9.2 Defining Your Own Exception Classes

```
public class DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Dividing by Zero!");
    }
    public DivideByZeroException(String message)
    {
        super(message);
    }
}
```

- Must be derived from some already defined exception class
- Often the only methods you need to define are

Listing 9.5. A Programmer-Defined Exception Class - DivideByZeroException.java

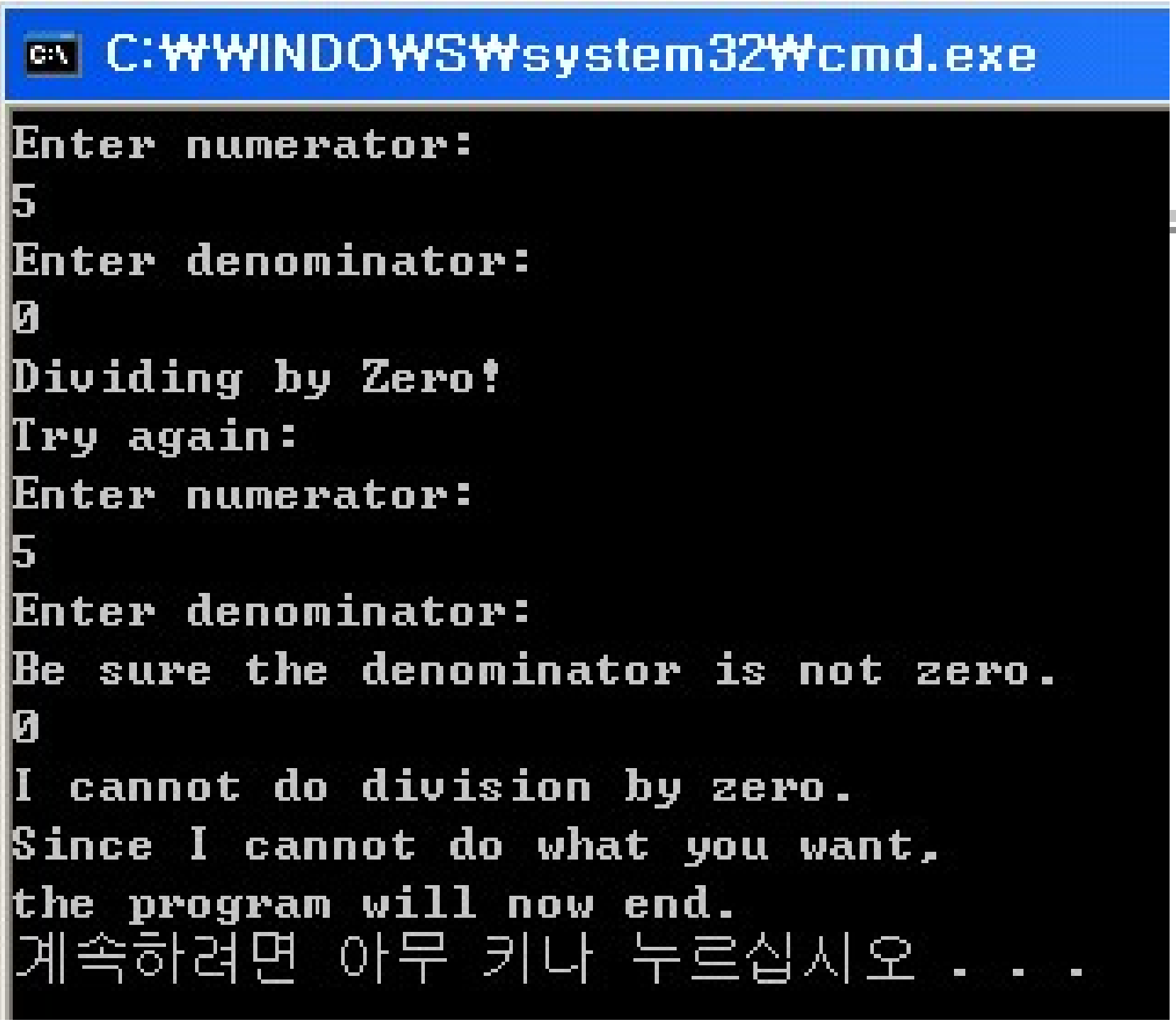
```
// Listing 9.5  A Programmer-Defined Exception Class

public class DivideByZeroException extends Exception
{

   // You Can do more in an exception constructor, but
   // this form is common
    public DivideByZeroException( )
    {
        super("Dividing by Zero!");
    }

    public DivideByZeroException(String message)
    {
  // super is an invocation of the constructor for the base class Exception.
        super(message);
    }
}
```

