NC STATE UNIVERSITY

Exceptions and Libraries

Zybooks Chapter 10

Some slides created by Marty Stepp
http://www.cs.washington.edu/143/
Edited by Sarah Heckman

2

NC STATE UNIVERSITY

Exceptions

- **Exception**: An *object* representing an error or unusual condition.
  - **Unchecked exceptions:** An exception that does not have to be handled by the caller for the program to compile
  - **Checked exception**: An exception that **must be handled by the caller** for the program to compile.

- **Handling Exceptions**
  - **Throw** the exception from the method (including a `throws` clause on the method header for checked exceptions)
  - **Catch** the exception and do something

3

Throwing an Exception

- A method can throw an exception by:
  - Calling a method that throws an exception
  - Constructing and then throwing an exception
- If a method throws a *checked* exception, then the exception information must be included in the method's header
  - Unchecked exceptions do not require the **`throws`** clause
- All exceptions should be documented in Javadoc comments using **`@throws`** tag
  - **`throws`** clause(s) ***requires*** Javadoc with the **`@throws`** tag

Throwing an Exception Example

Javadoc!!!

Throws clause is required for the checked exception (but not for the unchecked exception).

```
* @throws FileNotFoundException when…
* @throws IllegalArgumentException when…
*/
public type readFile(String name) throws FileNotFoundException {
   if (name == null) {
      throw new IllegalArgumentException();
   }

   Scanner in = new Scanner(new File(name));
}
```

Throwing a constructed exception

Calling a method or constructor that throws a FileNotFoundException

NC STATE UNIVERSITY

Catching an Exception

- If you call a method that might throw an exception, you can
  - Wrap the method call in a **`try`** block
  - And if the exception is thrown, you can **`catch`** it and respond as per the requirements/design
- Catching an Exception Flow-of-Control
  - **`try`** block executes
    - Must contain at least one statement that could throw an exception
  - If an exception is thrown
    - The rest of the **`try`** block is skipped
    - Flow-of-control switches to the first matching exception in the **`catch`** block
    - Statements in the **`catch`** block are executed
    - All other **`catch`** blocks are skipped
  - If an exception is not thrown, all **`catch`** blocks are skipped

NC STATE UNIVERSITY

Multi-catch

- Java 1.7 and Later
- For two specific **`Exceptions`** that have common catch functionality

```
try {
  //Code that will cause multiple Exceptions
} catch (ExceptionType1 | ExceptionType2 e) {
  //Common exception code
}
```

Finally Blocks

- `try/catch` blocks can also have a corresponding `finally` block
- If any code in the `try` block is executed then the code in the `finally` block is guaranteed to be executed – even if the exception occurs
- Used to clean up resources an reduce code duplication:
  - Close file streams
  - Close database streams

Exception Tracing

```
public static int readFile(File f) {
   Scanner fileScanner = null;
   int sum = 0;
   try {
      fileScanner = new Scanner(f);
      while (fileScanner.hasNextLine()) {
         sum += fileScanner.nextInt();
      }
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (InputMismatchException e) {
      e.printStackTrace();
   } finally {
      if (fileScanner != null) {
         fileScanner.close();
      }
   }
   return sum;
}
```

NC STATE UNIVERSITY

Try-with-Resources

- Java 1.7 and greater
- Declares one or more resources that must be closed
  - Resource must implement **`AutoCloseable`**

```
public String readLineFromFile(String path) throws FileNotFoundException {
   try (Scanner in = new Scanner(new File(path))) {
      return in.readLine();
   }
}
```

