Exception Handling

Exception-Handling Fundamentals

- ***Exception**:* an abnormal condition that arises in a code sequence at run time/ run time error
  - Java exception is an object that describes an exceptional (that is, error) condition
  - When an exceptional condition arises, an object representing that exception is created and ***thrown*** in the method that caused the error
- Exception can be
  - generated by the Java run-time system (relate to fundamental errors that violate the rules of the Java language)
  - Manually generated (typically used to report some error condition to the caller of a method)

Keywords for exception handling

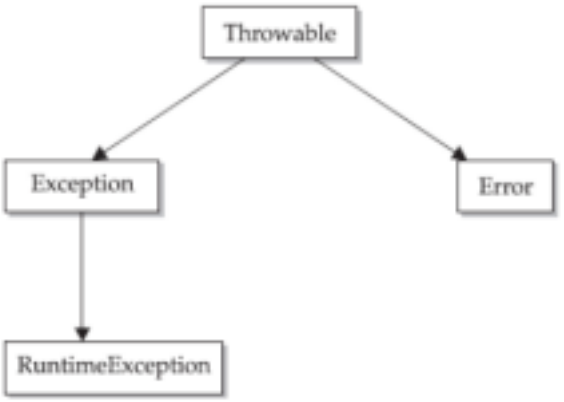
- **try** block contains program statements that are to be to monitored for exceptions
  - If an exception occurs within the **try** block, it is thrown
- **catch** block contain statements that catch this exception and handle it in some rational manner
  - System-generated exceptions are automatically thrown by the Java runtime system.
- **throw** is used to manually throw an exception
- **throws** clause is used to specify any exception that is thrown out of a method
- **finally** block contains code that absolutely must be executed after a **try** block completes

General form of an exception-handling block

```
try {
// block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// ...
finally {
// block of code to be executed after try block ends
}
```

Exception Types

- All exception types are subclasses of the built-in class **Throwable**
- Below **Throwable** are two subclasses that partition exceptions into two branches
  - **Exception** class: used for exceptional conditions that user programs should catch
    - RuntimeException etc are subclasses of Exception
  - **Error** class: used by the Java run-time system to indicate errors having to do with the run-time environment (eg Stack overflow)



Uncaught Exceptions

- Uncaught exception :caught by the default handler (provided by the Java run-time system) that
  - displays a string describing the exception
  - prints a stack trace from the point at which the exception occurred
  - terminates the program

java.lang.ArithmeticException: / by zero
at Exc0.main(Exc0.java:4)

Using try and catch

- Exception handling benefits
  - allows you to fix the error
  - prevents the program from automatically terminating

```
class Exc2 {
  public static void main(String args[]) {
    int d, a;
    try { // monitor a block of code.
       d = 0;
       a = 42 / d;
       System.out.println("This will not be printed.");
    } catch (ArithmeticException e) {
         // catch divide-by-zero error
            System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
  }
}
```

Using try and catch

```
class Exc2 {
  public static void main(String args[]) {
    int d, a;
    try { // monitor a block of code.
       d = 0;
       a = 42 / d;
       System.out.println("This will not be printed.");
    } catch (ArithmeticException e) {
         // catch divide-by-zero error
            System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
  }
}
```

OUTPUT
Division by zero.
After catch statement.

Another Example

```
// Handle an exception and move on.
import java.util.Random;
class HandleError {
   public static void main(String args[]) {
        int a=0, b=0, c=0;
        Random r = new Random();
        for(int i=0; i<32000; i++) {
        try {
           b = r.nextInt();
           c = r.nextInt();
           a = 12345 / (b/c);
        } catch (ArithmeticException e) {
              System.out.println("Division by zero.");
              a = 0; // set a to zero and continue
        }
        System.out.println("a: " + a);
     }
  }
}
```

Displaying a Description of an Exception

- **Throwable** overrides the **toString( )** method (defined by **Object**) so that it returns a string containing a description of the exception

```
catch (ArithmeticException e) {
System.out.println("Exception: " + e);
    a = 0; // set a to zero and continue
}
```

OUTPUT
Exception: java.lang.ArithmeticException: / by zero

Multiple catch Clauses

- Use when more than one exception could be raised by a single piece of code
- When an exception is thrown, each **catch** statement is inspected in order, and the first one whose type matches that of the exception is executed
  - All others are bypassed
  - execution continues after the **try / catch** block
- NOTE: exception subclasses must come before any of their superclasses because a **catch** statement that uses a superclass will catch exceptions of that type plus any of its subclasses

Multiple catch Clauses (Example)

```
class MultipleCatches {
 public static void main(String args[]) {
  try {
    int a = args.length;
    System.out.println("a = " + a);
    int b = 42 / a;
    int c[] = { 1 };
    c[42] = 99;
  } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
  } catch(ArrayIndexOutOfBoundsException e) {
       System.out.println("Array index oob: " + e);
  }
  System.out.println("After try/catch blocks.");
 }
}
```

Multiple catch Clauses (Example)

OUTPUT

C:\>java MultipleCatches
a = 0
Divide by 0: java.lang.ArithmeticException: / by zero
After try/catch blocks.
C:\>java MultipleCatches TestArg
a = 1
Array index oob:
java.lang.ArrayIndexOutOfBoundsException:42
After try/catch blocks.

Nested try Statements

- a **try** statement can be inside the block of another **try**
- Each time a **try** statement is entered, the context of that exception is pushed on the stack
- If an inner **try** statement does not have a **catch** handler for a particular exception, the stack is unwound and the next **try** statement's **catch** handlers are inspected for a match
- This continues until one of the **catch** statements succeeds, or until all of the nested **try** statements are exhausted
- If no **catch** statement matches, then the Java run-time system will handle the exception

Nested try Statements (Example)

```
class NestTry {
  public static void main(String args[]) {
  try {
     int a = args.length;
     int b = 42 / a;
     System.out.println("a = " + a);
     try { // nested try block
        if(a==1) a = a/(a-a); // one command-line arg, division by zero
        if(a==2) {
           int c[] = { 1 };
           c[42] = 99; // two command-line args, out-of-bounds exception
        }
      } catch(ArrayIndexOutOfBoundsException e) {
           System.out.println("Array index out-of-bounds: " + e);
      }
    } catch(ArithmeticException e) {
          System.out.println("Divide by 0: " + e);
    }
  }
}
```

Nested try Statements (Example)

OUTPUT

```
C:\>java NestTry
Divide by 0: java.lang.ArithmeticException: / by zero
C:\>java NestTry One
a = 1
Divide by 0: java.lang.ArithmeticException: / by zero
C:\>java NestTry One Two
a = 2
Array index out-of-bounds:
java.lang.ArrayIndexOutOfBoundsException:42
```