Listing 9.6. Using a Programmer-Defined Exception CLass - DivideByZeroDemo.java

```
import java.util.Scanner;

public class DivideByZeroDemo
{
   private int numerator;
   private int denominator;
   private double quotient;

   public static void main(String[] args)
   {
      DivideByZeroDemo oneTime = new DivideByZeroDemo( );
      oneTime.doIt( );
   }
```

```
public void doIt( )
  {
     try
     {
        System.out.println("Enter numerator:");
        Scanner keyboard = new Scanner(System.in);
        numerator = keyboard.nextInt( );
        System.out.println("Enter denominator:");
        denominator = keyboard.nextInt( );

        if (denominator == 0)
           throw new DivideByZeroException( );

        quotient = numerator / (double)denominator;
        System.out.println(numerator + "/" + denominator +
                           " = " + quotient);
     }
     catch(DivideByZeroException e)
     {
        System.out.println(e.getMessage( ));
        giveSecondChance( );
     }

     System.out.println("End of program.");
  }
```

```
public void giveSecondChance( )
  {
     System.out.println("Try again:");
     System.out.println("Enter numerator:");
     Scanner keyboard = new Scanner(System.in);

     numerator = keyboard.nextInt( );
     System.out.println("Enter denominator:");
     System.out.println("Be sure the denominator is not zero.");
     denominator = keyboard.nextInt( );

     if (denominator == 0)
     {
        System.out.println("I cannot do division by zero.");
        System.out.println("Since I cannot do what you want,");
        System.out.println("the program will now end.");
        System.exit(0);
     }

     quotient = ((double)numerator) / denominator;
                System.out.println(numerator + "/" + denominator +
                                                   " = " + quotient);
  }
}
```


C:\WINDOWS\system32\cmd.exe
Enter numerator:
5
Enter denominator:
0
Dividing by Zero!
Try again:
Enter numerator:
5
Enter denominator:
Be sure the denominator is not zero.
0
I cannot do division by zero.
Since I cannot do what you want,
the program will now end.
계속하려면 아무 키나 누르십시오 . . .

Java Tip: Preserve **`getMessage`** When You Define Exception Classes

```
throw new Exception("This is a big exception!");
```

This string is stored in an instance variable in the exception object and is returned by the **`getMessage`** method.

To preserve the correct `getMessage` behavior in `Exception` classes that you define, include:

- a constructor with a string parameter that begins with a call to `super`

```
public DivideByZeroException(String message)
{
    super(message);
}
```

- a default constructor that passes a default message to the `super` constructor

```
public DivideByZeroException()
{
    super("Dividing by Zero!");
}
```

When to Define Your Own Exception Class

- When you use a `throw`-statement in your code, you should usually define your own exception class.
- If you use a predefined, more general exception class, then your `catch`-block will have to be general.

»

