Week 3: Review of Java Exception Handling

Primary Sources:
• Chapter 11 in Supplementary Book (Murach’s Java Programming)
• Appendices C (pp. C6 – C7) & E (pp. E1 – E19) in Textbook (Carrano)

Secondary Sources:
[1] Ch. 9: Error Handling with Exceptions, Thinking in Java, B. Eckel, 2002
[3] R. Waldhoff’s Weblog, Java's checked exceptions were a mistake, 2003,
http://www.mindview.net/Etc/Discussions/CheckedExceptions
Motivation


WWDC keynotes are usually carefully orchestrated affairs. But you might have just read in our live blog how Steve Jobs had to bail on a demo because of network trouble. Awkward.

Apple attempted to do their demo over Wi-Fi, but as you can see couldn't manage to get things up and running due to overload. So Jobs had to ask everyone in the audience, repeatedly, to shut off their Wi-Fi so the show could go on. A bizarre hiccup for a company in its prime, showing off its latest wares.

…
Java **Exceptions:** Simple Example

```java
import java.util.Scanner;

public class DemoSimpleExceptions {
    public static int quotient(int numerator, int denominator) {
        return numerator / denominator;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer numerator: ");
        int x = sc.nextInt();
        System.out.print("Enter an integer denominator: ");
        int y = sc.nextInt();
        int result = quotient(x, y);
        System.out.println("quotient = ", result);
    }
}
```

A. What happens if the user enters 10 for numerator and 3 for denominator when prompted?
   quotient = 3

B. What happens if the user enters 10 for numerator and 0 for denominator when prompted?
   java.lang.ArithmeticException

C. What happens if the user enters 10 for numerator and "abcd" for denominator when prompted?
   java.util.InputMismatchException
Motivation

• User Perspective: Robust Software
  • How do you feel when a software freezes, crashes, or gives blue screen of death?
  • Example: glitch in Apple iphone4 demo by Steve Jobs (2010)

• Programmer Perspective: Simplify Code!
  • Separate normal from abnormal in code
  • Unified, standard ways to report & manage unusual events

• In Java, exceptions are the preferred way to manage unusual events
  • Java assert() is for debugging
  • C, C++, Perl have many ways, e.g., error(), carp(), croak(), …

• Java introduced checked exceptions
  • Must either be caught or explicitly thrown
  • Controversial: mixes normal and abnormal code, burdens programmers [3,4]
  • Python etc. are moving away from checked exceptions!
Java Exceptions: Simple Example Revisited

```java
import java.util.Scanner;
public class DemoSimpleExceptions {
    public static int quotient( int numerator, int denominator)
    {
        return numerator / denominator;
    }
    public static void main( Strings[] args ) {
        Scanner sc = new Scanner(System.in);
        boolean problem = true;
        do {
            try {
                System.out.print("Enter an integer numerator: ");
                int x = sc.nextInt();
                System.out.print("Enter an integer denominator: ");
                int y = sc.nextInt(); int result = quotient( x, y );
                System.out.println("quotient = ", result );
                problem = false;
            } catch (ArithmeticException e) { 
                System.out.println("An ArithmeticException !");
            } catch (InputMismatchException e) { 
                System.out.println("InputMisMatchException !");
            }
        } while ( problem );
    }
}
```
Outline

- Objectives
- Basic Concepts
- Handling and Exception
- Throwing an Exception
- Programmer Defined Exception Classes
- Review Quiz
Objectives

Applied

- Given a method that throws one or more exceptions, code a method that calls that method and catches the exceptions.
- Given a method that throws one or more exceptions, code a method that calls that method and throws the exceptions.
- Code a method that throws an exception.
- Use the methods of the Throwable class to get information about an exception.
- Code a class that defines a new exception, and then use that exception in an application.
- Use an assert statement in an application to make an assertion about a condition.
Objectives (cont.)

- Given Java code that uses any of the language elements presented in this chapter, explain what each statement does.

Knowledge

- Explain what assertions are and how you can use them in your applications.
- Describe the Throwable hierarchy and the classes that are derived from it.
- Describe the difference between checked and unchecked exceptions and explain when you need to catch each.
- Explain how Java propagates exceptions and how it uses the stack trace to determine what exception handler to use when an exception occurs.
Objectives (cont.)