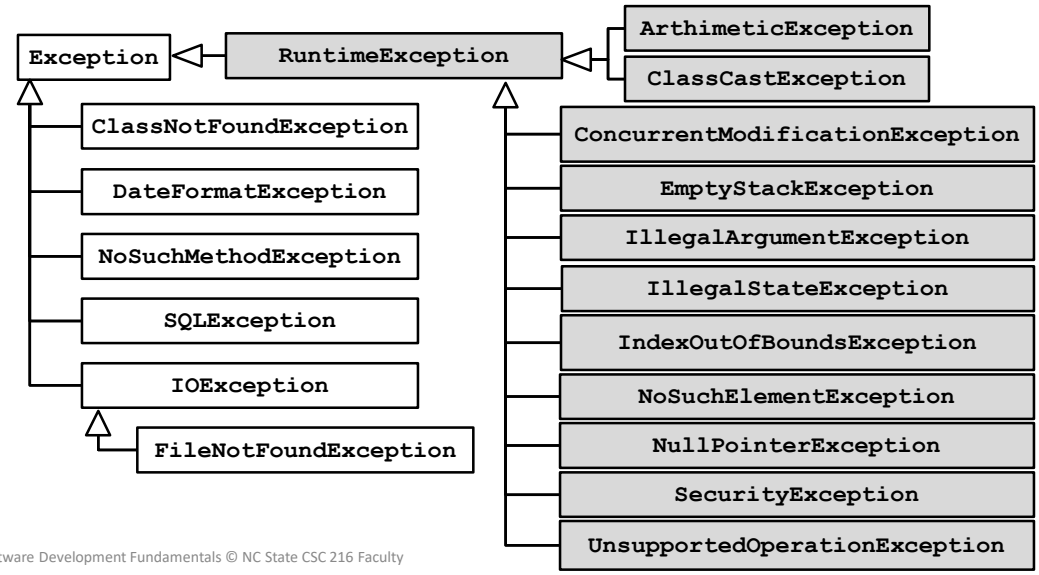
Example modified from: http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

NC STATE UNIVERSITY

Handling Exceptions

- There are lots of options for handling exceptions
  - Try again
  - Prompt the user for new information
  - Display an error message
    - The **`Exception`** class provides a **`getMessage()`**, **`getStackTrace()`**, and **`toString()`** to help with this.
  - Quit the program
  - Do nothing

Exception Inheritance Hierarchy



Inheritance and Exceptions

- You can catch a general exception to handle any subclass:

```
try {
    Scanner input = new Scanner(new File("myfile"));
    System.out.println(input.nextLine());
} catch (Exception e) {
    System.out.println("File was not found.");
}
```

- Similarly, you can state that a method throws any exception:

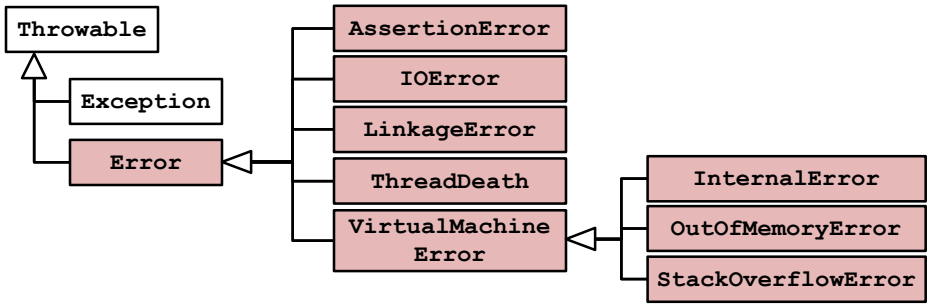
```
public static void method() throws Exception { ...
}
```

- Are there any disadvantages of doing so?

