# Closing Resources

- Resources that must be closed require careful handling, such as PrintWriter
- Use the try-with-resources statement:
  - If no exception occurs, out.close() is called after writeData() returns
  - If an exception occurs, out.close() is called before exception is passed to its handler

```
try (PrintWriter out = new PrintWriter(filename)) {
        writeData(out);
    } // out.close() is always called
```

# Designing Your Own Exception Types

- You can design your own exception types — subclasses of
- Exception or RuntimeException.
- Throw an InsufficientFundsException when the amount to withdraw an amount from a bank account exceeds the current balance.

- How do we create InsufficientFundsException as an unchecked exception class?
  - We can extend the IllegalArgumentException class

```
if (amount > balance){
    throw new
    InsufficientFundsException("withdrawal of " +
                                amount + " exceeds balance of " + balance);
}
```

# Designing Your Own Exception Types

- Supply two constructors for the class
  - A constructor with no arguments
  - A constructor that accepts a message string describing reason for exception
- When the exception is caught, its message string can be retrieved by using the getMessage method

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

    public InsufficientFundsException(String message)
    {
        super(message);
    }
}
```

# Self Check 11.16

- Suppose balance is 100 and amount is 300. What is the value of balance after these statements?

```
if (amount > balance)
{
 throw new IllegalArgumentException("Amount exceeds
 balance");
}
balance = balance - amount;
```

# Self Check 11.16

- Suppose balance is 100 and amount is 300. What is the value of balance after these statements?

```
if (amount > balance)
{
  throw new IllegalArgumentException("Amount exceeds balance");
}
balance = balance - amount;
```

- Answer: It is still 200. The last statement was not executed because the exception was thrown.

# Self Check 11.17

- When depositing an amount into a bank account, we don't have to worry about overdrafts—except when the amount is negative.
- Write a statement that throws an appropriate exception in that case.

# Self Check 11.17

- When depositing an amount into a bank account, we don't have to worry about overdrafts—except when the amount is negative.
- Write a statement that throws an appropriate exception in that case.

- Answer:

```
if (amount < 0)
{
throw new IllegalArgumentException("Negative amount");
}
```

# Self Check 11.18

- Consider the method

```
public static void main(String[] args){
try{
  Scanner in = new Scanner(new File("input.txt"));
  int value = in.nextInt();
  System.out.println(value);
}
catch (IOException exception){
  System.out.println("Error opening file.");
}
}
```

- Suppose the file with the given file name exists and has no contents. Trace the flow of execution.

# Self Check 11.18

- Consider the method

```
public static void main(String[] args){
try{
  Scanner in = new Scanner(new File("input.txt"));
  int value = in.nextInt();
  System.out.println(value);
}
catch (IOException exception){
  System.out.println("Error opening file.");
}
}
```

- Suppose the file with the given file name exists and has no contents. Trace the flow of execution.
- Answer: The Scanner constructor succeeds because the file exists. The nextInt method throws a NoSuchElementException. This is not an IOException. Therefore, the error is not caught. Because there is no other handler, an error message is printed and the program terminates.

# Self Check 11.19

- Why is an ArrayIndexOutOfBoundsException not a checked exception?
- Answer: Because programmers should simply check that their array index values are valid instead of trying to handle an ArrayIndexOutOfBoundsException.

# Self Check 11.21

- What is wrong with the following code, and how can you fix it?

```
public static void writeAll(String[] lines, String filename){
 PrintWriter out = new PrintWriter(filename);
 for (String line : lines){
 out.println(line.toUpperCase());
 }
 out.close();
}
```

# Self Check 11.21

- What is wrong with the following code, and how can you fix it?

```
public static void writeAll(String[] lines, String filename){
  PrintWriter out = new PrintWriter(filename);
  for (String line : lines){
  out.println(line.toUpperCase());
  }
  out.close();
}
```

- Answer: There are two mistakes.
  - The PrintWriter constructor can throw a FileNotFoundException.
    - You should supply a throws clause.
  - And if one of the array elements is null, a NullPointerException is thrown. In that case, the out.close() statement is never executed.
    - You should use a try-with-resources statement.

# Self Check 11.23

- Suppose you read bank account data from a file. Contrary to your expectation, the next input value is not of type double. You decide to implement a BadDataException.
- Which exception class should you extend?

- Answer: Because file corruption is beyond the control of the programmer, this should be a checked exception, so it would be wrong to extend RuntimeException or IllegalArgumentException. Because the error is related to input, IOException would be a good choice.

# The try/finally statement

- If cleanup other than close method is required

