Overview

We are continuing to cover the Java's exception/error handling mechanism. The previous lesson described the difficulties inherent in handling errors in programs, described the basic of Java's mechanism for error handling, and brought us as far as the class hierarchy (rooted at the class Throwable) of exceptions in the Java API. In this class, we will see more fully how inheritance, polymorphism and object-oriented design combine to provide a flexible, extensible, and relatively easy to use mechanism for handling errors and exceptional considtions.

What you get for free from inheritance

You should look at the Java API documentation for class Throwable. There are a few methods you get from Throwable that are really important:

- printStackTrace() - with this you can print the function call stack as it appeared when the problem giving rise to this error or exception occurred. This is good information for debugging or for logfiles. It comes in a few different flavors depending on where you want to output to go.
- getMessage() - Each Throwable has a message associated with it, which is a String that was passed to the Throwable's constructor or, if none was passed, an empty message ... which is not very helpful. So if you create Throwables, you usually want to do it like this:

  ```
  throw new Throwable("Input string could not be interpreted as a number");
  ```

- getCause() - You might catch an exception in your code and, because of it, throw another, different exception. However, that exception you caught was the root cause of the new exception you threw, and you might want to remember it. You can pass the first exception to the Throwable constructor, in which case it will be stored in the Throwable as the "cause". So, for example:

  ```
  int n = 0;
  try {
    n = Integer.parseInt(s);
  } catch(NumberFormatException e)
  {
    throw new Throwable("Input string could not be interpreted as a number",e);
  }
  ```

  Whoever catches the Throwable we've thrown can use the getCause method to get the original exception that caused use to throw our Throwable.

Using just the different messages, we can easily produce meaningful error messages ... at least for some errors.

Ex1.java

```
public class Ex1
{
  public static int getSF(String[] av) throws Throwable
  {
    int[] B = null;
    try {
      String[] A = av[0].split(",");
      B = new int[A.length];
      for(int i = 0; i < A.length; i++)
        B[i] = Integer.parseInt(A[i]);
    } catch(Throwable e) {
      throw new Throwable("Couldn't convert input to array of ints");
    }
    return SpecialFunc.compute(B);
  }

  public static void main(String[] args)
  {
    try {
      System.out.println(getSF(args));
    }
    catch(Throwable e)
    {
      System.out.println("Error in program Ex1: " + e.getMessage());
    }
  }
}
```

SpecialFunc.java

```
public class SpecialFunc
{
  public static int compute(int[] B) throws Throwable
  {
    double sum = 0;
    for(int i = 0; i < B.length; i++)
    {
      if (B[i] == 0) throw new Throwable("Zero value in array B!");
      sum += 1.0/B[i];
    }
    if (sum < 0) throw new Throwable("Sum is negative, cannot take sqrt!");
    double ssum = Math.sqrt(sum/B.length);
    int res = (int)(100.0*ssum);
    return res;
  }
}
```

```
~/$ java Ex1
Error in program Ex1: Couldn't convert input to array of ints
~/$ java Ex1 2,three,4
Error in program Ex1: Couldn't convert input to array of ints
~/$ java Ex1 2,0,3
Error in program Ex1: Zero value in array B!
~/$ java Ex1 -2,-1,3
Error in program Ex1: Sum is negative, cannot take sqrt!
```

Note: We could ad a verbose mode to our program in which errors would result in a stack trace being printed.

Adding to the exception hierarchy to organize better

We might decide that we need to handle exceptions generated by SpecialFunc differently than exceptions generated by other parts of the program. Perhaps main simply wants to preface each error message with whether or not it came from SpecialFunc or from Ex1? In order to do this cleanly we need to define a new exception type specifically for our SpecialFunc class. This way, SpecialFunc.compute() can throw this new type instead of a generic Throwable, which allows main to catch those exceptions separately from any others.

