CHP-11
Java Numbers

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.
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);
}
}
```

CHAPTER

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:

SN Methods with Description

1 xxxValue()
Converts the value of *this* Number object to the xxx data type and returned it.
2 compareTo()
Compares *this* Number object to the argument.
3 equals()
Determines whether *this* number object is equal to the argument.
4 valueOf()
Returns an Integer object holding the value of the specified primitive.
5 toString()

Returns a String object representing the value of specified int or Integer.
6 parseInt()
This method is used to get the primitive data type of a certain String.
7 abs()
Returns the absolute value of the argument.
8 ceil()
Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
9 floor()
Returns the largest integer that is less than or equal to the argument. Returned as a double.
10 rint()
Returns the integer that is closest in value to the argument. Returned as a double.
11 round()
Returns the closest long or int, as indicated by the method's return type, to the argument.
12 min()
Returns the smaller of the two arguments.
13 max()
Returns the larger of the two arguments.
14 exp()
Returns the base of the natural logarithms, e, to the power of the argument.
15 log()
Returns the natural logarithm of the argument.
16 pow()
Returns the value of the first argument raised to the power of the second argument.

17 sqrt()
Returns the square root of the argument.
18 sin()
Returns the sine of the specified double value.
19 cos()
Returns the cosine of the specified double value.
20 tan()
Returns the tangent of the specified double value.
21 asin()
Returns the arcsine of the specified double value.
22 acos()
Returns the arccosine of the specified double value.
23 atan()
Returns the arctangent of the specified double value.
24 atan2()
Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
25 toDegrees()
Converts the argument to degrees
26 toRadians()
Converts the argument to radians.
27 random()
Returns a random number.

xxxValue()

Description:

The method converts the value of the Number Object that invokes the method to the primitive data type that is returned from the method.

Syntax:

Here is a separate method for each primitive data type:

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

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- These method returns the primitive data type that is given in the signature.

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());
}
}
```

This produces the following result:

```
5
5.0
5
```

compareTo()

Description:

The method compares the Number object that invoked the method to the argument. It is possible to compare Byte, Long, Integer, etc.
However, two different types cannot be compared, both the argument and the Number object invoking the method should be of same type.

Syntax:

```
publicint compareTo(NumberSubClass referenceName )
```

Parameters:

Here is the detail of parameters:

- **referenceName** -- This could be a Byte, Double, Integer, Float, Long or Short.

Return Value:

- 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()

Description:

The method determines whether the Number Object that invokes the method is equal to the argument.

Syntax:

```
publicboolean equals(Object o)
```

Parameters:

Here is the detail of parameters:

- **o** -- Any object.

Return Value:

- 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()

Description:

The valueOf method returns the relevant Number Object holding the value of the argument passed. The argument can be a primitive data type, String, etc.

This method is a static method. The method can take two arguments, where one is a String and the other is a radix.

Syntax:

All the variants of this method are given below:

```
staticInteger valueOf(int i)
staticInteger valueOf(String s)
staticInteger valueOf(String s,int radix)
```

Parameters:

Here is the detail of parameters:

- **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.

Return Value:

 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.

```
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()

Description:

The method is used to get a String object representing the value of the Number Object.
If the method takes a primitive data type as an argument, then the String object representing the primitive data type value is return.
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.

Syntax:

All the variant of this method are given below:

```
String toString()
staticString toString(int i)
```

Parameters:

Here is the detail of parameters:
 i -- An int for which string representation would be returned.

Return Value:

 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()

Description:

This method is used to get the primitive data type of a certain String. parseXxx() is a static method and can have one argument or two.

Syntax:

All the variant of this method are given below:

```
staticint parseInt(String s)
s
taticint parseInt(String s,int radix)
```

Parameters:

Here is the detail of parameters:

- **s** -- This is a string representation of decimal.
- **radix** -- This would be used to convert String s into integer.

Return Value:

- **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()

Description:

The method gives the absolute value of the argument. The argument can be int, float, long, double, short, byte.

Syntax:

All the variant of this method are given below:

```
double abs(double d)
float abs(float f)
int abs(int i)
long abs(long lng)
```

Parameters:

Here is the detail of parameters:

- Any primitive data type

Return Value:

 This method Returns the absolute value of the argument.

Example:

```
public class Test{
public static void main(String args[]){
Integer a =-8;
double d =-100;
float f =-90;
System.out.println(Math.abs(a));
System.out.println(Math.abs(d));
System.out.println(Math.abs(f));
}
}
```


This produces the following result:

```
8
100.0
90.0
```

ceil()

Description:

The method ceil gives the smallest integer that is greater than or equal to the argument.

Syntax:

This method has following variants:

```
double ceil(double d)
double ceil(float f)
```

Parameters:

Here is the detail of parameters:

 A double or float primitive data type

Return Value:

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

Example:

```
public class Test{
public static void main(String args[]){
double d =-100.675;
float f =-90;
System.out.println(Math.ceil(d));
System.out.println(Math.ceil(f));
System.out.println(Math.floor(d));
System.out.println(Math.floor(f));
}
}
```

This produces the following result:

```
-100.0
-90.0
-101.0
```



```
-90.0
```

floor()

Description:

The method floor gives the largest integer that is less than or equal to the argument.

Syntax:

This method has following variants:

```
double floor(double d)
double floor(float f)
```

Parameters:

Here is the detail of parameters:

- A double or float primitive data type

Return Value:

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

Example:

```
public class Test{
public static void main(String args[]){
double d =-100.675;
float f =-90;
System.out.println(Math.floor(d));
System.out.println(Math.floor(f));
System.out.println(Math.ceil(d));
System.out.println(Math.ceil(f));
}
}
```

This produces the following result:

```
-101.0
-90.0
-100.0
-90.0
```


rint()

Description:

The method rint returns the integer that is closest in value to the argument.

Syntax:

```
double rint(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double primitive data type

Return Value:

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

Example:

```
public class Test{
public static void main(String args[]){
double d =100.675;
double e =100.500;
double f =100.200;
System.out.println(Math.rint(d));
```

```
System.out.println(Math.rint(e));
System.out.println(Math.rint(f));
}
}
```

This produces the following result:

```
101.0
100.0
100.0
```

round()

Description:

The method round returns the closest long or int, as given by the methods return type.

Syntax:

This method has following variants:

```
long round(double d)
int round(float f)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double or float primitive data type
- **f** -- A float primitive data type

Return Value:

- This method 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()

Description:

The method gives the smaller of the two arguments. The argument can be int, float, long, double.

Syntax:

This method has following variants:

```
double min(double arg1,double arg2)
float min(float arg1,float arg2)
int min(int arg1,int arg2)
```

```
long min(long arg1,long arg2)
```

Parameters:

Here is the detail of parameters:

- A primitive data types

Return Value:

- This method Returns the smaller of the two arguments.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Math.min(12.123,12.456));
System.out.println(Math.min(23.12,23.0));
}
}
```

This produces the following result:

```
12.123
23.0
```

max()

Description:

The method gives the maximum of the two arguments. The argument can be int, float, long, double.

Syntax:

This method has following variants:

```
double max(double arg1,double arg2)
float max(float arg1,float arg2)
int max(int arg1,int arg2)
long max(long arg1,long arg2)
```

Parameters:

Here is the detail of parameters:

- A primitive data types

Return Value:

- This method returns the maximum of the two arguments.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Math.max(12.123,12.456));
System.out.println(Math.max(23.12,23.0));
}
}
```

This produces the following result:

```
12.456
23.12
```

exp()

Description:

The method returns the base of the natural logarithms, e, to the power of the argument.

Syntax:

```
double exp(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A primitive data types

Return Value:

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

Example:

```
public class Test{
public static void main(String args[]){
double x =11.635;
double y =2.76;
System.out.printf("The value of e is %.4f%n",Math.E);
```



```
System.out.printf("exp(%.3f) is %.3f%n", x,Math.exp(x));
}
}
```

This produces the following result:

```
The value of e is 2.7183
exp(11.635) is 112983.831
```

log()

Description:

The method returns the natural logarithm of the argument.

Syntax:

```
double log(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A primitive data types

Return Value:

- This method Returns the natural logarithm of the argument.

Example:

```
public class Test{
public static void main(String args[]){
double x =11.635;
double y =2.76;
System.out.printf("The value of e is %.4f%n",Math.E);
System.out.printf("log(%.3f) is %.3f%n", x,Math.log(x));
}
}
```

This produces the following result:

```
The value of e is 2.7183
log(11.635) is 2.454
```

pow()

Description:

The method returns the value of the first argument raised to the power of the second argument.

Syntax:

```
double pow(doublebase,double exponent)
```

Parameters:

Here is the detail of parameters:

- **base** -- A primitive data type
- **exponenet** -- A primitive data type

Return Value:

- This method Returns the value of the first argument raised to the power of the second argument.

Example:

```
public class Test{
public static void main(String args[]){
double x =11.635;
double y =2.76;
System.out.printf("The value of e is %.4f%n",Math.E);
System.out.printf("pow(%.3f, %.3f) is %.3f%n",x, y,Math.pow(x, y));
}
}
```

This produces the following result:

```
The value of e is 2.7183
pow(11.635, 2.760) is 874.008
```

sqrt()

Description:

The method returns the square root of the argument.

Syntax:

```
double sqrt(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A primitive data type

Return Value:

- This method Returns the square root of the argument.