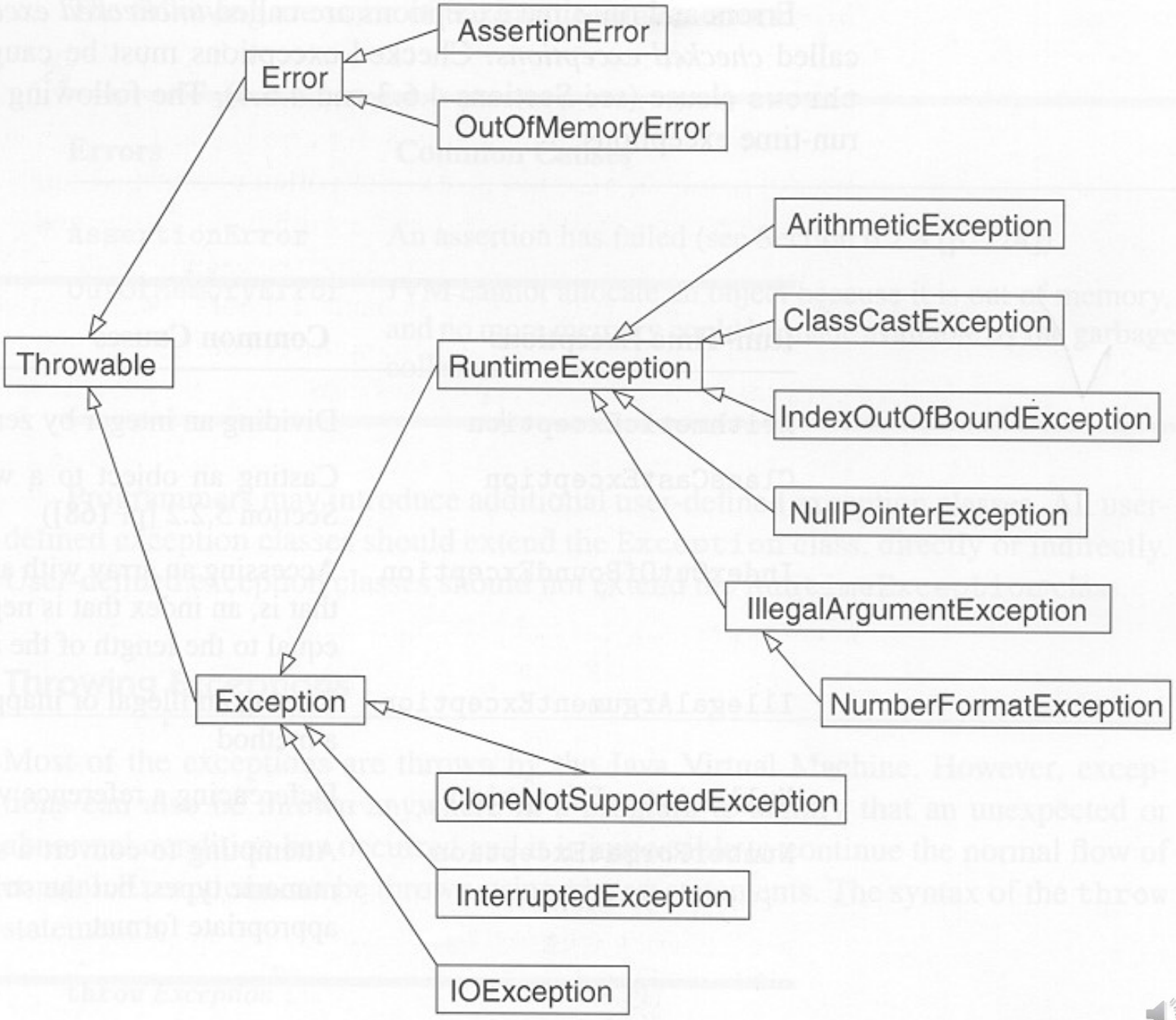
```
public void dolt( )
  {
     try
     {
        System.out.println("Enter numerator:");
        Scanner keyboard = new Scanner(System.in);
        numerator = keyboard.nextInt( );
        System.out.println("Enter denominator:");
        denominator = keyboard.nextInt( );

        if (denominator == 0)
           throw new Exception("Divide by Zero");

        quotient = numerator / (double)denominator;
        System.out.println(numerator + "/" + denominator +
                           " = " + quotient);
     }
     catch(Exception e)
     {
        System.out.println(e.getMessage( ));
        giveSecondChance( );
     }

     System.out.println("End of program.");
  }
```

- A general `catch`-block could also catch exceptions that should be handled somewhere else.
- A specific `catch`-block for your own exception class will catch the exceptions it should and pass others on.

Example: Using the **DivideByZeroException** Class

```
public void doIt( )
   {
      try
      {
         System.out.println("Enter numerator:");
         Scanner keyboard = new Scanner(System.in);
         numerator = keyboard.nextInt( );
         System.out.println("Enter denominator:");
         denominator = keyboard.nextInt( );

         if (denominator == 0)
            throw new DivideByZeroException( );

         quotient = numerator / (double)denominator;
         System.out.println(numerator + "/" + denominator +
                            " = " + quotient);
      }
    //  catch(DivideByZeroException e)
      catch(Exception e)
      {
         System.out.println(e.getMessage( ));
         giveSecondChance( );
      }

      System.out.println("End of program.");
   }
```

Excerpt from `DivideByZero-ExceptionDemo`


AssertionError
Error
OutOfMemoryError
ArithmeticException
ClassCastException
Throwable
RuntimeException
IndexOutOfBoundException
NullPointerException
IllegalArgumentException
Exception
NumberFormatException
CloneNotSupportedException
InterruptedException
IOException

Category	Description
`Error`	A subclass of `Throwable`. Errors are serious and fatal problems in programs. Errors are thrown by the
`Exception`	A subclass of `Throwable`. Exceptions can be thrown by any . All user-defined exceptions should be a subclass of `Exception`.
`RuntimeException`	A subclass of `Exception`. Run-time exceptions are caused by illegal operations and thrown by

- Error
- Run time Exception
  - » 잘못된 캐스트
  - » 배열의 경계 초과 접근
  - » Null Pointer 접근 (프로그램에 의한 에러)
- IOException
  - » 파일의 끝을 초과하여 읽음
  - » 잘못된 URL을 열려는 시도
  - » 존재하는 클래스를 나타내지 않는 문자열을 위한 클래스 객체를 찾으려는 시도 등