- Describe the order in which you code the catch clauses in a try statement.
- Explain when the code in the finally clause of a try statement is executed and what that code typically does.
- Explain what it means to swallow an exception.
- Explain how a try-with-resources statement works.
- Explain how a multi-catch block works.
- Describe three situations where you might want to throw an exception from a method.
- Describe two situations where you might create a custom exception class.
Outline

• Objectives
• Basic Concepts
  • Java Throwable Hierarchy
  • Error
  • Exceptions: Checked & Unchecked
• Handling and Exception
• Throwing an Exception
• Programmer Defined Exception Classes
• Review Quiz
The Throwable hierarchy

Leading Question: What is an Error?
Catastrophic events, which justify termination of program!
**assert statement**

```java
assert booleanExpression [: message ];
```

**Example Code using assert**

```java
for (int i = 1; i <= months; i++)
{
    futureValue = (futureValue +
    monthlyInvestment) * monthlyInterestRate;
}
// future value should be at least monthlyInvestment * months

assert(futureValue > monthlyInvestment * months):"FV out of range";
```

**Note:** Assert statement is used for **debugging**. It prints a message and exits a program. By default assert statement are disabled at runtime.

Ex. “java –ea myProgram” executes myProgram with assertions enabled.

Q? How does one switch it on in Eclipse for debugging?
What are Java (Errors &) Exceptions?

- **Errors**: catastrophic events justifying termination of program
- **Exception**: non-catastrophic unusual events
- **Q?** Have we come across any Exceptions so far? Name a few.

- **Throwable**
  - **Errors**
    - AssertionError,
    - LinkageError, e.g., NoClassDefFoundError
    - VirtualMachineError e.g., OutOfMemoryError
  - **Exception:**
    - ClassNotFoundException
    - IOException
      - EOFException, FileNotFoundException
    - run-timeException
      - ArithmeticException
      - IllegalArgumentException
      - IndexOutOfBoundsException, e.g., ArrayIndexOutOfBoundsException
      - NullPointerException
Common **checked** exceptions

- ClassNotFoundException
- IOException
- EOFException
- FileNotFoundException
- NoSuchMethodException

Common **unchecked** exceptions

- ArithmeticException
- IllegalArgumentException
- NumberFormatException
- IndexOutOfBoundsException
- ArrayIndexOutOfBoundsException
- StringIndexOutOfBoundsException
- NullPointerException
- InputMismatchException

**Q?** What is the difference between **checked** and **unchecked** exceptions?

A Java Program must handle “checked” exception or face termination!

Either handle immediately (e.g., catch)

or postpone (e.g., throw now) and handle later (e.g., catch in caller chain)
Outline

• Objectives
• Basic Concepts
• Handling an Exception
  • throw, catch
  • try {} catch {} finally {} statement
  • Multi-exception catch
  • Methods on exceptions: getMessage(), toString(), printStackTrace()
• Throwing an Exception
• Programmer Defined Exception Classes
• Review Quiz
Handling checked exceptions

- **Throw** the exception to the calling method
- **Catch** the exception and handle it

How Java handles Exceptions

Choice 1: Handle it now:
```
try{} catch {} statement
```

Choice 2: Postpone Handling:
```
throw statement
```
The syntax of the try statement

```
try {statements}
[catch (MostSpecificExceptionType e) {statements}]
...
[catch (LeastSpecificExceptionType e) {statements}]
[finally {statements}]
```

**Simple Example:**
```java
class TryCatch {
    public static void main(String args[]) {
        try {
            if (args.length > 0) throw(new Exception("Thrown"));
            return;
        }
        catch (Exception e) {
            System.out.println("Exception caught");return;}
        finally {
            // Execute this Code block, no matter how try block exited!
            System.out.println("finally");
        }
    }
}
```

**Console output:**
```
Exception caught
finally
```

**Alternative Console output:**
```
finally
```
The syntax of the try statement

```
try { statements }
[catch (MostSpecificExceptionType e) { statements }]
...
[catch (LeastSpecificExceptionType e) { statements }]
[finally { statements }]
```

Example 2: try { } catch { } catch { } ... finally { }

```java
public static String readFirstLine(String path){
    RandomAccessFile in = null;
    try {
        in = new RandomAccessFile(path, "r"); // may throw FileNotFoundException
        String line = in.readLine(); // may throw IOException
        return line;
    }
    catch (FileNotFoundException e) {
        System.out.println("File not found."); return null;
    }
    catch (IOException e) {
        System.out.println("I/O error occurred."); return null;
    }
    finally {
        try { if (in != null) in.close(); // may throw IOException
            } catch (Exception e){
                System.out.println("Unable to close file.");
            }
    }
}
```
Review Quiz: Exception Handling

1. From which problems is it possible for a program to recover?
   (a) Errors, e.g., divide by zero, dereference a null pointer, ...
   (b) Exceptions, e.g., ArithmeticException, NullPointerException, ...
   (c) Both errors and exceptions

2. Which exceptions is a program required to catch?
   (a) None. You can write a program to catch just the exceptions you want.
   (b) All. If a program is not written to catch all exceptions it will not compile.
   (c) Checked Ones, e.g., a program can not do I/O unless it catches all exceptions.

