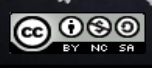
Introduction to Java Exceptions

try

throw

catch

Dan McElroy



Oct 2018

Definition from the Textbook

"The term exception is used to refer to the type of error that one might want to handle with a try..catch. An exception is an exception to the normal flow of control in the program. The term is used in preference to 'error' because in some cases, an exception might not be considered to be an error at all. You can sometimes think of an exception as just another way to organize a program."

Introduction to Programming Using Java, David Eck, section 3.7.1

try…throw

Sometimes conditions will occur that can cause a program to crash. These could include trying to divide by zero, the wrong data type entered at the keyboard when using Scanner, trying to open an non existing file, etc. Place code that might cause an error inside a **try** block. If an error occurs within the try block, execution is immediately jumped out of the try block to see if there is a **catch** block that can respond to the error.

No Problems, skip the catch block

```
try {
    doSomething;
    doSomeMore;
    doEvenMore;
}
catch ( exceptionType, identifier) {
    doSomething;
    doSomeMore;
}
moreCodeAfterTryCatch;
someMoreCode;
```

OH NO! Something happened. Jump out to the catch block

```
try {
    doSomething;
    doSomeMore;
    doEvenMore;
}
catch ( exceptionType, identifier) {
    doSomething;
    doSomeMore;
}
moreCodeAfterTryCatch;
someMoreCode;
```

Using **try…catch** in the Paycheck Project

```
import java.util.InputMismatchException;
    • • •
try {
   // INPUT: hours and payRate
    System.out.print ("Enter the hours worked: ");  // prompt message
    hours = stdin.nextDouble();            // input from the keyboard
    System.out.print ("Enter the pay rate: ");
    payRate = stdin.nextDouble();
}
catch (InputMismatchException   e) {
     System.out.println ("Values for hours and pay rate must be numeric");
     return; // no more processing
}
```

Using **try…catch** in the Paycheck Project

```
import java.util.InputMismatchException;
    • • •
try {
   // INPUT: hours and payRate
    System.out.print ("Enter the hours worked: ");  // prompt message
    hours = stdin.nextDouble();            // input from the keyboard
    System.out.print ("Enter the pay rate: ");
    payRate = stdin.nextDouble();
}
catch (InputMismatchException   e) {
     System.out.println ("Values for hours and pa
     return; // no more processing
}
```

If Scanner (as stdin here) is expecting a number from the keyboard and something else is entered, an exception occurs and the program immediately jumps out of the try block to see if the error is one that can be caught.

Built-in Exceptions

Java has several exceptions that are built-in. For example, Scanner can cause an exception when the user inputs ***fourty*** when Scanner is expecting an integer or double. When this occurs inside a **try** block Scanner can **throw** the exception and the program will jump out of the **try** block. However, no error occurs if the user enters -1 when Scanner is expecting an integer or double.

Built-in Exceptions

Arithmetic Exception
ArrayIndexOutOfBoundException
ClassNotFoundException
FileNotFoundException
IOException
InterruptedException
NoSuchFieldException
NoSuchMethodException
NullPointerException
NumberFormatException
RuntimeException
StringIndexOutOfBoundsException

try…throw…catch

If an exception of a type that is not automatically thrown within a try block, it is up to the programmer to throw the exception. The next example shows how an exception can be thrown when a negative number is detected. Note that there are two catch blocks that follow the try block. When an exception is thrown, Java will look at the first **catch** block to see if the exception can be handled. If not, Java looks at the next catch block. A fatal error will occur if there are no catch blocks that can handle an exception that is thrown.

try…throw…catch Example

```
import java.util.InputMismatchException;
    • • •
try {
       System.out.print ("Enter the hours worked: ");  // prompt message
       hours = stdin.nextDouble();            // input from the keyboard
       System.out.print ("Enter the pay rate: ");
       payRate = stdin.nextDouble();
       if (hours < 0 || payRate < 0)    // test for negative input values
         throw new IllegalArgumentException ("Inputs must be positive");
}
catch (InputMismatchException   e) {
      System.out.println ("Values for hours and pay rate must be numeric");
      return; // no more processing
}
catch (IllegalArgumentException e) {
      System.out.println (e.getMessage());
      return; // no more processing
}
```

Is Exception Handling Always Needed ???

Although the test for a negative input value is placed inside the **try** block in the previous example, the same thing can be accomplished without using exception handling. Look at the next example, the test for values less than zero is placed after the catch block and it works fine.

Exception not needed to test for less than zero

```
import java.util.InputMismatchException;
    • • •
try {
       System.out.print ("Enter the hours worked: ");  // prompt message
       hours = stdin.nextDouble();            // input from the keyboard
       System.out.print ("Enter the pay rate: ");
       payRate = stdin.nextDouble();
}
catch (InputMismatchException   e) {
      System.out.println ("Values for hours and pay rate must be numeric");
      return; // no more processing
}
if (hours < 0 || payRate < 0) {   // test for negative input values
      System.out.println ("Inputs must be positive");
      return; // no more processing
}
```