COSC 1020

Yves Lespérance

Lecture Notes
Week 10 — Exception Handling

Recommended Readings:
Horstmann: Ch. 14
Lewis & Loftus: Ch. 8 Sec. 0

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.

```
IO.println("Enter student number & name separated by ;'
String inp = IOreadLine();
StringTokenizer t = new StringTokenizer(inp,";");
String nt = t.nextToken();
   // may throw NoSuchElementException
int sno = Integer.parseInt(nt);
   // may throw NumberFormatException
String sname = t.nextToken();
   // may throw NoSuchElementException
```

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

```
import type.lang.*;
import java.util.*;
public class EHegA
{  public static void main(String[] args)
   {  IO.println("Enter student number & name separated by ;");
      String inp = IO.readLine();
      StringTokenizer t = new StringTokenizer(inp,";");
      try
      {  String nt = t.nextToken();
         int sno = Integer.parseInt(nt);
         String sname = t.nextToken();
         IO.println("name= " + sname);
         IO.println("number= " + sno);
      }
      catch(NumberFormatException e)
      {  // handler for NumberFormatException
         IO.println("Invalid number");
      }
      catch(NoSuchElementException e)
      {  // handler for NoSuchElementException
         IO.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 executer.

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 stock symbol");
```


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 type.lang.*;
import java.util.*;
public class EHegB
{  public static void main(String[] args)
   {  UniReader r = new UniReader("sinput.txt");
      try
      {  for(String inp = r.readLine(); !r.eof();
                inp = r.readLine())
         {  StringTokenizer t = new StringTokenizer(inp,";");
            int sno = Integer.parseInt(t.nextToken());
            String sname = t.nextToken();
            IO.println("name= " + sname);
            IO.println("number= " + sno);
         }
      }
      catch(NumberFormatException e)
      {  IO.println("Invalid number");
      }
      catch(NoSuchElementException e)
      {  IO.println("Not enough arguments");
      }
      finally
      {  r.close();
      }
   }
}
```



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

followed by `{ statements_finally }`.

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

Make sure you import the packages of `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.

Checked and Unchecked Exceptions

Exceptions and errors are organized into a hierarchy whose root is the `Throwable` class (see Horstmann p. 562).

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.


```
import type.lang.*;
import java.io.*;
import java.util.*;
public class EHegC1
{  public static void main(String[] args)
   {  BufferedReader r = new BufferedReader(
         new InputStreamReader(System.in));
      try
  {  IO.println("Enter a line");
         String line = r.readLine();
         IO.println(line);
      }
      catch(IOException e)
      {  IO.println("IO problem");
      }
   }
}
```

or

```
import type.lang.*;
import java.io.*;
import java.util.*;
public class EHegC2
{  public static void main(String[] args) throws IOException
   {  BufferedReader r = new BufferedReader(
         new InputStreamReader(System.in));
      IO.println("Enter a line");
      String line = r.readLine();
      IO.println(line);
   }
}
```

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 thrown by `SE.require`:

```
import type.lang.*;
import type.lib.*;
import java.util.*;
public class EHeg
{  public static void main(String[] args)
    {  boolean inputDone = false;
       String prog = null;
       int no = 0;
       String term = null;
       while(!inputDone)
       {  IO.println("Enter course, e.g. COSC 1020 F");
          try
          {  String input = IO.readLine();
             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);
             SE.require(term.equals("F") || term.equals("W"));
             inputDone = true;
          }
          catch(IndexOutOfBoundsException e)
          {  IO.println("Missing field in input");
          }
          catch(NumberFormatException e)
          {  IO.println("Incorrect format for course number");
          }
          catch(SEpreconditionException e)
          {  IO.println("Incorrect term code");
          }
       }
       IO.println(prog + ":" + no + ":" + term);
    }
}
```



```
zebra 313 % java EHeg
Enter course, e.g. COSC 1020 F
COSC 1020
Missing field in input
Enter course, e.g. COSC 1020 F
COSC F
Missing field in input
Enter course, e.g. COSC 1020 F
COSC 1020F
Missing field in input
Enter course, e.g. COSC 1020 F
COSC 102a F
Incorrect format for course number
Enter course, e.g. COSC 1020 F
COSC 1020 XX
Incorrect term code
Enter course, e.g. COSC 1020 F
COSC 1020 F
COSC:1020:F
```


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

```
import type.lang.*;
import type.lib.*;
import java.util.*;
public class EHeg2
{  public static void main(String[] args)
   {  try
      {  while(true)
         {  boolean inputDone = false;
            String prog = null;
            int no = 0;
            String term = null;
            while(!inputDone)
            {  IO.println("Enter course, e.g. COSC 1020 F,"
                  + " blank line to exit");
               try
               {  String input = IO.readLine();
                  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);
                  SE.require(term.equals("F") || term.equals("W")
                  inputDone = true;
               }
```

```
            catch(IndexOutOfBoundsException e)
            {  IO.println("Missing field in input");
            }
            catch(NumberFormatException e)
            {  IO.println("Incorrect format for course number"
            }
            catch(SEpreconditionException e)
            {  IO.println("Incorrect term code");
            }
         }
         IO.println(prog + ":" + no + ":" + term);
      }
   }
   catch(Throwable t)
   {
   }
  }
}
```

```
zebra 319 % java EHeg2
Enter course, e.g. COSC 1020 F, blank line to exit
COSC 1202
Missing field in input
Enter course, e.g. COSC 1020 F, blank line to exit
COSC 1020 F
COSC:1020:F
Enter course, e.g. COSC 1020 F, blank line to exit
COSC 1030 W
COSC:1030:W
Enter course, e.g. COSC 1020 F, blank line to exit

zebra 320 %
```


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

```
import type.lang.*;
import type.lib.*;
import java.util.*;
public class EHeg3
{  public static void main(String[] args)
   {  while(true)
      {  boolean inputDone = false;
         String input = null;
         String prog = null;
         int no = 0;
         String term = null;
         while(!inputDone)
         {  IO.println("Enter course, e.g. COSC 1020 F,"
               + " blank line to exit");
            try
            {  input = IO.readLine();
               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);
                  SE.require(term.equals("F") || term.equals("W")
               }
               inputDone = true;
            }
```



```
            catch(IndexOutOfBoundsException e)
            {  IO.println("Missing field in input");
            }
            catch(NumberFormatException e)
            {  IO.println("Incorrect format for course number");
            }
            catch(SEpreconditionException e)
            {  IO.println("Incorrect term code");
            }
         }
         if(input.equals(""))
         {  break;
         }
         IO.println(prog + ":" + no + ":" + term);
      }
   }
}
```