TRY..CATCH IN JAVA

When an exception occurs, we say that the exception is "thrown". For example, we say that `Integer.parseInt(str)` throws an exception of type *NumberFormatException* when the value of `str` is illegal. When an exception is thrown, it is possible to "catch" the exception and prevent it from crashing the program. This is done with a try..catch statement. In somewhat simplified form, the syntax for a `try..catch` is:

```
try {
   statements-1
}
catch ( exception-class-name  variable-name ) {
   statements-2
}
```

The **exception-class-name** could be *NumberFormatException*, *IllegalArgumentException*, or some other exception class. When the computer executes this statement, it executes the statements in the `try` part. If no error occurs during the execution of **statements-1**, then the computer just skips over the `catch` part and proceeds with the rest of the program. However, if an exception of type **exception-class-name** occurs during the execution of **statements-1**, the computer immediately jumps to the `catch` part and executes **statements-2**, skipping any remaining statements in **statements-1**. During the execution of**statements-2**, the **variable-name** represents the exception object, so that you can, for example, print it out. At the end of the `catch` part, the computer proceeds with the rest of the program; the exception has been caught and handled and does not crash the program. Note that only one type of exception is caught; if some other type of exception occurs during the execution of **statements-1**, it will crash the program as usual.

By the way, note that the braces, { and }, are part of the syntax of the `try..catch` statement. They are required even if there is only one statement between the braces. This is different from the other statements we have seen, where the braces around a single statement are optional.

As an example, suppose that `str` is a variable of type *String* whose value might or might not represent a legal real number. Then we could say:

```
try {
   double x;
   x = Double.parseDouble(str);
   System.out.println( "The number is " + x );
}
catch ( NumberFormatException e ) {
   System.out.println( "Not a legal number." );
}
```

If an error is thrown by the call to `Double.parseDouble(str)`, then the output statement in the `try` part is skipped, and the statement in the `catch` part is executed.

It's not always a good idea to catch exceptions and continue with the program. Often that can just lead to an even bigger mess later on, and it might be better just to let the exception crash the program at the point where it occurs. However, sometimes it's possible to recover from an error. For example, suppose that we have the enumerated type

```
enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
SATURDAY, SUNDAY }
```

and we want the user to input a value belonging to this type. `TextIO` does not know about this type, so we can only read the user's response as a string. The function `Day.valueOf` can be used to convert the user's response to a value of type *Day*. This will throw an exception of type *IllegalArgumentException* if the user's response is not the name of one of the values of type *Day*, but we can recover from the error easily enough by asking the user to enter another response. Here is a code segment that does this. (Converting the user's response to upper case will allow responses such as "Monday" or "monday" in addition to "MONDAY".)

```
Day  weekday;  // User's response as a value of type Day.
while ( true ) {
   String response;  // User's response as a String.
   System.out.print("Please enter a day of the week: ");
   response = TextIO.getln();
   response = response.toUpperCase();
   try {
      weekday = Day.valueOf(response);
      break;
   }
   catch ( IllegalArgumentException e ) {
      System.out.println( response + " is not the name of a day
of the week." );
   }
}
// At this point, a legal value has definitely been assigned to
weekday.
```

The `break` statement will be reached only if the user's response is acceptable, and so the loop will end only when a legal value has been assigned to `weekday`.

Source : http://math.hws.edu/javanotes/c3/s7.html