ITEC 1630

Yves Lespérance

Lecture Notes

Week 7 — Exception Handling

Readings: Horstmann Ch. 15

Exception Handling

Exception handling is a mechanism for making programs more robust, i.e. have then continue executing when errors, failures, or exceptional conditions occur, rather than crash.

This is especially important for embedded software such as a system controlling an airplane. But most applications should tolerate failures such as bad user input, e.g.

```
Scanner in = new Scanner(System.in);
System.out.println("Enter student number & name separated by ;");
String line = in.nextLine();
Scanner inl = new Scanner(line).useDelimiter(";");
String nt = inl.next();
   // may throw NoSuchElementException
int sno = Integer.parseInt(nt);
   // may throw NumberFormatException
String sname = inl.next();
   // may throw NoSuchElementException
```

You can catch and handle such exceptions using the `try-catch` construct, e.g.

```
import java.util.Scanner;
import java.util.NoSuchElementException;
public class EHegA
{  public static void main(String[] args)
   {  Scanner in = new Scanner(System.in);
      System.out.println("Enter student number & name separated by ;");
      String line = in.nextLine();
      Scanner inl = new Scanner(line).useDelimiter(";");
      try
      {  String nt = inl.next();
         int sno = Integer.parseInt(nt);
         String sname = inl.next();
         System.out.println("name= " + sname);
         System.out.println("number= " + sno);
      }
      catch(NumberFormatException e)
      {  // handler for NumberFormatException
         System.out.println("Invalid number");
      }
      catch(NoSuchElementException e)
      {  // handler for NoSuchElementException
         System.out.println("Not enough arguments");
      }
   }
}
```

```
zebra 348 % java EHegA
Enter student number & name separated by ;
203045;John Doe
name= John Doe
number= 203045
zebra 349 % java EHegA
Enter student number & name separated by ;
203045
Not enough arguments
zebra 350 % java EHegA
Enter student number & name separated by ;
203045John Doe
Invalid number
zebra 351 % java EHegA
Enter student number & name separated by ;
203045;
Not enough arguments
zebra 352 %
```

When an exception is thrown, e.g. `Integer.parseInt` throws `NumberFormatException`, the execution of the `try` block is aborted and the corresponding `catch` handler block is executed.

If there is no matching `catch` clause, the exception propagates out of the `try` block and may be caught by an outside `try` block. If the exception is never caught, the program's execution will terminate abnormally.

You can also throw exceptions yourself using the `throw` statement, e.g.

```
throw new RunTimeException("Invalid CallZone");
```

The `finally` clause is used to indicate that some statements must be executed even when an exception is thrown, usually to do some cleaning up operations or ensure that an object's state is consistent. E.g.

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
import java.util.NoSuchElementException;
```

```
public class EHegB
{  public static void main(String[] args) throws IOException
   {  FileReader r = new FileReader("sinput.txt");
         // may throw FileNotFoundException
      Scanner in = new Scanner(r);
      try
      {  for(String inp = in.nextLine(); in.hasNextLine();
                 inp = in.nextLine())
         {  Scanner inl = new Scanner(inp).useDelimiter(";");
            String nt = inl.next();
            int sno = Integer.parseInt(nt);
            String sname = inl.next();
            System.out.println("name= " + sname);
            System.out.println("number= " + sno);
         }
      }
      catch(NumberFormatException e)
      {  System.out.println("Invalid number");
      }
      catch(NoSuchElementException e)
      {  System.out.println("Not enough arguments");
      }
      finally
      {  r.close(); // may throw IOException
      }
   }
}
```

```
zebra 356 % more sinput.txt
203045;John Doe
234578 Mary Wong
217690;Paul Smith
zebra 357 % java EHegB
name= John Doe
number= 203045
Invalid number
zebra 358 %
```

The `finally` block is *always* executed, whether an exception is thrown or not, and if one is, whether it is caught or not.

`try-catch` Control Flow

```
try
{  statements_try
}
catch(C1 th)
{  statements_C1
}
catch(C2 th)
{  statements_C2
}
catch(C3 th)
{  statements_C3
}
...
finally
{  statements_finally
}
```

In normal control flow, when nothing is thrown in `{ statements_try }`:
execute all of `{ statements_try }`
and then all of `{ statements_finally }`.

If a `Throwable th` is thrown somewhere in
`{ statements_try }`:
the block is immediately exited, and then we execute

