Erroneous Conditions

Consider the following erroneous implementation of a stack of integers using a linked list.

```
public class Stack {
    StackNode head;

    public Stack() { head = null; }

    public int pop() {
        int item = head.Item();
        head = head.Next();
        return item;
    }
    :
}
```

Let the calling method be this:

```
public class Test {
   public static void main (String args[]) {
      Stack s = new Stack();
      int item;
      for (int i = 0; i < 10; ++i) {
         item = s.pop();
         :
         item = s.pop();
      }
      item = s.pop();
   }
}
```

When we run this program we get the following error message:

```
Exception in thread 'main' java.lang.NullPointerException
         at Stack.pop(Stack.java:7)
         at Test.main(Compiled Code)
```

One way of fixing this problem is to add code to `Test` class to check whether the stack is empty before trying to perform a `pop` operation:

```
public class Test {
    public static void main (String args[]) {
        Stack s = new Stack();
        int item;
        for (int i = 0; i < 10; ++i) {
            if (not s.isEmpty()) then {
                item = s.pop();
                ⋮
            }
            else {
                //Code to deal with error
            }
            ⋮
            if (not s.isEmpty()) then {
                item = s.pop();
                ⋮
            }
            else {
                //Code to deal with error
            }
        }
    }
}
```

Exceptions

In the above solution class `Test` is cluttered with error-handling code.

A better solution is by using exceptions:

```
public class Stack {
   StackNode head;
   public Stack() { head = null; }
   public int pop() throws StackEmptyException{
      if (head == null)
         throw new StackEmptyException(``Stack empty'');
      else {
         int item = head.Item();
         head = head.Next();
         return item;
      }
   }
   ⋮
}
```

Ignoring Exceptions

Since now class `Stack` checks that the stack is not empty before popping an element off, it seems that the calling class `Test` does not need any error handling code:

```
public class Test {
   public static void main (String args[]) {
      Stack s = new Stack();
      int item;
      for (int i = 0; i < 10; ++i) {
         item = s.pop();
         ⋮
         item = s.pop();
      }
      item = s.pop();
   }
}
```

However, when compiling this class we get this error:

```
Test.java.9:  Exception StackEmptyException must be caught,
or it must be declared in the throws clause of this method.
```

item = s.pop();

1 error

Catching and Re-Throwing Exceptions

The calling method can either catch an exception or it can re-throw it.

- We catch the exception if this method knows how to deal with the error.
- Otherwise, the exception is re-thrown

Catching Exceptions

```
public class Test {
   public static void main (String args[]) {
      Stack s = new Stack();
      int item;
      try {
         for (int i = 0; i < 10; ++i) {
            item = s.pop();
            :
            item = s.pop();
         }
         item = s.pop();
      }
      catch (StackEmptyException e) {
         // Error handling code
         System.out.println(e.getMessage());
      }
   }
}
```

Re-Throwing Exceptions

```
public class Test {
   static void helper(Stack s) throws StackEmptyException{
      ⋮
      int item = s.pop();
      ⋮
   }
   public static void main (String args[]) {
      Stack s = new Stack();
      int item;
      try {
         ⋮
         helper();
         ⋮
         item = s.pop();
      }
      catch (StackEmptyException e) {
         // Error handling code
      }
   }
}
```

Declaring Exception Classes

We must declare classes of new exceptions we are going to throw. The usual way to do this is to make the class a subclass of `Exception`:

```
class StackEmptyException extends Exception {
    public StackEmtpyException(String message) {
        super(message);
    }
}
```

`Exception` has a method `getMessage()`, which returns the string we gave to the constructor.