Numbers

By Dr. Subrat Kumar Nayak

Associate Professor

Department of CSE

ITER, SOADU

Some More Functions in Wrapper Classes…

equals()

- The method determines whether the Number object that invokes the method is equal to the object that is passed as an argument.

Syntax

```
public boolean equals(Object o)
```

Return Value:

- The method returns True if the argument is not null and is an object of the same type and with the same numeric value.

Example:

```
public class Test {
public static void main(String args[]) {
       Integer x = 5;
       Integer y = 10;
       Integer z =5;
       System.out.println(x.equals(y));
       System.out.println(x.equals(z));
} }
```

Some More Functions in Wrapper Classes…

toString()

- 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 returned.
- 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:

```
String toString()
static String toString(int i)
Static String toString(int i, int radix)
```

Example :

```
public class Test {
public static void main(String args[]) {
Integer x = 5;
System.out.println(x.toString());
System.out.println(Integer.toString(12));
System.out.println(Integer.toString(12,2));
} }
```

```
Output:
5
12
1100
```

Some More Functions in Wrapper Classes…

parseXxx()

- 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.
- Does not work for Character.

Note: same as valueOf(). However, it only works on String object.

Syntax:

```
static int parseInt(String s)
static int parseInt(String s, int radix)
```

Example:

```
public class Test { public static void main(String args[]) {
    int x =Integer.parseInt("9");
    double c = Double.parseDouble("5");
    int b = Integer.parseInt("1100",2);
    System.out.println(x);
    System.out.println(c);
    System.out.println(b);
} }
```

```
Output:
9
5
12
```

How much we have proceeded?

- **Introduction**
- **Checking Whether a String Is a Valid Number (parseInt())**
- **Wrapper Class and its Methods**
- **Converting Numbers to Objects and Vice Versa**

Storing a Larger Number in a Smaller Number

```
float f=3.0// won't even compile!
This line will be understood as follows.
double tmp=3.0;
float f=tmp;
```

How to fix?

- Can be fixed in one of the several ways:

1) By making the 3.0 a float (probably the best solution)

2) By making f a double

3) By putting in a cast

4) By assigning an integer value of 3, which will get "promoted“

Example:

```
float f=3.0f;
double f=3.0;
float f=(float)3.0;
float f=3;
```

Ensuring the Accuracy of Floating-Point Numbers

- In java integer division by 0 consider as logical error so it throws an ArithmeticException.
- Floating-point operations, however, do not throw an exception because they are defined over an (almost) infinite range of values.

Java act differently for the following cases.

1) Java signal errors by producing the constant POSITIVE_INFINITY if you divide a positive floating-point number by zero

2) It signal constant NEGATIVE_INFINITY if you divide a negative floating-point value by zero.

3) Produces NaN (Not a Number) if you otherwise generate an invalid result

- Values for these **three public constants** are defined in both the Float and the Double wrapper classes.
- The value NaN has the unusual property that it is not equal to itself (i.e., NaN != NaN).
- x==NaN never be true, instead, the methods Float.isNaN(float) and Double.isNaN(double) must be used.

Ensuring the Accuracy of Floating-Point Numbers

```
public static void main(String[] args){
    double d = 123;
    double e = 0;
    if (d/e == Double.POSITIVE_INFINITY)
        System.out.println("Check for POSITIVE_INFINITY works");
    double s = Math.sqrt(-1);
    if (s == Double.NaN)
        System.out.println("Comparison with NaN incorrectly returns true");
    if (Double.isNaN(s))
        System.out.println("Double.isNaN() correctly returns true");
}
```

Output:

```
Check for POSITIVE_INFINITY works
Double.isNaN() correctly returns true
```

Comparing Floating Point Numbers

- The equals() method of Float and Double wrapper class returns true if the two values are the same bit for bit (i.e., if and only if the numbers are the same or are both NaN).
- It returns false otherwise, including if the argument passed in is null, or if one object is +0.0 and the other is –0.0.
- To actually compare floating-point numbers for equality, it is generally desirable to compare them within **some tiny range of allowable differences**; this range is often regarded as a tolerance or as *epsilon*.

Example:

```
public class NumberTest {
    public static void main(String[] args) {
        float x=0.3f*3;
        if(x==0.9)
            System.out.println(x);
    }
}
Output:
```

Comparing Floating Point Numbers

```
public class FloatCmp {
   final static double EPSILON = 0.0000001;
   public static void main(String[] argv) {
      double da = 3 * .3333333333;
      double db = 0.99999992857;
      // Compare two numbers that are expected
      to be close.
      if (da == db) {
        System.out.println("Java considers " +
        da + "==" + db);
       // else compare with our own equals
       overload
      }
      else if (equals(da, db, 0.0000001)) {
        System.out.println("Equal within epsilon
        " + EPSILON);
      }
      else {
        System.out.println(da + " != " + db);
      }
}
```

```
/** Compare two doubles within a given
epsilon */
public static boolean equals(double a, double
b, double eps) {
   if (a==b)
      return true;
   // If the difference is less than epsilon,
   treat as equal.
   return Math.abs(a - b) < eps;
}
/** Compare two doubles, using default
epsilon */
public static boolean equals(double a, double
b) {
   return equals(a, b, EPSILON);
}
```

Rounding Floating-Point Numbers

- To round floating-point numbers properly, use Math.round() .
- It has two overloads:

  if you give it a double , you get a long result;

  if you give it a float , you get an int .

- If the argument is *NaN* (not a number), then the function will return 00.
- If the argument is negative infinity (**Float**) or any value less than or equal to the value of **Integer.MIN_VALUE**, then the function returns **Integer.MIN_VALUE**. (Try for Double.NEGATIVE_INFINITY)
- If the argument is positive infinity or any value greater than or equal to the value of Integer.MAX_VALUE, then the function returns Integer.MAX_VALUE. (Try for Double.POSITIVE_INFINITY)

Example:

```
double d=5.67;

System.out.println(Math.round(d));

float f=9.4255f;

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

End of Session