Example:

```
public class Test{
public static void main(String args[]){
double x =11.635;
double y =2.76;
System.out.printf("The value of e is %.4f%n",Math.E);
System.out.printf("sqrt(%.3f) is %.3f%n", x,Math.sqrt(x));
}
}
```

This produces the following result:

```
The value of e is 2.7183
sqrt(11.635) is 3.411
```

sin()

Description:

The method returns the sine of the specified double value.

Syntax:

```
double sin(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data types

Return Value:

- This method Returns the sine of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
```



```
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The sine of %.1f degrees is
%.4f%n",degrees,Math.sin(radians));
}
}
```

This produces the following result:

```
The value of pi is 3.1416
The sine of 45.0 degrees is 0.7071
```

cos()

Description:

The method returns the cosine of the specified double value.

Syntax:

```
double cos(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data types

Return Value:

- This method Returns the cosine of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The cosine of %.1f degrees is %.4f%n",
degrees,Math.cos(radians));
}
}
```

This produces the following result:


```
The value of pi is 3.1416
The cosine of 45.0 degrees is 0.7071
```

tan()

Description:

The method returns the tangent of the specified double value.

Syntax:

```
double tan(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data type

Return Value:

- This method returns the tangent of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The tangent of %.1f degrees is %.4f%n",
degrees,Math.tan(radians));
}
}
```

This produces the following result:

```
The value of pi is 3.1416
The tangent of 45.0 degrees is 1.0000
```

asin()

Description:

The method returns the arcsine of the specified double value.

Syntax:

```
double asin(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data types

Return Value:

- This method Returns the arcsine of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The arcsine of %.4f is %.4f degrees %n",
Math.sin(radians),
Math.toDegrees(Math.asin(Math.sin(radians))));
}
}
```

This produces the following result:

```
The value of pi is 3.1416
The arcsine of 0.7071 is 45.0000 degrees
```

acos()

Description:

The method returns the arccosine of the specified double value.

Syntax:

```
double acos(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data types

Return Value:

- This method Returns the arccosine of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The arccosine of %.4f is %.4f degrees %n",
Math.cos(radians),
Math.toDegrees(Math.acos(Math.sin(radians))));
}
}
```

This produces the following result:

```
The value of pi is 3.1416
The arccosine of 0.7071 is 45.0000 degrees
```

atan()

Description:

The method returns the arctangent of the specified double value.

Syntax:

```
double atan(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data types

Return Value :

- This method Returns the arctangent of the specified double value.

Example:

```
public class Test{
public static void main(String args[]){
double degrees =45.0;
double radians =Math.toRadians(degrees);
System.out.format("The value of pi is %.4f%n",Math.PI);
System.out.format("The arctangent of %.4f is %.4f degrees %n",
Math.cos(radians),
Math.toDegrees(Math.atan(Math.sin(radians))));
}
```

```
}
```
This produces the following result:
```
The value of pi is 3.1416
The arctangent of 1.0000 is 45.0000 degrees
```
atan2()
Description:
The method Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
Syntax:
```
double atan2(double y,double x)
```
Parameters:
Here is the detail of parameters:
- **X** -- X co-ordinate in double data type
- **Y** -- Y co-ordinate in double data type

Return Value:
- This method Returns theta from polar coordinate (r, 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()
Description:
The method converts the argument value to degrees.
Syntax:
```
double toDegrees(double d)
```
Parameters:
Here is the detail of parameters:
- **d** -- A double data type.

Return Value:
- This method returns a double value.

Example:
```
public class Test{
public static void main(String args[]){
double x =45.0;
double y =30.0;
System.out.println(Math.toDegrees(x));
System.out.println(Math.toDegrees(y));
}
}
```
This produces the following result:

```
2578.3100780887044
1718.8733853924698
```

toRadians()

Description:

The method converts the argument value to radians.

Syntax:

```
double toRadians(double d)
```

Parameters:

Here is the detail of parameters:

- **d** -- A double data type.

Return Value:

- This method returns a double value.

Example:

```
public class Test{
public static void main(String args[]){
double x =45.0;
double y =30.0;
System.out.println(Math.toRadians(x));
System.out.println(Math.toRadians(y));
}
}
```

This produces the following result:

```
0.7853981633974483
0.5235987755982988
```

random()

Description:

The method is used to generate a random number between 0.0 and 1.0. The range is: 0.0 =< Math.random < 1.0. Different ranges can be achieved by using arithmetic.

Syntax:

```
staticdouble random()
```

Parameters:

Here is the detail of parameters:

- NA

Return Value:

- This method returns a double

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Math.random());
System.out.println(Math.random());
}
}
```

This produces the following result:

```
0.16763945061451657
0.400551253762343
```

Note: Above result would vary every time you would call random() method.