3. Which statement is FALSE about the try{} block?
   (a) Some of the statements in a try{} block will never throw an exception.
   (b) The statements in a try{} block may throw several types of exception.
   (c) The try{} block can not contain loops or branches.
   (d) The try{} block must appear before the catch{} blocks.

4. Which statement is FALSE about catch{} blocks?
   (a) There can be several catch{} blocks in a try/catch structure.
   (b) The catch{} block for a child exception class must PRECEDE that of a parent exception class.
   (c) The catch{} block for a child exception class must FOLLOW that of a parent exception class.
   (d) If there is no catch{} block there must be a finally{} block.

5. When is a finally{} block executed?
   (a) Only when an unhandled exception is thrown in a try{} block.
   (b) Only when any exception is thrown in a try{} block.
   (c) Always after execution has left a try{} block, no matter for what reason.
   (d) Always just as a method is about to finish.
The syntax of the try statement

```java
try {statements}  
[catch (MostSpecificExceptionType e) {statements}]  
...  
[catch (LeastSpecificExceptionType e) {statements}]  
[finally {statements}]  
```

The syntax of the try-with-resources statement

```java
try (statement[;statement] ...) {statements}  
[catch (MostSpecificExceptionType e) {statements}]  
...  
[catch (LeastSpecificExceptionType e) {statements}]  
```

Q?: Compare and contrast above two forms of “try” statement:
Which one allows “finally” clause?
Which one automatically closes resources?
Which one do you prefer? Why?
The syntax of the try-with-resources statement

```
try (statement[;statement] ...) {statements} 
[catch (MostSpecificExceptionType e) {statements}] ... 
[catch (LeastSpecificExceptionType e) {statements}] 
```

Example: automatically close the specified resource

```
public static String readFirstLine(String path) 
{ 
  try (RandomAccessFile in = new RandomAccessFile(path, "r")) 
  { 
    String line = in.readLine(); //may throw IOException 
    return line; 
  } 
  catch (FileNotFoundException e) 
  {System.out.println("File not found."); return null; } 
  catch (IOException e) 
  {System.out.println("I/O error occurred."); return null;} 
  // finally { in.close(); } is implicit! 
} 
```
Four methods available from all exceptions

- `getMessage()`
- `toString()`
- `printStackTrace()`
- `printStackTrace(outputStream)`

How to print exception data to standard output stream

```java
catch(IOException e) {
    System.out.println(e.getMessage() + "\n");
    System.out.println(e.toString() + "\n");
    e.printStackTrace(System.out);
    return null;
}
```

Leading Question:
Should error message be printed on standard output? Is there an alternative?
How to print exception data to the error output stream

catch (IOException e)
{
    System.err.println(e.getMessage() + "\n");
    System.err.println(e.toString() + "\n");
    e.printStackTrace();
    return null;
}

Resulting output for a FileNotFoundException

c:\murach\java\files\produx.txt (The system cannot find the file specified)

java.io.FileNotFoundException: c:\murach\java\files\produx.txt (The system cannot find the file specified)

java.io.FileNotFoundException: c:\murach\java\files\produx.txt (The system cannot find the file specified)
    at java.io.RandomAccessFile.open(Native Method)
    at java.io.RandomAccessFile.<init>(RandomAccessFile.java:118)
    at ProductApp.readFirstLine(ProductApp.java:70)
    at ProductApp.main(ProductApp.java:10)
Syntax of multi-exception catch block

```
catch (ExceptionType
    | ExceptionType
    [ | ExceptionType]... e)
{statements}
```

