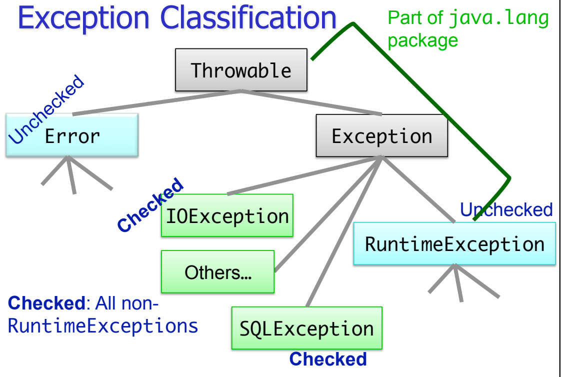
Objectives

- Catching Exceptions
- Files
- Streams

Review

- What are the two types of exceptions?
- What is one way to handle exceptions?
- What does it mean to “advertise” an exception?

Exception Classification



CATCHING EXCEPTIONS

Catching Exceptions

- After we throw an exception, some part of program needs to *catch* it
  - Knows how to deal with the situation that caused the exception
  - Handles the problem—hopefully gracefully, without exiting

Try/Catch Block

- The simplest way to catch an exception
- Syntax:

```
try {
      code;
      more code;
}
catch (ExceptionType e) {
      error code for ExceptionType;
}
catch (ExceptionType2 e) {
      error code for ExceptionType2;
}
…
```

Try/Catch Block

```
try {
        code;
        more code;
}
catch (ExceptionType e) {
        error code for
        ExceptionType
}
```

- Code in `try` block runs first
- If `try` block completes without an exception, `catch` block(s) are skipped
- If `try` code generates an exception
  - A `catch` block runs
  - Remaining code in `try` block is skipped
- If an exception of a type other than `ExceptionType` is thrown inside `try` block, method exits immediately*

Try/Catch Block

```
try {
        code;
        more code;
}
catch (ExceptionType e) {
        error code for
        ExceptionType
}
catch (ExceptionType2 e) {
        error code
        for ExceptionType2
}
```

Can catch any exception with `Exception e` but won't have customized messages

- You can have more than one `catch` block
  - To handle > 1 type of exception
- If exception is not of type `ExceptionType1`, falls to `ExceptionType2`, and so forth
  - Run the first matching `catch` block

Try/Catch Example

```
public void read(BufferedReader in) {
        try {
                boolean done = false;
                while (!done) {
                        String line=in.readLine();
                        // above could throw IOException!
                        if (line == null)
                                done = true;
                }
        }
        catch (IOException ex) {
                ex.printStackTrace();
        }
}
```

Prints out stack trace to method call that caused the error

Try/Catch Example

```
public void read(BufferedReader in) {
        try {
                boolean done = false;
                while (!done) {
                        String line=in.readLine();
                        // above could throw IOException!
                        if (line == null)
                                done = true;
                }
        }
        catch (IOException ex) {
                ex.printStackTrace();
        }
}
```

More precise `catch` may help pinpoint error
But could result in messier code

The `finally` Block

```
try {
        …
}
catch (Exception e) {
        …
}
finally {
        …
}
```

- Optional: add a `finally` block after all `catch` blocks
  - Code in `finally` block **always** runs after code in `try` and/or `catch` blocks
    - After `try` block finishes or, if an exception occurs, after the `catch` block finishes
- Allows you to clean up or do maintenance before method ends (one way or the other)
  - E.g., closing files or database connections

`FinallyTest.java`

Practice: try/catch/finally Blocks

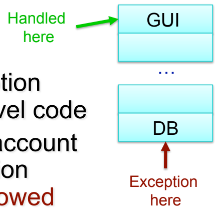
```
try {
        statement1;
        statement2;
}
catch (EOFException e) {
        statement3;
        statement4;
}
finally {
        statement5;
}
```

- Which statements run if:
  - Neither `statement1` nor `statement2` throws an exception
  - `statement1` throws an `EOFException`
  - `statement2` throws an `EOFException`
  - `statement1` throws an `IOException`

What to do with a Caught Exception?

- Dump the stack after the exception occurs
  - What else can we do?
- Generally, two options:
  1. Catch the exception and recover from it
  2. Pass exception up to whoever called it

To Throw or Catch?