```
if(th instanceof C1)
{  statements_C1
}
else if(th instanceof C2)
{  statements_C2
}
else if(th instanceof C3)
{  statements_C3
}
...
```

followed by `{ statements_finally }`.

Classes `C1, C2, ...` must be subclasses of `Throwable`.

Make sure you import `C1, C2, ...`.

Order the catches as you would order if's.

Can intentionally create and throw exceptions.

The `finally` block is *always* executed:

- after a normal try,
- after a normal catch,
- after a throw in a catch handler,
- after an uncaught throwable.

Often better to put `finally` with a separate `try`.

Checked and Unchecked Exceptions

Exceptions and errors are organized into a hierarchy whose root is the `Throwable` class (see Horstmann p. 554).

All exceptions except instances of `RunTimeException` are *checked* (by the compiler).

When you call a method that may throw a checked exception, your program must say what it will do if the checked exception is thrown, either catch it or declare that it may throw it. E.g. `IOException` in `EHegB`, or the following

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
import java.util.NoSuchElementException;
```

```
public class EHegC
{  public static void main(String[] args)
   {  try
      {  FileReader r = new FileReader("sinput.txt"); // may throw FileNotFoundExce
         Scanner in = new Scanner(r);
         try
         {  for(String inp = in.nextLine(); in.hasNextLine();
                   inp = in.nextLine())
            {  Scanner inl = new Scanner(inp).useDelimiter(";");
               String nt = inl.next();
               int sno = Integer.parseInt(nt);
               String sname = inl.next();
               System.out.println("name= " + sname);
               System.out.println("number= " + sno);
            }
         }
         catch(NumberFormatException e)
         {  System.out.println("Invalid number");  }
         catch(NoSuchElementException e)
         {  System.out.println("Not enough arguments");  }
         finally
         {  r.close(); // may throw IOException  }
      }
      catch(IOException e)
      {  e.printStackTrace();
      }
   }
}
```

Checked exceptions are due to external circumstances that programmer cannot control, e.g. `FileNotFoundException`; they must either be caught or a declaration that the exception can be thrown must appear.

Unchecked exceptions are usually due to preventable errors by the programmer, e.g. `NoSuchElementException` thrown by `next()` of `Scanner`; programmer should have called `hasNext()` to check first. It is better to fix this problem so that the exception is never thrown.

To set the message attribute of an exception, use the 1 argument constructor, e.g.

`new RunTimeException("too few arguments")`.

To retrieve the message attribute of an exception:

`e.getMessage()`.

To print a stack trace:

`e.printStackTrace()`.

E.g. catching an exception that we threw:

```
import java.util.Scanner;
public class EHegD
{  public static void main(String[] args)
    {  boolean inputDone = false;
       String prog = null;
       int no = 0;
       String term = null;
       Scanner in = new Scanner(System.in);
       while(!inputDone)
       {  System.out.println("Enter course, e.g. ITEC 1630 W");
          try
          {  String input = in.nextLine();
             int sep1 = input.indexOf(" ");
             int sep2 = input.indexOf(" ",sep1+1);
             prog = input.substring(0,sep1);
             no = Integer.parseInt(
                input.substring(sep1+1,sep2));
             term = input.substring(sep2+1);
             if(!term.equals("F") && !term.equals("W"))
             {  throw new IllegalArgumentException("Incorrect term code");
             }
             inputDone = true;
          }
```

```
          catch(IndexOutOfBoundsException e)
          {  System.out.println("Missing field in input");
          }
          catch(NumberFormatException e)
          {  System.out.println("Incorrect format for course number");
          }
          catch(IllegalArgumentException e)
          {  System.out.println(e.getMessage());
          }
       }
       System.out.println(prog + ":" + no + ":" + term);
    }
}
```

```
zebra 313 % java EHegD
Enter course, e.g. ITEC 1630 W
ITEC 1630
Missing field in input
Enter course, e.g. ITEC 1630 W
ITEC W
Missing field in input
Enter course, e.g. ITEC 1630 W
ITEC 1630W
Missing field in input
Enter course, e.g. ITEC 1630 W
ITEC 163a W
Incorrect format for course number
Enter course, e.g. ITEC 1630 W
ITEC 1630 XX
Incorrect term code
Enter course, e.g. ITEC 1630 W
ITEC 1630 W
ITEC:1630:W
```

