Numbers

By Dr. Subrat Kumar Nayak

Associate Professor

Department of CSE

ITER, SOADU

Operating on a Series of Integers

- For a contiguous set, use a for loop.

Example:

```
String months[]={
"January", "February", "March", "April",
"May", "June", "July", "August",
"September", "October", "November", "December"
};
for(int i=0;i<months.length;i++)
{
System.out.println("Month " + months[i])
}
```

Operating on a Series of Integers

- For discontinuous ranges of numbers, use a java.util.BitSet .

```
// A discontiguous set of integers, using a BitSet
// Create a BitSet and turn on a couple of bits.
String months[]={"January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December‟ };
BitSet b = new BitSet();
b.set(0);
// January
b.set(3);
// April
b.set(8);
// September
// Presumably this would be somewhere else in the code.
for (int i = 0; i<months.length; i++)
{
    if (b.get(i))
    System.out.println("Month " + months[i]);
}
```

Output:
Month January
Month April
Month September

Generating Random Numbers

- Use java.lang.Math.random() to generate random numbers.

Example:

```
//java.lang.Math.random( ) is static, don't need any constructor calls
System.out.println("A random from java.lang.Math is " + Math.random( ));
```

- If you need integers random number, construct a java.util.Random object and call its nextInt() method.

Example:

```
public class RandomInt {
public static void main(String[] a) {
Random r = new Random();
for (int i=0; i<1000; i++)
{
    // nextInt(10) goes from 0-9; add 1 for 1-10;
    System.out.println(1+r.nextInt(10));
}}}
```

Generating Random Numbers

Some other nextXxx()methods from java.util.Random class

- boolean nextBoolean() to find a random boolean .
- byte nextByte() to find a random boolean .
- int nextInt() to find a random int.
- float nextFloat() to find a random float.
- double nextDouble() to find next double.

Handling Very Large Numbers

- **Can you store 100! using any primitive data type?**
- In order to handle very large numbers java provides two classes from java.math package.

(1) BigInteger, to create a large integer number.

(2) BigDecimal, to create a large decimal number(Real number).

BigInteger

- BigInteger class is used for mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

Constructors:

- `BigInteger(String val)`

Translates the decimal String representation of a BigInteger into a BigInteger.

- `BigInteger(String val, int radix)`

Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Handling Very Large Numbers

Example:

```
import java.math.BigInteger;
public class BigIntTest {
        public static void main(String args[])
        {
                BigInteger bi=new BigInteger("1234455666666");
                System.out.println(bi);
                BigInteger bi1=new BigInteger("1111",2);
                System.out.println(bi1);
        }
}
```

```
Output:
1234455666666
15
```

Handling Very Large Numbers

Methods:

abs()

- It returns a BigInteger, whose value is the absolute value of this BigInteger.

Example:

```
import java.math.BigInteger;
public class BigIntegerAbsExample {
public static void main(String[] args){
BigInteger big1, big2, big3, big4;  // create 4 BigInteger objects
big1=new BigInteger("345");// assign value to big1
big2=new BigInteger("-345");    // assign value to big2
big3=big1.abs ();         // assign absolute value of big1 to big 3
big4=big2.abs ();        // assign absolute value of big 2 to big 4.
String str1 = "Absolute value of" + big1 + "is" + big3;
String str2 = "Absolute value of" + big2 + "is" + big4;
System.out.println(str1);
System.out.println(str2 );
} }
```

Handling Very Large Numbers

add()

- This method returns a BigInteger by simply computing 'this + val' value.

```
public class BigIntTest {
public static void main(String args[])
{
    BigInteger bi1=new BigInteger("1234455666456");
    BigInteger bi2=new BigInteger("1234455666456");
    BigInteger bi3=bi1.add(bi2);
    System.out.println("Sum of big integer= "+bi3);
}}}
```

```
Output:
Sum of big integer= 2468911332912
```

Handling Very Large Numbers

`BigInteger multiply(BigInteger val)`

- Returns a BigInteger whose value is (this * val).

`BigInteger divide(BigInteger val)`

- Returns a BigInteger whose value is (this / val).

`int compareTo(BigInteger val)`

- Compares this BigInteger with the specified BigInteger.

`boolean equals(Object x)`

- Compares this BigInteger with the specified Object for equality.

`float floatValue()`

- Converts this BigInteger to a float.

`int intValue()`

- Converts this BigInteger to a int.

Handling Very Large Numbers

BigDecimal()

BigDecimal class is used for mathematical operation which involves very big real number calculations that are outside the limit of all available primitive data types.

Constructors:

- `BigDecimal(String val)`

Translates the string representation of a BigDecimal into a BigDecimal object.

- `BigDecimal(BigInteger val)`

Translates a BigInteger into a BigDecimal.

Handling Very Large Numbers

Methods of BigDecimal

`BigDecimal abs()`

- Returns a BigDecimal whose value is the absolute value of this BigDecimal.

`BigDecimal add(BigDecimal augend)`

- Returns a BigDecimal whose value is (this + augend).

`BigDecimal divide(BigDecimal divisor)`

- Returns a BigDecimal whose value is (this / divisor).

`BigDecimal multiply(BigDecimal multiplicand)`

- Returns a BigDecimal whose value is (this * multiplicand). + multiplicand.scale()).

`int compareTo(BigDecimal val)`

- Compares this BigDecimal with the specified BigDecimal.

`boolean equals(Object x)`

- Compares this BigDecimal with the specified Object for equality.

End of Session