- Problem: lower-level exception propagated up to higher-level code
- Example: user is entering account information and get exception message "field exceeds allowed length in database"
  - Lost context
  - Lower-level detail polluting higher-level API

Solution: higher-levels should catch lower-level exceptions and throw them in terms of higher-level abstraction

Exception Translation

```
try {
    // Call lower-level abstraction
}
catch (LowerLevelException ex) {
    // log exception …
    throw new HigherLevelException(…);
}
```

- Special case: Exception Chaining
  - When higher-level exception needs info from lower-level exception

```
try {
    // Call lower-level abstraction
}
catch (LowerLevelException cause) {
    // log exception …
    throw new HigherLevelException(cause);
}
```

Most standard `Exceptions` have this constructor

Guidelines for Exception Translation

- Try to ensure that lower-level APIs succeed
  - Ex: verify that your parameters satisfy invariants
- Insulate higher-level from lower-level exceptions
  - Handle in some reasonable way
  - Always log problem so admin can check
- If can't do previous two, then use exception translation

Summary: Methods Throwing Exceptions

- API documentation tells you if a method can throw an exception
  - If so, you **must** handle it
- If your method could possibly throw an exception (by generating it or by calling another method that could), advertise it!
  - If you can't handle every error, that's OK…let whoever is calling you worry about it
  - However, they can only handle the error if you advertise the exceptions you can't deal with

Programming with Exceptions

- Exception handling is slow
- Use one big `try` block instead of nesting `try` -`catch` blocks
  - Speeds up EH. Also, code gets too messy
- Don't ignore exceptions (e.g., `catch` block does nothing)
  - Better to pass them along to higher calls

```
try {
}
catch () {
}
try {
}
catch () {
}
```

```
try {
    try {
    }
    catch () {
    }
}
catch () {
}
```

```
try {
    …
    …
}
catch () {
}
```

Creating Our Own Exception Class

- Try to reuse an existing exception
  - Match in name as well as semantics
- If you cannot find a predefined Java `Exception` class that describes your condition, implement a new `Exception` class!

Creating Our Own Exception Class

```
public class FileFormatException extends IOException {
      public FileFormatException() {

      }

      public FileFormatException(String message) {
            super(message);
      }

      // other 2 standard constructors…
}
```

What happens in this constructor implicitly?

Is this a checked or unchecked exception?

- Can now throw exceptions of type `FileFormatException`

Guidelines for Creating Your Own Exception Classes

- Include accessor methods to get more information about the cause of the exception
  - "failure-capture information"
- Checked or unchecked exception?
  - Checked: *forces* API user to handle BUT more difficult to use API (has to handle all checked exceptions)
  - Use checked exception if exceptional condition cannot be prevented by proper use of API *and* API user can take a useful action afterward

Practice: Designing a New Exception Class

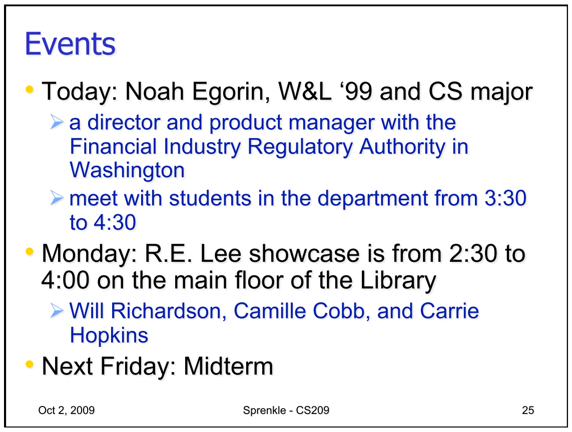
- Scenario: When an attempt to make a purchase with a gift card fails because card doesn't have enough money, throw a new exception that you created.
- Recall that all `Exceptions` are `Throwable`, so they have the methods: `getMessage()`, `printStackTrace()`, `getStackTrace()`

• How would someone else use your class?
• What constructors, additional method(s) may you want to add for your exception class?

Benefits of Exceptions?

Benefits of Exceptions

- Force error checking/handling
  - Otherwise, won't compile
  - Does not guarantee "good" exception handling
- Ease debugging
  - Stack trace
- Separates error-handling code from "regular" code
  - Error code is in catch blocks at end
  - Descriptive messages with exceptions
- Propagate methods up call stack
  - Let whoever "cares" about error handle it
- Group and differentiate error types



Events

- Today: Noah Egorin, W&L ‘99 and CS major
  - a director and product manager with the Financial Industry Regulatory Authority in Washington
  - meet with students in the department from 3:30 to 4:30
- Monday: R.E. Lee showcase is from 2:30 to 4:00 on the main floor of the Library
  - Will Richardson, Camille Cobb, and Carrie Hopkins
- Next Friday: Midterm

Oct 2, 2009 Sprenkle - CS209 25