Ex1b.java

```
public class Ex1b
{
  public static int getSF(String[] av) throws Throwable
  {
    int[] B = null;
    try {
      String[] A = av[0].split(",");
      B = new int[A.length];
      for(int i = 0; i < A.length; i++)
        B[i] = Integer.parseInt(A[i]);
    } catch(Throwable e) {
```

SpecialFunc.java

```
public class SpecialFunc
{

  public static class ComputeException extends Exception
  {
    public ComputeException(String msg) { super(msg,null); }
  }

  public static int compute(int[] B) throws Throwable
  {
    double sum = 0;
```

```
      throw new Throwable("Couldn't convert input to array of ints");
    }
    return SpecialFunc.compute(B);
  }

  public static void main(String[] args)
  {
    try {
      System.out.println(getSF(args));
    }
    catch(SpecialFunc.ComputeException e)
    {
      System.out.println("Error in SpecialFunc: " + e.getMessage());
    }
    catch(Throwable e)
    {
      System.out.println("Error in program Ex1: " + e.getMessage());
    }
  }
}
```

```
    for(int i = 0; i < B.length; i++)
    {
      if (B[i] == 0) throw new ComputeException("Zero value in array B!");
      sum += 1.0/B[i];
    }
    if (sum < 0) throw new ComputeException("Sum is negative, cannot take sqrt!");
    double ssum = Math.sqrt(sum/B.length);
    int res = (int)(100.0*ssum);
    return res;
  }
}
```

Extending functionality:
Suppose we want to be able to specify exactly which element in the comma-separated list could not be converted? Suppose we wanted to include the index of of the integer that was zero? We need exceptions that carry extra information with them! In other words, we need to start using inheritance to extend functionality.

Ex2.java

```
public class Ex2
{

  public static class GetSFNumFormEx extends Exception
  {
    private String text;
    public GetSFNumFormEx(String msg, Throwable cause, String text)
    {
      super(msg,cause);
      this.text = text;
    }
    String getText() { return text; }
  }

  public static int getSF(String[] av) throws Throwable
  {
    String[] A = av[0].split(",");
    if (A.length == 0) throw new Throwable();
    int[] B = new int[A.length];
    for(int i = 0; i < A.length; i++)
      try {
        if (A[i].equals(""))
          throw new GetSFNumFormEx("Not number",null,A[i]);
        B[i] = Integer.parseInt(A[i]);
      }
      catch(NumberFormatException e)
      {
        throw new GetSFNumFormEx("Not number",e,A[i]);
      }
    return SpecialFunc.compute(B);
  }

  public static void main(String[] args)
  {
    try {
      System.out.println(getSF(args));
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println("Error! An argument is required.");
    }
    catch (GetSFNumFormEx e)
    {
      System.out.println("Error! Could not convert input '" +
                         e.getText() + "' to an integer!");
    }
    catch(SpecialFunc.ElementException e)
    {
      System.out.println("Error '" + e.getMessage() +
                         "' with input at index " + e.getIndex());
    }
    catch(SpecialFunc.ComputeException e)
    {
      System.out.println(e.getMessage());
    }
    catch(Throwable e)
    {
      System.out.println("Bad stuff happened!");
    }
  }
}
```

SpecialFunc.java

```
public class SpecialFunc
{

  public static class ComputeException extends Exception
  {
    public ComputeException(String msg) { super(msg,null); }
  }

  public static class ElementException extends ComputeException
  {
    private int index;
    public ElementException(String msg, int i) { super(msg); index = i; }
    public int getIndex() { return index; }
  }

  public static int compute(int[] B) throws ComputeException
  {
    double sum = 0;
    for(int i = 0; i < B.length; i++)
    {
      if (B[i] == 0) throw new ElementException("Zero value",i);
      sum += 1.0/B[i];
    }
    if (sum < 0) throw new ComputeException("Negative sum, can't take root.");
    double ssum = Math.sqrt(sum/B.length);
    int res = (int)(100.0*ssum);
    return res;
  }
}
```

```
~/$ java Ex2
Error! An argument is required.
~/$ java Ex2 2,,3
Error! Could not convert input '' to an integer!
~/$ java Ex2 2,six,3
```

```
Error! Could not convert input 'six' to an integer!
~/$ java Ex2 2,0,3
Error 'Zero value' with input value 1
~/$ java Ex2 2,0,3
Error 'Zero value' with input at index 1
~/$ java Ex2 2,-1,3
Negative sum, can't take root.
~/$ java Ex2 ,
Bad stuff happened!
```