Exceptions

Exceptions

When writing programs, it's good to think about all the situations where things could go wrong. For example, if you ask the user to input an integer, you should probably check to see if they actually did.

Java has a special class called `Exception` that is used when things go wrong in a program. If you reach a certain error state (such as getting bad input), you can **throw an exception** that describes the error. You can also **try** troublesome code, and then **catch** (and handle) any problems that might have happened.

Try/Catch Block

Here's the format for trying code that might cause problems, and then catching and handling the errors:

```
try {
     code that might cause problems
}
catch (Exception e) {
     handle the error
}
```

Suppose I try to read from a file that doesn't exist. That could certainly cause problems! To handle this, I'm going to try reading from a file, and will catch any problems:

```
Scanner console = new Scanner(System.in);
try {
     System.out.print(“Input the name of a file: “);
     String name = console.nextLine();
     Scanner fileIn = new Scanner(new File(name));
     String line = fileIn.nextLine();
     System.out.println(“1st line: “ + line);
}
catch (Exception e) {
     System.out.println(“Error reading file.”);
     System.exit(1);     //exits the program
}
```

In this example, ask the user for the name of an input file. We try reading and printing the first line of that file. If something goes wrong, we catch the exception, print an error message, and exit the program.

Throwing Exceptions
You can also throw your own exceptions if something unexpected happens in your code. To do that, say:

```
throw new Exception(“description of problem”);
```

when the unexpected state occurs. If the error occurs in a method, you can throw an exception instead of returning a value. Here’s an example:

```
public void divide(int a, int b) {
     if (b == 0) {
          throw new Exception(“Division by zero”);
     }
     else return a/b;
}
```

When I call `divide`, it will either return a value, or it will throw an exception. I could try/catch my call to divide as follows:

```
int a, b;
//get values for a and b from user
try {
     int result = divide(a, b);
     System.out.println(result);
}
catch(Exception e) {
     //will print “Division by zero”
     System.out.println(e.getMessage());
}
```

If I use the try/catch block, the program will not crash if I try to divide by zero. Alternatively, if I call `divide` without using a try/catch, the program will crash and the exception message (“`Division by zero`”) will get printed.

Common Exceptions
The Java `Exception` class can be used for all types of unexpected problems. However, there are also a number of specific types of exceptions you can use instead. It’s best to throw exceptions that are specifically designed for the type of problem you’re having. It’s also best to catch specific kinds of exceptions so you don’t end up with a catch block that catches a bunch of different possible problems. Here are the most common Java exceptions:

`IOException` – input/output problems, or trouble with files

`NullPointerException` – access method/variable on a variable that has the value `null` (the default value for objects)

`IllegalArgumentException` – an unexpected argument value in a method

`NumberFormatException` – try to convert something that doesn't have the right format (like converting "Bob" to an int)

`ArrayIndexOutOfBoundsException` – try to access an element beyond the bounds of an array

Now, instead of throwing/catching `Exceptions`, you can throw/catch these more specific exceptions. For example, if we revisit the divide method:

```
public void divide(int a, int b) {
      if (b == 0) {
            throw new IllegalArgumentException("Division by zero");
      }
      else return a/b;
}
```

Since `b==0` is a bad value for the divide argument, we throw an `IllegalArgumentException`. Now, we can also catch an `IllegalArgumentException`:

```
      int a, b;
      //get values for  a  and  b  from user
      try {
            int result = divide(a, b);
            System.out.println(result);
      }
      catch(IllegalArgumentException e) {
            //will print a descriptive error message
            System.out.println(e.getMessage());
      }
```

You can also have more than one catch block if more than one type of problem might occur. For example, if we read in a line from a file and then try to convert it to a number, there are two possible problems: there might be an error reading from the file, and there might be an error converting the input to a number. Here's how to catch both problems:

```
      try {
            //read the 1st line of nums.txt, convert it to an int
            Scanner fileIn = new Scanner(new File("nums.txt"));
            String line = fileIn.nextLine();
            int val = Integer.parseInt(line);
            System.out.println("First number: " + val);
      }
      catch (IOException e) {
            System.out.println(“Error reading file.”);
      }
      catch (NumberFormatException e) {
            System.out.println(“1st line of file not an int.”);
      }
```

Now, we will get an error message specific to the type of problem that occurs.