```
public double deposit (double amount) {
  try{
  . . .
  }
  finally{
      cleanup statements   // This code is executed whether or not an exception occurs
  }
}
```

- finally block always executes
  - If no exception occurs, finally block executes after try block
  - If exception occurs, finally block is executed and exception is propagated to its handler

# Application: Handling Input Errors

- Program algorithm
- Asks user for name of file
- File expected to contain data values
- First line of file contains total number of values Remaining lines contain the data
- Typical input file:

```
3
1.45
-2.1
0.05
```

# Case Study: A Complete Example

- What can go wrong?
  - File might not exist
  - File might have data in wrong format
- Who can detect the faults?
  - Scanner constructor will throw an exception when file does not exist
  - Methods that process input need to throw exception if they find error in data format
- What exceptions can be thrown?
  - FileNotFoundException can be thrown by Scanner constructor
  - BadDataException, a custom checked exception class for reporting wrong data format
- Who can remedy the faults that the exceptions report?
  - Only the main method of DataAnalyzer program interacts with user
    - Catches exceptions
    - Prints appropriate error messages
    - Gives user another chance to enter a correct file

# DataAnalyzer.java

```
class DataAnalyzer {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                DataSetReader reader = new DataSetReader();

                boolean done = false;
                while (!done) {
                        try {
                                System.out.println("Please enter the file name: ");
                                String filename = in.next();
                                double[] data = reader.readFile(filename);
                                double sum = 0;
                                for (double d : data) {
                                        sum = sum + d;
                                }
                                System.out.println("The sum is " + sum);
                                done = true;

                        } catch (FileNotFoundException exception) {
                                System.out.println("File not found.");
                        } catch (BadDataException exception) {
                                System.out.println("Bad data: " + exception.getMessage());
                        } catch (IOException exception) {
                                exception.printStackTrace();
                        }

                } // while ends
        } // main ends
} // class ends
```

This program reads a file containing numbers and analyzes its contents. If the file doesn't exist or contains strings that are not numbers, an error message is displayed.

# DataSetReader.java

```
/* Reads a data set from a file. The file must have the format  numberOfValues */
class DataSetReader{
        private double [] data;

        /** Reads a data set.
        @param filename the name of the file holding the data
        @return the data in the file
        */
        public double[] readFile(String filename) throws IOException {
        }

        /** Reads all data.
        @param in the scanner that scans the data
        */
        private void readData(Scanner in) throws BadDataException {
        }

        /** Reads one data value.
        @param in the scanner that scans the data
        @param i the position of the value to read */
        private void readValue(Scanner in, int i) throws BadDataException{
        }
}
```

# The readFile Method of the DataSetReader Class

- Constructs Scanner object
- Calls readData method
- Completely unconcerned with any exceptions
- If there is a problem with input file, it simply passes the exception to caller:

```
public double[] readFile(String filename) throws IOException{
    File inFile = new File(filename);
    try (Scanner in = new Scanner(inFile)){
        readData(in);
        return data;
    }
}
```

# The readData Method of the DataSetReader Class

- Reads the number of values
  - Constructs an array
  - Calls readValue for each data value:
- Checks for two potential errors: File might not start with an integer and File might have additional data after reading all values.
  - Makes no attempt to catch any exceptions.

```
private void readData(Scanner in) throws BadDataException {
        if (!in.hasNextInt()) {
                throw new BadDataException("Length expected");
        }
        int numberOfValues = in.nextInt();
        data = new double[numberOfValues];

        for (int i = 0; i < numberOfValues; i++){
                readValue(in, i);
        }

        if (in.hasNext()){
                throw new BadDataException("End of file expected");
        }
}
```

# The readValue method of the DataSetReader class

```
private void readValue(Scanner in, int i) throws BadDataException {
        if (!in.hasNextDouble()){
                throw new BadDataException("Data value expected");
        }
        data[i] = in.nextDouble();
}
```

# Error Scenario

- DataAnalyzer.main calls DataSetReader.readFile
- readFile calls readData
- readData calls readValue in a loop
  - readValue doesn't find expected value and throws BadDataException
  - readData has no handler for exception and terminates
  - readFile has no handler for exception and terminates immediately after closing the Scanner object
- DataAnalyzer.main has handler for BadDataException
  - Handler prints a message
  - User is given another chance to enter file name
  - The program does not crash!

# BadDataException.java