- Unchecked exceptions
  - » errors and run-time exception
  - » 컴파일러에 의해 check안됨
  - » try – catch 구문만 사용
- Checked exceptions
  - » must be caught or declared in the throws clause.
  - » 컴파일러에 의해 검사되어짐
  - » 예) IOException 등

컴파일러에 의한 checked exception 사례

```
class CheckedUncheckedExceptionTest {
   public static void main(String args[]) {
      int a[]=null;
      int i;
      i = a[0];
      System.out.println("i="+i);
      i = System.in.read();  //  (a)
      System.out.println("i="+i);
   }
}
```

- /*
  * Results:
  D:\javac CheckedUncheckedExceptionTest.java

  CheckedUncheckedExceptionTest.java:9:

```
        i = System.in.read();
                      ^
```

  1 error

- **read**
- public abstract int **read**() throws IOException
  - » 입력 스트림로부터 데이터의 다음의 바이트를 읽어들입니다. 값의 바이트는,0 ~ 255 의 범위의 int 로서 돌려주어집니다. 스트림의 마지막에 이르렀기 때문에 읽어들이는 바이트가 없는 경우는, 값 -1 이 돌려주어집니다. 입력 데이터를 읽어들일 수 있게 되는지, 파일의 마지막이 검출되는지, 또는 예외가 발생할 때까지, 이 메소드는 블록 합니다. 서브 클래스는, 이 메소드의 구현을 제공하지 않으면 안됩니다.
  - » **반환값:** 데이터의 다음의 바이트. 스트림의 마지막에 이르렀을 경우는 -1
  - » **예외:** IOException - 입출력 에러가 발생했을 경우

컴파일러에 의한 unchecked exception 사례

```
class CheckedUncheckedExceptionTest2 {
   public static void main(String args[]) {
      int a[]=null;
      int i;
      i = a[0];  //  (a)
      System.out.println("i="+i);
      try {
         i = System.in.read();
      }
      catch(java.io.IOException e) {
      }
      System.out.println("i="+i);
   }
}
```

```
/*
    * Results:
    D:\4>java CheckedUncheckedExceptionTest2

        at CheckedUncheckedExceptionTest2.main
    (CheckedUncheckedExceptionTest2.java:6)
```

- common run-time exception

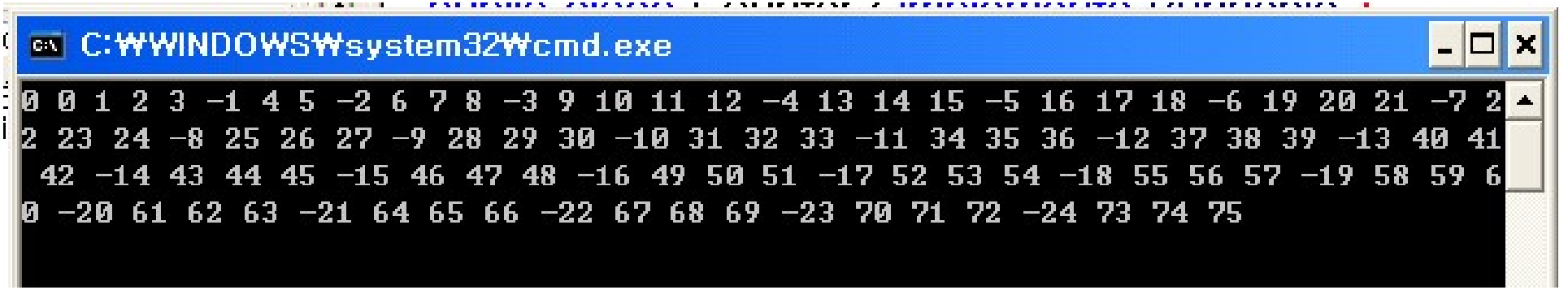
Run-Time Exceptions	Common Causes
`ArithmeticException`	Dividing an integer by zero
`ClassCastException`	Casting an object to a wrong class (see Section 5.2.2 [p. 168])
`IndexOutOfBoundException`	Accessing an array with an out-of-bound index, that is, an index that is negative, greater than or equal to the length of the array
`IllegalArgumentException`	Passing an illegal or inappropriate argument to a method
`NullPointerException`	Deferencing a reference variable that is `null`
`NumberFormatException`	Attempting to convert a string to one of the numeric types, but the string does not have the appropriate format

