MODULE 6q - Exceptions

THE TRY-CATCH CONSTRUCT

Three different exceptions are referred to in the program below. They are the ArrayIndexOutOfBoundsException which is built-into Java and two others, BadLuckException and BadZeroException, which are declared (as classes) in the program.

```
public class TryCatch
 { private static int[] a = {12,22,0,19,13,29};
   private static int p=0;

   public static void main(String[] args)
    { while (true)
       { try
          { System.out.printf("%d%n", read());
          }
         catch(ArrayIndexOutOfBoundsException e)               // catch 1
          { System.out.printf("Finished%n");
            break;
          }
         catch(BadLuckException e)                             // catch 2
          { System.out.printf("---> %s%n", e.getMessage());
            continue;
          }
       }
    }

   private static int read() throws ArrayIndexOutOfBoundsException,
                                    BadLuckException
    { int n=0;
      try
       { n = a[p++];
         if (n==0)
            throw new BadZeroException();
         if (n==13)
            throw new BadLuckException();
       }
      catch(BadZeroException e)                                // catch 3
       { System.out.printf("---> ");
       }
      return n;
    }
 }

class BadLuckException extends Exception
 { public String getMessage()
    { return "unlucky thirteen";
    }
 }

class BadZeroException extends Exception
 { public String getMessage()
```

```
    { return "Zero";
    }
 }

// This yields:
//
// 12
// 22
// ---> 0
// 19
// ---> unlucky thirteen
// 29
// Finished
```

TRY IT OUT

Key the program in, compile it and run it. It ought to give the results which appear as comments at the end of the program.

Next see what happens if, in turn, the catch 1 clause is omitted, the catch 2 clause is omitted and the catch 3 clause heading is changed to

```
         catch(BadLuckException e)
```

Note also what happens if the throws clause is, in turn, changed to

```
         throws ArrayIndexOutOfBoundsException
```
and to
```
         throws BadLuckException
```

THE TRY-CATCH-FINALLY CONSTRUCT

An adapted version of the above program is given here. It refers to the same three exceptions but includes a try-catch-finally construct.

```
public class TryFinally
 { private static int[] a = {12,22,0,19,13,29};
   private static int p=0;
   private static int totup=0;

   public static void main(String[] args)
    { while (true)
       { try
          { System.out.printf("%d%n", read());
          }
         catch(ArrayIndexOutOfBoundsException e)
          { System.out.printf("Finished%n");
            break;
          }
         catch(BadLuckException e)
```

```
          { System.out.printf("---> %s%n", e.getMessage());
            continue;
          }
       }
      System.out.printf("Read entered %d times%n", totup);
    }

   private static int read() throws ArrayIndexOutOfBoundsException,
                                    BadLuckException
    { int n=0;
      try
       { n = a[p++];
         if (n==0)
            throw new BadZeroException();
         if (n==13)
            throw new BadLuckException();
       }
      catch(BadZeroException e)
       { System.out.printf("---> ");
       }
      finally
       { totup++;
       }
      return n;
    }
 }

class BadLuckException extends Exception
 { public String getMessage()
    { return "unlucky thirteen";
    }
 }

class BadZeroException extends Exception
 { public String getMessage()
    { return "Zero";
    }
 }

// This yields:
//
// 12
// 22
// ---> 0
// 19
// ---> unlucky thirteen
// 29
// Finished
// Read entered 7 times
```

TRY IT OUT

Edit the previous program so that is appears as this new version,
compile it and run it. It ought to give the results shown.

TRY THIS JIFFY PROGRAM TOO

```
public class HelloC
 { public static void main(String[] args) throws InterruptedException
    { System.out.printf("Hello%n");
      Thread.sleep(3000L);
      System.out.printf("World%n");
    }
 }
```

JAVA COLLECTIONS

Java has many facilities for processing collections of items. A simple requirement is an array that will expand and contract as items are are added or removed and the Java class ArrayList in the java.util package achieves this. Class ArrayList implements the interface Collection. To ensure full polymorphism the so-called generic <Object> is specified.