Example: A multi-exception catch block

```java
public static String readFirstLine(String path)
{
    try (RandomAccessFile in = new RandomAccessFile(path, "r"))
    {
        String line = in.readLine(); // may throw IOException
        return line;
    }
    catch (FileNotFoundException | EOFException e)
    {
        System.err.println(e.toString()); return null;
    }
    catch (IOException e)
    {
        e.printStackTrace(); return null;
    }
}
```
A method with a **single-exception** catch block

```java
public static String readFirstLine(String path) {
    try (RandomAccessFile in = new RandomAccessFile(path, "r")) {
        String line = in.readLine(); // may throw IOException
        return line;
    }
    catch (FileNotFoundException e) {
        System.err.println(e.toString());
        return null;
    }
    catch (EOFException e) {
        System.err.println(e.toString());
        return null;
    }
    catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```
Outline
• Objectives
• Basic Concepts
• Handling and Exception
• Throwing Exception
  • Method header and checked Exception
  • Statement for Explicit Throw
• Programmer Defined Exception Classes
• Review Quiz
Declaration: Method may throw a checked exceptions

```
modifiers returnType methodName([parameterList])
    throws exceptionList {}
```

Ex.: `getFileSizeLength()` may throw IOException

```
public static long getFileSizeLength() throws IOException
{
    RandomAccessFile in = new RandomAccessFile(path, "r");
    long length = in.length();// may implicitly throw IOException
    return length;
}
```

Leading Question: What may a caller of `getFileSizeLength()` do with IOException?
• Note IOException is a checked Exception!
• Choice for caller of `getFileSizeLength()`
  • Catch IOException from `getFileSizeLength()`
  • Re-throw IOException from `getFileSizeLength()`
  • Compiler error
Either Caller of getFileLength() catches IOException

```java
public static int getRecordCount2()
{
    try
    {
        long length = getFileLength(); // may throw IOException
        int recordCount = (int) (length / RECORD_SIZE);
        return recordCount;
    }
    catch (IOException e)
    {
        System.err.println("An IO error occurred."); return 0;
    }
}
```

Or Caller of getFileLength() throws IOException

```java
public static int getRecordCount3() throws IOException
{
    long length = getFileLength(); // may throw IOException
    int recordCount = (int) (length / RECORD_SIZE);
    return recordCount;
}
```

Compiler error if method neither catches nor throws a checked exception

```
C:\murach\java\netbeans\examples\ch14\src\ProductApp.java:12: error: unreported exception IOException; must be caught or declared to be thrown
    getRecordCount()
```
Review Quiz: Exception Handling

1. Which of the following are “checked” exceptions?
   (a.) OutOfMemoryError (b.) AssertionError (c.) ClassNotFoundException (d.) EOFException
   (e.) FileNotFoundException (f.) NoSuchMethodException (g.) ArithmeticException
   (h.) IllegalArgumentException (i.) IndexOutOfBoundsException (j.) NullPointerException

2. Consider the following code fragment:
   ```java
   Scanner sc = new Scanner(System.in);
   boolean tryAgain = true;  int num = 0;
   System.out.print("Enter a number less than 10: ");
   while (tryAgain == true) {
       try {
           num = sc.nextInt();
           if (num > 9) throw new InputMismatchException();
           tryAgain = false;
       } catch (InputMismatchException ime) {
           System.out.print("Enter a valid integer: ");
       }
   } 
   sc.nextLine();
   ```

2A. What happens if the user enters “11” at the prompt?
   a. An InputMismatchException is thrown and tryAgain is set to false.
   b. An InputMismatchException is thrown and tryAgain remains set to true.
   c. A runtime error occurs.
   d. No exceptions are thrown and tryAgain is set to false.

2B. Repeat question 1. with user input of 5.
2C. Repeat question with user input of “abc”
Outline

• Objectives
• Basic Concepts
• Handling and Exception
• Throwing Exception
  • Method header and checked Exception
  • Statement for Explicit Throw
• Programmer Defined Exception Classes
• Review Quiz
The syntax of the throw statement

```java
throw throwableObject;
```

Common constructors of the Throwable class

- `Throwable()`
- `Throwable(message)`
A method that throws an unchecked exception

```java
public double calculateFutureValue(double monthlyPayment, double monthlyInterestRate, int months)
{
    if (monthlyPayment <= 0)
        throw new IllegalArgumentException("Monthly payment must be > 0");
    if (monthlyInterestRate <= 0)
        throw new IllegalArgumentException("Interest rate must be > 0");
    if (months <= 0)
        throw new IllegalArgumentException("Months must be > 0");