- Most Common checked exceptions

Exceptions	Common Causes
`CloneNotSupportedException`	Attempting to clone an object whose class does not implement the `Cloneable` interface (see Section 6.3.4 [p. 231])
`InterruptedException`	Interrupting a thread that is not running (see Section 11.1.2 [p. 553])
`IOException`	Encountering problems while performing input/output operations (see Section 8.4 [p. 366])

checked exceptions Test

```
public class Counter2 implements Runnable {
    protected int count;
    protected int inc;
    protected int delay;
    public Counter2(int init, int inc, int delay) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }
    public void run() {
        try {
            for (;;) {
                System.out.print(count + " ");
                count += inc;
                Thread.sleep(delay);
            }
        } catch (InterruptedException e) {}
    }

    public static void main(String[] args) {
        new Thread(new Counter2(0, 1, 33)).start();
        new Thread(new Counter2(0, -1, 100)).start();
    }
}
```


C:\WINDOWS\system32\cmd.exe
0 0 1 2 3 -1 4 5 -2 6 7 8 -3 9 10 11 12 -4 13 14 15 -5 16 17 18 -6 19 20 21 -7 2
2 23 24 -8 25 26 27 -9 28 29 30 -10 31 32 33 -11 34 35 36 -12 37 38 39 -13 40 41
42 -14 43 44 45 -15 46 47 48 -16 49 50 51 -17 52 53 54 -18 55 56 57 -19 58 59 6
0 -20 61 62 63 -21 64 65 66 -22 67 68 69 -23 70 71 72 -24 73 74 75

checked exceptions Test

```
public class Counter2 implements Runnable {
    protected int count;
    protected int inc;
    protected int delay;
    public Counter2(int init, int inc, int delay) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }
    public void run() {
//      try {
          for (;;) {
              System.out.print(count + " ");
              count += inc;
              Thread.sleep(delay);
          }
//      } catch (InterruptedException e) {}
    }

    public static void main(String[] args) {
        new Thread(new Counter2(0, 1, 33)).start();
        new Thread(new Counter2(0, -1, 100)).start();
    }
}
```

checked exceptions Test

```
D:\#과거강의-2004년-2005년\@@@@강의-객체지향-200502\@@@@@@oo\Chapter11\Counter2ExceptionTest.java:18: unreported exception java.lang.InterruptedException; must be caught or declared to be thrown
                Thread.sleep(delay);
                       ^
1 error

Tool completed with exit code 1
```

Thread Class

- **Sleep**
  - » public static void **sleep**(long millis) throws InterruptedException
  - » Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.
  - » **Parameters:**
    - – millis - the length of time to sleep in milliseconds.
  - » **Throws:**
    - – InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.
  - » **See Also:**
    - – Object.notify()

Examples of Exception

- Ex . The Maximun of Two Integers with Exception Handling
  - » to illustrate the handling of exceptions in the Maximum program in (Ex)
  - » NumberFormatException

```
venus% java Maximum2 12 11
The maximum of 12 and 11 is: 12
```

```
venus% java Maximum2 eleven twelve
Invalid input value: eleven
The input values must be integers.
```

The maximum of two integer arguments: Maximum.java

```
public class Maximum {

   public static void main(String[] args) {
      if (args.length >= 2) {
         int i1 = Integer.parseInt(args[0]);
         int i2 = Integer.parseInt(args[1]);
         System.out.println("The maximum of " + i1 + " and " + i2 +
                            " is: " + ((i1 >= i2) ? i1 : i2));
      } else {
         System.out.println("Usage: java Maximum integer1 integer2");
      }
   }
}
```

```
venus% java Maximum 12 11
The maximum of 12 and 11 is: 12
```

```
C:₩maximum>java Maximum one two
Exception in thread "main" java.lang.                    : For input string: "o
ne"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at Maximum.main(Maximum.java:16)

C:₩maximum>java Maximum 1 2
The maximum of 1 and 2 is: 2

C:₩maximum>
```

The maximum of two integer arguments: Maximum2.java

```
public class Maximum2 {
   public static void main(String[] args) {
      if (args.length >= 2) {
         try {
            int i1 = Integer.parseInt(args[0]);
            int i2 = Integer.parseInt(args[1]);
            System.out.println("The maximum of " + i1 + " and " +
                               i2 + " is: " + ((i1 >= i2) ? i1 : i2));
         } catch (NumberFormatException e) {
            System.out.println("Invalid input value: " +
                               e.getMessage());
            System.out.println("The input values must be integers.");
         }
      } else {
         System.out.println("Usage: java Maximum integer1 integer2");
      }
   }
}
```