THE JAVA THROW, TRY AND CATCH STATEMENTS

In our natural world, an exception is a situation that requires special treatment such as when you order a Philly cheese steak sandwich in a delicatessen but ask them to “hold the cheese.” Executing computer code can also run into situations that require special handling.

Example
Consider this statement that calls the Scanner object’s nextDouble method:

```java
salary = in.nextDouble();
```

Suppose that when executed the user enters input of $2,125.5. Since nextDouble can only scan tokens that represent numbers, it cannot handle this situation.

In the olden days, computer programmers were at the mercy of the operating system, which generated a run-time error and halted any executing program that encountered an exceptional situation. Nowadays, modern programming languages have incorporated facilities that allow us to detect and respond to exceptions that occur during execution. Java’s facilities for doing this use the statements try and catch. Also, Java goes further and allows you to create your own exceptions using the throw statement.

Java Throw Statement

```
throw exception object;
```

exception object is an object of class java.lang.Exception or some subclass of it.

When JVM encounters a throw statement, it stops executing the block in which the throw occurs and will not execute any additional code in the program unless it encounters a proper catch statement.

Example

```java
1 // cannot take square root of a negative number
2 if ( x < 0 )
3    throw new Exception();
4 // x not negative, OK to compute
5 s = Math.sqrt( x );
```
During the execution of a program, an exception may never be thrown. But once thrown, an exception must be “caught” or else the program will halt with a run-time error message.

**Example**
This application illustrates the semantics of the `throw` statement.

```java
import java.util.*;

public class ThrowException {
  public static void main( String args [ ] ) {
    int n;
    Scanner in = new Scanner( System.in );
    System.out.print( "Enter a whole number: " );
    // expecting an integer
    if ( ! in.hasNextInt( ) )
      throw new InputMismatchException( );
    // next token is integer, OK to scan
    n = in.nextInt( );
    System.out.println( "n is " + n + ", done." );
  }
}
```

If the user inputs an integer then the program runs to completion and terminates normally. For example,

```
Enter a whole number: 10
n is 10, done.
```

On the other hand, non-integer input causes the program to halt with a run-time error.

```
Enter a whole number: ten
Exception in thread "main" java.util.InputMismatchException
```
To catch an exception, you use the `catch` statement, which is always used along with a `try` statement.

### Java Try and Catch Statement Syntax

```
try
{
    statements with the potential to throw an exception
}
catch ( exception class 1  reference variable )
{
    statements handling exception class 1
}
catch ( exception class 2  reference variable )
{
    statements handling exception class 2
}
...
statements following the catch statements
```

You cannot have a `catch` statement without it being preceded by a `try` statement.

You can, however have several `catch` statements following a `try`.

The `try` statement encloses any code that potentially throws an exception that you want to catch.

The parentheses of the `catch` statement contain the declaration of a reference variable whose data type is `java.lang.Exception` or some subclass of it. This variable “catches” the object thrown by the `throw` statement. Its scope is restricted to the block of the `catch` statement.
Java Try and Catch Statement Semantics

What the computer executes:
1. Execute *statements with the potential to throw an exception*
2. Execute *statements handling exception class 1*
3. Execute *statements handling exception class 2*
4. ... (none of the above)
5. Proceed to execute *statements following the catch statements*

As illustrated in the picture, the computer begins by (1) executing the *statements with the potential to throw an exception* that are within the *try* statement.

If no exception is thrown then execution (2) jumps over all the *catch* statements and continues with the *statements following the catch statements*.

If an exception is thrown then the computer (3) branches to the *catch* statement for which the class of the thrown object is an *exception class*. The reference variable becomes a reference to the thrown object and the computer executes the *statements handling exception class* that are within the *catch*. After finishing the *catch*, the computer (4) jumps over all remaining *catch* statements and continues with the *statements following the catch statements*.

If none of the *exception classes* match that of the thrown object then the computer (5) skips any code after the exception was thrown, skips all the *catch* statements and resumes its search for a *catch* statement at a higher level.
Example
This application illustrates the semantics of the `try` and `catch` statements.

```java
import java.util.*;

public class TryCatchDemo1 {
    public static void main( String args [] ) {
        int age;
        Scanner in = new Scanner( System.in );
        try {
            System.out.print( "Enter your age: " );
            // nextInt may throw an exception
            age = in.nextInt();
            // it didn't
            System.out.println( age + " years" );
        } // nextInt threw an exception
        catch ( InputMismatchException e ) {
            System.out.println( "Bad integer" );
        }
        System.out.println( "Goodbye" );
    }
}
```

For user input of **16**, the call to `nextInt` on line 13 successfully reads it, continues with line 15 and jumps over the `catch` to finish with line 22.

---

For user input of **sixteen**, the call to `nextInt` on line 13 throws an `InputMismatchException` object, which is caught at line 18. Line 20 executes after which the application finishes with line 22.

---

Enter your age: 16
16 years
Goodbye

Enter your age: sixteen
Bad integer
Goodbye
The JVM considers the catch statements in order and picks the first one where the class of the thrown object IS_AN exception class of the catch. Thus, if exception class A IS_A B then you must put A’s catch before B’s to insure that it will be executed.

Example
This application gives this syntax error at line 21:

    exception InputMismatchException has already been caught

Because InputMismatchException IS_A RuntimeException and so must be placed first.

```java
import java.util.*;

public class TryCatchDemo2 {
    public static void main( String args [] ) {
        int age;
        Scanner in = new Scanner( System.in );
        try {
            System.out.print( "Enter your age: " );
            age = in.nextInt( );
            if ( age < 0 )
                throw new RuntimeException( );
            System.out.println( age + " years" );
        }
        catch ( RuntimeException e ) {
            System.out.println( "Age cannot be negative" );
        }
        catch ( InputMismatchException e ) {
            System.out.println( "Bad integer" );
        }
        System.out.println( "Goodbye" );
    }
}
```
Exercises

Enter the `TryCatchDemo2` application into jGRASP and fix the compiler error. Do the exercises that follow.

1. Run the program; enter `16` in response to the user prompt. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

2. Run the program; enter `-16` in response to the user prompt. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

3. Run the program; enter `sixteen` in response to the user prompt. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

Enter the following application into jGRASP. Save it to a file and compile it. Do the exercises that follow.

```java
import static javax.swing.JOptionPane.*;

public class TryCatchDemo3
{
    public static void main( String args [ ] )
    {
        String prompt, input;
        prompt = "Enter your age";
        int age;
        try
        {
            input = showInputDialog( prompt );
            if ( input == null )
                throw new NullPointerException( );
            // parseInt may throw NumberFormatException
            age = Integer.parseInt( input );
            if ( age < 0 )
                throw new IllegalArgumentException( );
            // no exceptions
        }
```
showMessageDialog( null, age + " years" );
}
catch ( NullPointerException e )
{
    showMessageDialog( null, "Operation cancelled" );
}
catch ( NumberFormatException e )
{
    showMessageDialog( null, e.getMessage() + " is not an integer" );
}
catch ( IllegalArgumentException e )
{
    showMessageDialog( null, "Age cannot be negative" );
}
showMessageDialog( null, "Goodbye" );
}

4. Run the program; enter 16 in the input dialog and click OK. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

5. Run the program; cancel or close the input dialog. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

6. Run the program; enter sixteen in the input dialog and click OK. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

7. Run the program; click OK on the input dialog without entering any input. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.

8. Run the program; enter -16 in the input dialog and click OK. What is the program’s output? By drawing on the program listing with a pencil, describe the execution path through the program.