What is Next?

In the next section, we will be going through the Character class in Java. You will be learning how to use object Characters and primitive data type char in Java.

CHP-12
Java Characters

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. Inorder 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 =newCharacter('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 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.

CHAPTER

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

Following table shows the Java escape sequences:

Escape Sequence Description

\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:

```
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:

SN Methods with Description

1 isLetter()
Determines whether the specified char value is a letter.
2 isDigit()
Determines whether the specified char value is a digit.
3 isWhitespace()
Determines whether the specified char value is white space.
4 isUpperCase()
Determines whether the specified char value is uppercase.

5 isLowerCase()
Determines whether the specified char value is lowercase.
6 toUpperCase()
Returns the uppercase form of the specified char value.
7 toLowerCase()
Returns the lowercase form of the specified char value.
8 toString()
Returns a String object representing the specified character valuethat is, a one-character string.
For a complete list of methods, please refer to the java.lang.Character API specification.

isLetter()

Description:

The method determines whether the specified char value is a letter.

Syntax:

```
boolean isLetter(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns true if passed character is really a character.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.isLetter('c'));
System.out.println(Character.isLetter('5'));
}
}
```

This produces the following result:

```
true
```

```
false
```


isDigit()

Description:

The method determines whether the specified char value is a digit.

Syntax:

```
boolean isDigit(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns true if passed character is really a digit.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.isDigit('c'));
System.out.println(Character.isDigit('5'));
}
}
```

This produces the following result:

```
false
true
```

isWhitespace()

Description:

The method determines whether the specified char value is a white space, which includes space, tab or new line.

Syntax:

```
boolean isWhitespace(char ch)
```


Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns true if passed character is really a white space.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.isWhitespace('c'));
System.out.println(Character.isWhitespace(' '));
System.out.println(Character.isWhitespace('\n'));
System.out.println(Character.isWhitespace('\t'));
}
}
```

This produces the following result:

```
false
true
true
true
```

isUpperCase()

Description:

The method determines whether the specified char value is uppercase.

Syntax:

```
boolean isUpperCase(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns true if passed character is really an uppercase.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.isUpperCase('c'));
System.out.println(Character.isUpperCase('C'));
System.out.println(Character.isUpperCase('\n'));
System.out.println(Character.isUpperCase('\t'));
}
}
```

This produces the following result:

```
false
true
false
false
```

isLowerCase()

Description:

The method determines whether the specified char value is lowercase.

Syntax:

```
boolean isLowerCase(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns true if passed character is really an lowercase.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.isLowerCase('c'));
System.out.println(Character.isLowerCase('C'));
System.out.println(Character.isLowerCase('\n'));
```

```
System.out.println(Character.isLowerCase('\t'));
}
}
```


This produces the following result:

```
true
false
false
false
```

toUpperCase()

Description:

The method returns the uppercase form of the specified char value.

Syntax:

```
char toUpperCase(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value :

- This method Returns the uppercase form of the specified char value.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.toUpperCase('c'));
System.out.println(Character.toUpperCase('C'));
}
}
```

This produces the following result:

```
C
C
```

toLowerCase()

Description:

The method returns the lowercase form of the specified char value.

Syntax:

```
char toLowerCase(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns the lowercase form of the specified char value.

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.toLowerCase('c'));
System.out.println(Character.toLowerCase('C'));
```

```
}
}
```
This produces the following result:
```
c
c
```
toString()

Description:

The method returns a String object representing the specified character value, that is, a one-character string.

Syntax:

```
String toString(char ch)
```

Parameters:

Here is the detail of parameters:

- **ch** -- Primitive character type

Return Value:

- This method Returns String object

Example:

```
public class Test{
public static void main(String args[]){
System.out.println(Character.toString('c'));
System.out.println(Character.toString('C'));
}
}
```
This produces the following result:
```
c
C
```
What is Next?

In the next section, we will be going through the String class in Java. You will be learning how to declare and use Strings efficiently as well as some of the important methods in the String class.