, which by default refers to the keyboard.
`public static void setOut(PrintStream out)`
Changes the stream associated with *System.out*, which by default refers to the console.

The *System Class: Example*

```
import java.io.IOException;
public class Elapsed {
        public static void main(String[] args) throws IOException {
                long lngStart, lngEnd = 0;
                System.out.println("Timing a for from 0 to 1,000,000");
                /time a for loop from 0 to 1,000,000
                lngStart=System.currentTimeMillis();
                for(int j=0;j<1000000;j++)
                        lngEnd = System.currentTimeMillis();
                System.out.println("Elapsed time : " + (lngEnd-lngStart));

        }
  }
```

Output:

Timing a for from 0 to 1,000,000 Elapsed time: 78

Recap (Hot Keywords)

Math.cos

Integer.parseInt

System.out.print

StringBuilder

Immutable

Append

String

Thank You For Your Time

People matter, results count.