One can use `throw` and `catch` to exit from several nested loops, e.g.

```
import java.util.Scanner;
public class EHegE
{  public static void main(String[] args)
   {  try
      {  while(true)
         {  boolean inputDone = false;
            String prog = null;
            int no = 0;
            String term = null;
            Scanner in = new Scanner(System.in);
            while(!inputDone)
            {  System.out.println("Enter course, e.g. ITEC 1630 W,"
                  + " blank line to exit");
               try
               {  String input = in.nextLine();
                  if(input.equals(""))
                  {  throw new Throwable();
                  }
                  int sep1 = input.indexOf(" ");
                  int sep2 = input.indexOf(" ",sep1+1);
                  prog = input.substring(0,sep1);
                  no = Integer.parseInt(
                     input.substring(sep1+1,sep2));
                  term = input.substring(sep2+1);
               if(!term.equals("F") && !term.equals("W"))
               {  throw new IllegalArgumentException("Incorrect term code");
               }
               inputDone = true;
            }
            catch(IndexOutOfBoundsException e)
            {  System.out.println("Missing field in input");
            }
            catch(NumberFormatException e)
            {  System.out.println("Incorrect format for course number");
            }
            catch(IllegalArgumentException e)
            {  System.out.println(e.getMessage());
            }
         }
         System.out.println(prog + ":" + no + ":" + term);
      }
   }
   catch(Throwable t)
   {
   }
 }
}
```

```
zebra 319 % java EHegE
Enter course, e.g. ITEC 1630 F, blank line to exit
ITEC 1202
Missing field in input
Enter course, e.g. ITEC 1630 F, blank line to exit
ITEC 1630 F
ITEC:1630:F
Enter course, e.g. ITEC 1630 F, blank line to exit
ITEC 1030 W
ITEC:1030:W
Enter course, e.g. ITEC 1630 F, blank line to exit

zebra 320 %
```

But this is not recommended. Same e.g. without using `throw` for deep exit:

```
import java.util.Scanner;
public class EHegF
{  public static void main(String[] args)
   {  while(true)
      {  boolean inputDone = false;
         String input = null;
         String prog = null;
         int no = 0;
         String term = null;
         Scanner in = new Scanner(System.in);
         while(!inputDone)
         {  System.out.println("Enter course, e.g. ITEC 1630 W,"
               + " blank line to exit");
            try
            {  input = in.nextLine();
               if(!input.equals(""))
               {  int sep1 = input.indexOf(" ");
                  int sep2 = input.indexOf(" ",sep1+1);
                  prog = input.substring(0,sep1);
                  no = Integer.parseInt(
                     input.substring(sep1+1,sep2));
                  term = input.substring(sep2+1);
```

```
                  if(!term.equals("F") && !term.equals("W"))
                  {  throw new IllegalArgumentException("Incorrect term code");
                  }
               }
               inputDone = true;
            }
            catch(IndexOutOfBoundsException e)
            {  System.out.println("Missing field in input");
            }
            catch(NumberFormatException e)
            {  System.out.println("Incorrect format for course number");
            }
            catch(IllegalArgumentException e)
            {  System.out.println(e.getMessage());
            }
         }
         if(input.equals(""))
         {  break;
         }
         System.out.println(prog + ":" + no + ":" + term);
      }
   }
}
```

Defining Your Own Exception Classes

Sometimes standard exception classes are not very appropriate to your application. Then you can design your own. E.g.

```
public class IllegalTermCodeException extends IllegalArgumentException
{  public IllegalTermCodeException()
   {}

   public IllegalTermCodeException(String message)
   {super(message);
   }
}
...

if(!term.equals("F") && !term.equals("W"))
{  throw new IllegalTermCodeException(term + "is not a legal term code");
}
```

Generally, one defines 2 constructors, one that takes no arguments and one that takes a message string as argument.

It is best to use exceptions that are as specific as possible.