NC STATE UNIVERSITY

Exceptions and errors

- There are also **`Errors`**, which represent serious Java problems.
  - **`Error`** and **`Exception`** have common superclass **`Throwable`**.
  - You can catch an **`Error`** (but you probably shouldn't)



NC STATE UNIVERSITY

Custom Exceptions

- You can create your own exceptions by extending **`Exception`** or any of its child classes
  - If you want a checked exception, extend **`Exception`**
  - If you want an unchecked exception, extend **`RuntimeException`**
- Why would we want to do this?

```
public class MyException extends Exception {
   public MyException(String message) {
      super(message);
   }
   public MyException() {
      this("Exception message");
   }
}
```

Using Custom Exceptions

- You use your custom exception (or any in the Java API) by throwing it.
- Code calling your method must either
  - Pass the exception on – which may require a **`throws`** in header
  - Catch the exception and handle

```
public void myMethod() throws MyException {
   if (<some condition>) {
      throw new MyException("My new error message");
   }
}
```


LIBRARIES

NC STATE UNIVERSITY

Libraries

- A library is a 3rd party collection of code that can be used in other programs
  - Promotes code reuse
  - Don't need to "reinvent the wheel" to complete common tasks
  - Example: Java API
  - Example: JUnit

NC STATE UNIVERSITY

Using Libraries

- Download the Library
- Install the Library
  - An executable (Java Libraries)
  - A jar executable to download rest of library (JAXB)
  - A jar file (JUnit, Apache libraries, etc.)
- Include Library on path when compiling and executing your application

NC STATE UNIVERSITY

Classpaths

- The path(s) where Java looks for libraries (i.e., referenced code) when compiling and running a program
- Use the `-cp` argument to the java command
- Separate directories/jars with a semi-colon (;)
- Each jar should be listed individually
  - You cannot specify a directory of jars
- Alternatively, you can update the System's `PATH` environment variable
  - Make sure you restart your terminal so the new path will be available

NC STATE UNIVERSITY

Creating a User Library in Eclipse

- Right-click on your project, select **Build Path > Configure Build Path**
- Select **Add Library … > User Library > User Libraries….**
- Select **New** and give your library a name. You do NOT need to add your library to the system path
  - But if you do add it to the system path, you can use it from the command line
- Select your library and press the **Add Jars** button. Brows your file system and select the ***.jar** files appropriate for your library. Press **OK**. Press **Finish**. Press **OK**.

NC STATE UNIVERSITY

Creating a User Library in Eclipse (2)

- If working with a partner, you should agree on a library name (case matters)
- Each partner will create a local version of the library that points to the jar(s) on their system
- The project must have the library name associated with it
- If there are build path errors, the project icon will be highlighted with a big red explanation point!

NC STATE UNIVERSITY

Other Library Setups

- Some projects store all relevant *.jar files in a lib/ directory
  - Jars are added to lib in file system
  - In Eclipse, select the Add Jar option and add the jar(s)
- Building applications, like Maven, will download the required jar(s) from online repositories
- Building applications, like Ant and Maven, simplify compilation and running when using the command line rather than an IDE

Using Libraries on the Command Line

- Command to compile and run
  - Assumption is that the project is set up in directory structure like Eclipse project

```
eos% javac  -d bin -sourcepath src -sourcepath test -cp path1;path2;path3
src/pack/age1/*.java src/pack/age2/*.java test/pack/age2/*.java

eos% java -cp ./bin;path1;path2;path3 pack.age1.ClassName
```


APIs

- API: Application Programming Interface
  - Series of webpages
  - Describe the public classes, interfaces, and methods of Java class libraries
  - Describe what functionality is available for you, the client, of the libraries to use
- Example of abstraction
  - API tells you what a class can do, but doesn't tell you how the class does it

API Layout

- Packages: Upper left
- Classes: Lower left
- Details: Main frame
  - List of packages and descriptions
  - List of classes and descriptions
  - List of class members and descriptions
    - Inheritance hierarchy
    - Public fields and constants
    - Constructors
    - Methods

Creating Our Own APIs

- Sometimes we're the client of a library
  - Java API
  - 3rd Party Libraries – JFreeChart, JUnit, JAXB, etc.
- Sometimes we're writing code that others will use
  - Need to provide an API for those clients
- Sometimes we're working with a large team and other members of the team may need to know what we're doing
  - Need to provide high level details