    // code to calculate and return future value goes here
}
```

Q?: Why does method header not include `throws IllegalArgumentException`?
• Is IllegalArgumentException a checked Exception?
Using throw statement in “try” (to test throw statement)

try
{  // code that reads the first line of a file
    if (true) throw new IOException("I/O exception test");
    return firstLine;
}
catch (IOException e)
{
    // code to handle IOException goes here
}

Using throw statement in “catch” (Relay/Rethrow)

try {
    // code that throws IOException goes here
}
catch (IOException e)
{
    System.out.println("IOException thrown in getFileLength method.");
    throw e;
}
Outline

• Objectives
• Basic Concepts
• Handling and Exception
• Throwing Exception
• **Programmer Defined Exception Classes**
• Review Quiz
When to define your own exceptions

- When a method requires an exception that isn’t provided by any of Java’s exception types
- When using a built-in Java exception would inappropriately expose details of a method’s operation

Programmer-Defined Exception: DAOException class

```java
public class DAOException extends Exception {
    public DAOException() {
    }
    public DAOException(String message) {
        super(message);
    }
}
```
Catching a programmer defined Exception

try
{
    Product p = getProduct("1234");
}
catch (DAOException e)
{
    System.out.println(e.getMessage());
}

A method that throws programmer-defined Exception

public static Product getProduct(String pCode) throws DAOException
{
    try
    {
        Product p = readProduct(pCode);  // may throw IOException
        return p;
    }
    catch (IOException e)
    {
        throw new DAOException("Error in reading product " + pCode);
    }
}
Methods in Throwable class

- `getCause()`  // Get information on cause of an exception
- `initCause(cause)`  // Get initial trigger for a given cause

Code with `getCause()` for a DAOException

```java
try {
    // Code that may throw DAOException
}
catch (DAOException e) {
    System.out.println("DAOException: Error reading product");
    System.out.println(e.getCause().toString());
}
```

Resulting output

```
DAOException: Error reading the product
java.io.IOException: I/O exception test
```
Constructors of the Throwable class

- Throwable(cause)
- Throwable(message, cause)

Constructors for Programmer Defined Exception

```java
public class DAOException extends Exception {
    public DAOException() { } // Default constructor
    public DAOException(Throwable cause) // Another Constructor
    {
        super(cause);// Let superclass initialize inherited attributes
    }
}
```

Using a constructor for DAOException

```java
catch (IOException e)
{
    throw new DAOException(e); // Q. Which constructor is invoked?
}
```
Outline

• Objectives
• Basic Concepts
• Handling and Exception
• Throwing an Exception
• Programmer Defined Exception Classes
• Review Quiz
Review Quiz: Exception Handling

1. When coding catch blocks, for exceptions you should always code them in what order?
   (a.) More specific before less specific (b.) More general before specific, (c.) order does not matter

2. In which situation(s) would you typically not throw an exception from a method?
   a. When you want to test an exception handler
   b. When you need to perform some processing before exception is handled by calling method
   c. When invalid arguments are passed to the method
   d. When the exception is handled by the method

3. Which of the following statements is not true about assertions?
   a. They let you test a condition as an application executes.
   b. They let you print information to the standard error stream when an AssertionError occurs.
   c. They let you change the values of variables as an application executes.
   d. They can be enabled or disabled as an application executes.

4. In which of the following situations should you define a custom exception class?
   a. When you want to change the message for an exception
   b. When none of the Java exceptions are appropriate
   c. When using a Java exception would expose too many details of a method’s operation.
   d. All of the above.

5. The try-with-resources statement doesn’t need a finally clause because it
   a. uses a multi-catch block to catch all exceptions
   b. implements the AutoCloseable interface
   c. automatically closes all declared resources
   d. catches all exception that are thrown when you try to close a resource
Review Quiz: Exception Handling

1. What is wrong with the following exception handler as written?
   ```java
   try { }
   catch (Exception e) { }
   catch (ArithmeticException a) { }
   ```

2. Classify following into error, checked exception, compile error or no_exception.
   (a.) `int[] A; A[0] = 0;`
   (b.) A program is reading a stream and reaches the end of stream marker.
   (c.) Before closing the stream and after reaching the end of stream marker, a program tries to read the stream again.

3. Modify the following `cat` method so that it will compile.

   ```java
   public static void cat(File file) {
       RandomAccessFile input = null; String line = null;
       try { input = new RandomAccessFile(file, "r");
           while ((line = input.readLine()) != null)
               { System.out.println(line); } return;
       } finally { if (input != null) { input.close(); } }
   }
   ```