In the following example the ArrayList notes is set up and six items are added to it. One is then removed. The iterator() method in an ArrayList object returns an Iterator object which is assigned to the Iterator variable it. An Iterator enables every item in the collection to be processed. Look up ArrayList and Iterator and try this program out.

```
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListIntro
 { public static void main(String[] args)
    { ArrayList <Object> notes = new ArrayList<Object>();

      notes.add("Watch");
      notes.add("Clock");
      notes.add("Sundial");
      notes.add(new Integer(42));    // Primitive types must be wrapped
      notes.add(new Float(3.142f));  // up to turn them into objects.
      notes.add("Clock");
      notes.remove("Sundial");

      System.out.printf("The number of notes is %d%n", notes.size());
      System.out.printf("The selected entry is %s%n", notes.get(3));

      Iterator it = notes.iterator();

      while (it.hasNext())
         System.out.printf("%s%n", it.next());
    }
 }

// This yields:
//
// The number of notes is 5
// The selected entry is 3.142
// Watch
// Clock
// 42
// 3.142
// Clock
```

Class HashSet also implements the interface Collection. As its name implies a HashSet is a set so repeated elements are not permitted and order is not defined. Again the generic <Object> is specified. Note that an attempt to access element three, say, fails. Look up HashSet and try this program out.

```
import java.util.HashSet;
import java.util.Iterator;

public class HashSetIntro
 { public static void main(String[] args)
    { HashSet <Object> notes = new HashSet<Object>();

      notes.add("Watch");
      notes.add("Clock");
      notes.add("Sundial");
      notes.add(new Integer(42));
      notes.add(new Float(3.142f));
      notes.add("Clock");
      notes.remove("Sundial");

      System.out.printf("The number of notes is %d%n", notes.size());
      //  System.out.printf(notes.get(3));    ... won't compile

      Iterator it = notes.iterator();

      while (it.hasNext())
         System.out.printf("%s%n", it.next());
    }
 }

// This yields:
//
// The number of notes is 4
// Clock
// 3.142
// 42
// Watch
```

THE HashMap CLASS

Class HashMap does not implement the Collection interface but it can be thought of as a close relation. A HashMap is rather like an ArrayList but instead of the elements being indexed by integers they are indexed by keys. Sometimes the term ‘association list’ is used to describe a HashMap.

The generic pair <String, Object> is specified now. Here String

indicates that each key is always going to be of type String but
Object indicates that the items indexed may be of any type.

Notice that the put() method is used instead of an add() method
to incorporate a new entry and the put() method takes two arguments.
The first is the key and the second is the value. Each is of Object
type but in the following example type String is always used for the
key.

There is no iterator() method in a HashMap object but the keySet()
method returns an object which is a set of keys and this DOES have
an iterator() method which is duly assigned to the variable it.

Look up HashMap and try this program out.

```
import java.util.HashMap;
import java.util.Iterator;

public class HashMapIntro
 { public static void main(String[] args)
    { HashMap <String, Object> notes = new HashMap<String, Object>();

      notes.put("Please", "Watch");
      notes.put("Come", "Clock");
      notes.put("To", "Sundial");
      notes.put("Our", new Integer(42));
      notes.put("Christmas", new Float(3.142f));
      notes.put("Party", "Clock");
      notes.remove("To");

      System.out.printf("The number of notes is %d%n", notes.size());
      System.out.printf("The selected entry is %s%n", notes.get("Come"));

      Iterator it = notes.keySet().iterator();

      while (it.hasNext())
         System.out.printf("%s%n", notes.get(it.next()));
    }
 }

// This yields:
//
// The number of notes is 5
// The selected entry is Clock
// 42
// Clock
// Clock
// 3.142
// Watch
```