Exception Handling

- Reading: Savitch, Chapter 8
- Reference: Big Java, Horstmann, Chapter 14

Case Study

- Discuss the example

  Coin.java

  Purse.java

  PurseTest.java

- This example was also discussed in earlier lectures on polymorphism

Case Study

- Coin.java
  - A coin with a monetary value and the name of the coin
  - A method read() to read a file with coin descriptions
- Purse.java
  - A purse computes the total of a collection of coins.
  - A method read() which uses Coin's read() to read a file with coin descriptions and add these coins to the purse.
- PurseTest.java
  - prompts the user to enter a file name with coin values. A purse object is filled with the coins specified in the file. In the case of an exception, the user can choose another file.

A Complete Example

- Program: PurseTest
  - Asks user for the name of a file
  - Reads coin descriptions from file
  - Adds coins to purse
  - Prints total
- What can go wrong?
  - File might not exist
  - File might have data in wrong format
- Who can detect the faults?
  - `main` method of `PurseTest` interacts with user
  - `main` method can report errors
  - Other methods pass exceptions to caller

The `read` Method of the `Coin` Class

Distinguishes between *expected* and *unexpected* end of file

```
public boolean read(BufferedReader in) throws IOException
{
   String input = in.readLine();
   if (input == null) // normal end of file
         return false;
   value = Double.parseDouble(input);
    // may throw unchecked NumberFormatException
   name = in.readLine();
   if (name == null) // unexpected end of file
    throw new EOFException("Coin name expected");
   return true;
  }
```

The `read` Method of the `Purse` Class

- Unconcerned with exceptions
- Just passes them to caller

```
public void read(BufferedReader in) throws IOException
{
        boolean done = false;
        while (!done)
        {
                Coin c = new Coin();
                if (c.read(in)) add(c);
                else done = true;
        }
}
```

The read Method of the Purse Class

- `finally` clause closes files if exception happens

```
public void readFile(String filename) throws
                                        IOException
{
    BufferedReader in = null;
    try
    {
        in = new BufferedReader(
                new FileReader(filename));
        read(in);
    }
    finally
    {   if (in != null)
        in.close();
    }
}
```

User Interaction in Main

- If an exception occurs, user can specify another file name

```
boolean done = false;
String filename =
JOptionPane.showInputDialog("Enter file name");
while (!done)
{
      try
      {
            Purse myPurse = new Purse();
            myPurse.readFile(filename);
            System.out.println("total=" +
                                    myPurse.getTotal());
            done =true;
      }
```

```
 catch (IOException exception)
{
      System.out.println("Input/output error " + exception);
}
catch (NumberFormatException exception)
{
     exception.printStackTrace();   // error in file  format
}
if (!done)
{
   Filename = JOptionPane.showInputDialog("Try another file:");
   if (filename == null)
        done =true;
}
}
```

Scenario

1. `PurseTest.main` Calls `Purse.readFile`
2. `Purse.readFile` Calls `Purse.read`
3. `Purse.read` Calls `Coin.read`
4. `Coin.read` throws an `EOFException`
5. `Coin.read` has no handler for the exception and terminates immediately.
6. `Purse.read` has no handler for the exception and terminates immediately

Scenario

7. `Purse.readFile` has no handler for the exception and terminates immediately after executing the finally clause and closing the file.
8. `PurseTest.main` has a handler for an `IOException`, a superclass of `EOFException`. That handler prints a message to the user. Afterwards, the user is given another chance to enter a file name. Note that the statement printing the purse total has been skipped.

File PurseTest.java

```
0   // Reference: Horstmann, Chapter 14
1   import javax.swing.JOptionPane;
2   import java.io.IOException;
3   /**
4    Prompts the user to enter a file name with
5    coin values. A purse object is filled with the
6    coins specified in the file. In case of an
7    exception the user can choose another file.
8   */
9   public class PurseTest
10  {
11    public static void main(String[] args)
12    {
13      boolean done = false;
14      String filename  =
15                JOptionPane.showInputDialog(
16                "Enter file name");
```

```
18          while (!done)
19          {   try
20              {
21                  Purse myPurse = new Purse();
22                  myPurse.readFile(filename);
23                  System.out.println("total=" +
14                                      myPurse.getTotal());
25                  done = true;
26              }
27              catch (IOException exception)
28              {
29                  System.out.println("Input/output erro
30                                          + exception);
31              }
32              catch (NumberFormatException exception)
33              {
34                  exception.printStackTrace();
35              }
```

```
36      if (!done)
37      {
38        filename =
39          JOptionPane.showInputDialog(
39          "Try another file:");
40        if (filename == null) done = true;
41      }
42    }
43    System.exit(0);
44  }
45 }
```

File Purse.java

```
0 // Reference: Horstmann, Chapter 14
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4   /**
5     A purse computes the total of a
6     collection of coins.
7   */
8 public class Purse
9 {
10    /**
11       Constructs an empty purse.
12    */
13    public Purse()
14    {
15       total = 0;
16    }
```

```
18     /**
19        Read a file with coin descriptions and x
20        adds the coins to the purse.
21        Parameter filename:  the name of the file
22     */
23     public void readFile(String filename)
24        throws IOException
25     {
26        BufferedReader in = null;
27        try
28        { in = new BufferedReader(new
29                   FileReader(filename));
30           read(in);
31        }
32        finally
33        { if (in != null) in.close(); }
34     }
```

```
38     /**
39        Read a file with coin descriptions
40        and adds the coins to the purse.
41        Parameter in: the buffered reader for
42       reading the input
43     */
44     public void read(BufferedReader in)
45                                 throws IOException
46    {    boolean done = false;
47         while (!done)
48         {
49            Coin c = new Coin();
50            if (c.read(in))
51               add(c);
52            else
53               done = true;
54         }
55     }
```

```
57     /**
58        Add a coin to the purse.
59        Parameter aCoin: the coin to add
60     */
61     public void add(Coin aCoin)
62     {
63        total = total + aCoin.getValue();
64     }
65
66     /**
67        Get the total value of the coins in the purse.
68        return the sum of all coin values
69     */
70     public double getTotal()
71     {
72        return total;
73     }
75     private double total;
76  }
```

File Coin.java

```
0 //Reference: Horstmann, Chapter 14
1 import java.io.BufferedReader;
2 import java.io.EOFException;
3 import java.io.IOException;
4 /**
6    A coin with a monetary value.
7 */
8 public class Coin
9 {
10  /**
11     Constructs a default coin. Use the read
12     method to fill in the value and name.
13    */
14    public Coin()
15    {
```

```
16          value = 0;
17          name = "";
18       }
19
20       /**
21          Constructs a coin. Parameter aValue: the
22          monetary value of the coin.
23          Parameter aName: the name of the coin
24       */
25       public Coin(double aValue, String aName)
26       {
27          value = aValue;
28          name = aName;
29       }
30
```

```
31 /**
32    Reads a coin value and name.
33    Parameter in: the reader
34    Return true if the data was read, false
35    if the end of the stream was reached
36    */
37 public boolean read(BufferedReader in)
38                                throws IOException
39 {  String input = in.readLine();
40    if (input == null) return false;
41    value = Double.parseDouble(input);
42    name = in.readLine();
43    if (name == null)
44       throw new EOFException("Coin name
45                                   expected");
46    return true;
47 }
```

```
49      /**
50         Gets the coin value.
51         return the value
52      */
53      public double getValue()
54      {
55         return value;
56      }
58      /**
59         Gets the coin name.
60         return the name
61      */
62      public String getName()
63      { return name;      }
64      private double value;
65      private String name;
66  }
```