```
/**
	This class reports bad input data.
*/
public class BadDataException extends IOException{
	public BadDataException() {}
	public BadDataException(String message) {

	}
}
```

# Self Check 11.25

- Suppose the user specifies a file that exists and is empty. Trace the flow of execution.
- Answer:
  - DataAnalyzer.main calls DataSetReader.readFile, which calls readData.
  - The call in.hasNextInt() returns false, and readData throws a BadDataException.
  - The readFile method doesn't catch it, so it propagates back to main, where it is caught.

# Self Check 11.27

- What happens to the Scanner object if the readData method throws an exception?
- Answer: The close method is called on the Scanner object before the exception is propagated to its handler.

# Thank you

Please let me know if you have any questions.

# Chapter 12 - Object-Oriented Design

# Discovering Classes

- When designing a program, you work from a requirements specification
  - The designer's task is to discover structures that make it possible to implement the requirements
- To discover classes, look for nouns in the problem description.
- Find methods by looking for verbs in the task description.

# Example: Invoice

**INVOICE**

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

| Item | Qty | Price | Total |
|---|---|---|---|
| Toaster | 3 | $29.95 | $89.85 |
| Hair Dryer | 1 | $24.95 | $24.95 |
| Car Vacuum | 2 | $19.99 | $39.98 |

***AMOUNT DUE:*** **$154.78**

**Figure 1** An Invoice

# Example: Invoice

- Classes that come to mind:
  - `Invoice`
  - `LineItem`
  - `Customer`
- Good idea to keep a list of candidate classes.
- Brainstorm: put all ideas for classes onto the list.
- Cross not useful ones later.
- Concepts from the problem domain are good candidates for classes.
- Not all classes can be discovered from the program requirements:
  - Most programs need tactical classes

# The CRC Card Method

© Oleg Prikhodko/iStockphoto.

In a class scheduling system, potential classes from the problem domain include Class, LectureHall, Instructor, and Student.

# The CRC Card Method

- After you have a set of classes
  - Define the behavior (methods) of each class
- Look for verbs in the task description
  - Match the verbs to the appropriate objects
- The invoice program needs to compute the amount due
  - Which class is responsible for this method?
    - Invoice class

# The CRC Card Method

- To find the class responsibilities, use the CRC card method.
- A CRC card describes a class, its responsibilities, and its collaborating classes.
  - CRC - stands for “classes”, “responsibilities”, “collaborators”
- Use an index card for each class.
- Pick the class that should be responsible for each method (verb).
- Write the responsibility onto the class card.
- Indicate what other classes are needed to fulfill this particular responsibility (collaborators).

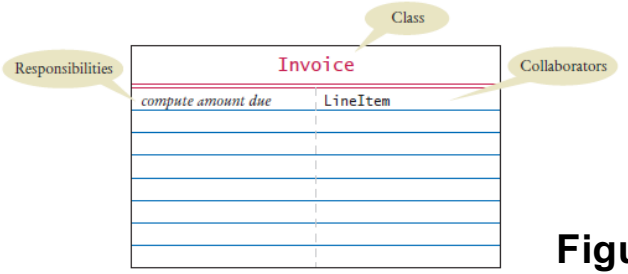


**Figure 2** A CRC Card

# Self Check 12.1

What is the rule of thumb for finding classes?

**Answer:** Look for nouns in the problem description.

# Self Check 12.2

Your job is to write a program that plays chess. Might `ChessBoard` be an appropriate class? How about `MovePiece`?

# Self Check 12.2

Your job is to write a program that plays chess. Might `ChessBoard` be an appropriate class? How about `MovePiece`?

**Answer:** Yes (`ChessBoard`) and no (`MovePiece`).

# Self Check 12.3

Suppose the invoice is to be saved to a file. Name a likely collaborator.

**Answer:** `PrintStream`

# Self Check 12.4

Looking at the invoice in Figure 1, what is a likely responsibility of the `Customer` class?

**Answer:** To produce the shipping address of the customer.

# Self Check 12.5

What do you do if a CRC card has ten responsibilities?

**Answer:** Reword the responsibilities so that they are at a higher level, or come up with more classes to handle the responsibilities.

# Relationships Between Classes

The most common types of relationships:

- Dependency
- Aggregation
- Inheritance


Questions?