Throwable Objects

We have said that when some sort of abnormal event occurs, like an attempt to divide by 0, Java creates an object and throws it, and this thrown object is then caught and handled somewhere. We now explain what a throwable object is.

Below is class Throwable, which is defined in package java.lang. Any thrown object is an instance of this class (or one of its subclasses). For example, if a division by zero occurs, an instance of class ArithmeticException is created and thrown; this class is a subclass of RuntimeException, which is a subclass of Exception, which is a subclass of Throwable. Class Error is also a subclass of Throwable.

Some classes in the Throwable Hierarchy

Throwable
  Error
  Exception
    RuntimeException
      ArithmeticException

Class Throwable

Let's take a look at class Throwable. We'll show some of its fields and methods in object a0 —we don't show all of them for lack of space.

```
public class Throwable implements ... {
   private transient object backtrace;
   private String detailMessage;

   /** Constructor: instance with no detail message. */
   public Throwable() { ... }

   /** Constructor: instance with detail message m. */
   public Throwable(String m) { ... }

   /** = the detail message (null if none) */
   public String getMessage() { ... }

   /** = localized message. If not overridden, same as getMessage() */
   public String getLocalizedMessage() { ... }

   /** = short description of this instance */
   public String toString() { ... }
   ...

   /** Store the call stack in field backtrace */
   public native Throwable fillInStackTrace();
}
```

Field backtrace automatically contains the call stack at the point where the abnormal event occurred --that is, it contains information about the methods that were called but have not yet terminated. And field detailMessage can contain a description of the error. For example, for the abnormal event division by 0, this field contains "/ by zero".

Every throwable object has these two pieces of information.

There are two constructors, one of which allows the caller to give the detail message. Getter method getMessage returns the detail message, and method getLocalizedMessage can be overridden to return a message that is particular to a subclass. The usual method toString is there as well.

There are several methods for printing the call stack in various places. And finally, there is a method to store the current call stack in field backtrace. This method is useful when you create and throw an object yourself.

Classes Exception, RuntimeException, and ArithmeticException

On the next page is class Exception:

Throwable Objects

```
/** An instance indicates a condition that a reasonable application might want to catch. */
    public class Exception extends Throwable {
      /** Constructor: an Exception with no detail message. */
      public Exception() {
            super();
      }
      /** Constructor: an Exception with detail message s */
      public Exception(String s) {
         super(s);
      }
    }
```

Hey, it doesn't contain much, because superclass Throwable contains all the necessary information. All we have are two constructors, one of which allows the creator of an Exception to specify the value for field detailMessage.

In the same way, class RuntimeException contains two constructors.

Concluding remarks

You now know what just about any throwable object contains. Basically, it can contain the call stack and a message that is particular to the object. This information can be useful when catching and handling the object. With this knowledge, we can now go on to see how to throw and catch an object.