CS212
Exceptions
Software Engineering

Fall 2004
Lecture 7

Announcements

- EA infosession, 5-7pm, Upson B17, free food
- P2 coming due...
- Part 3 coming up: functions, error handling

Overview

- Packages and Jar files tidbit
- Error handling with Exceptions
  - Java tutorial
    http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html
  - Part 3: Semantic/Syntax errors
- Some software engineering

Packages and Jar Files

- Packages:
  - http://java.sun.com/docs/books/tutorial/java/interpack/packages.html
  - short version:
    - create a directory path equivalent to package path (something.something.something....)
    - stick imported file in directory at end of path
- Jar files:
  http://java.sun.com/docs/books/tutorial/jar/basics/index.html

Errors

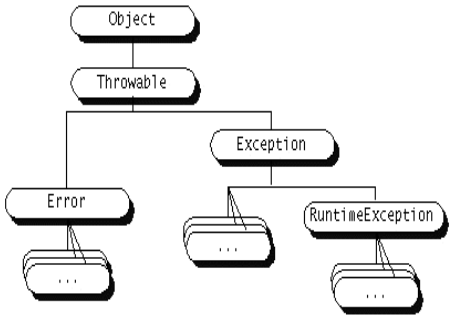
- Error (in general programming sense):
  - you've made a mistake!
- Categories:
  - language: syntax and semantic
  - runtime
  - logic
- When do those errors occur?
- How to handle?
  - write perfect language
  - all data is always legal
  - algorithms are precisely mapped
  - see http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html

Error Handling

- Want ways to deal with reality of errors
  - smart editors
  - smart compilers find errors
  - runtime environment alerts
- Common concept:
  - find error: ***catch***
  - alert something/something: ***throw***
- Java uses **`Object`**s:
  - **`Throwable`** object: special object that can be "sent" by code to other code (the "alert")
  - **`Error`**: special kind of **`Throwable`** for grievous error; not meant to recover
  - **`Exception`**: all other errors that can be dealt with (catch and throw)

Throwable Hierarchy

From Java tutorial:



Exceptions

- Java **`Exception`**:
  - Compile time (our focus)
  - Runtime
- Compile time
  - ***checked exception***
  - you must handle somehow in your code (why?)
  - example: see File I/O code
- Runtime:
  - ***unchecked exception***
  - you do not need to handle in your code (why?)
  - examples: array out of bounds, null pointers, divide by zero...

Aside: Your Compiler

- You, the programmer, are dealing with runtime exceptions...why?
- We, the users, get compile time exceptions when using your code...why?

Handling Exceptions

- Three operations:
  - claiming exception
    - method informs compiler about its possible exceptions
    - sometimes you see method headers like this:
      ```
      public void myMethod() throws IOException
      ```
  - throwing exception
    - statement can cause (compile time) or does cause (runtime) an error
    - method creates an exception object, which has info about program state, and passes to JVM
    - you can also deliberately throw an exception with **`throw`**
  - catching exception
    - Java looks for code that handles the exception
    - starts in current method and then works backward in chain of method calls

Catching Exceptions

- catching exceptions in body of method

```
try {
   statements
}
catch (Exception1 e) {
   statements
}
catch (Exception2 e) {
   statements
}
.
.
.
finally {
   statements
}
```

- examples...

Some Software Engineering

- Engineering
  ABET: "the profession in which a knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgment to develop ways to utilize, economically, the materials and forces of nature for the benefit of mankind."
- Software Engineering
  IEEE: "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."
- Is Software Engineering a type of Engineering?

General Development Principles

- Decomposition
  - "divide and conquer"
  - stepwise refinement
- Abstraction
  - work on details and then hide them
  - kinds: procedural, data, type, others... ("Generic Programming" in CS211)
- Specification
  - think interfaces
- Documentation

13

Validation

- Validation
  - how do you know your program actually works?
- Formal techniques (in Engineering, the theory!)
  - program correctness and proofs
  - need more CS
- Testing (in Engineering, the experiments!)
  - glassbox:
    examine implementation and attempt to test all possible "paths" through the program
  - blackbox:
    test cases are generated based on the specification without regard to implementation

14

Programming "in the large"

- Models for software development
  - waterfall and others
- Requirements analysis
  - functional, performance requirements
  - delivery schedule
- Data models
  - kinds and relations of data
  - UML very useful
- Program Design
  - top-down
  - bottom-up
  - stepwise refinement
  - coupling/cohesion
- Design patterns

15

Coding Quality

- Pareto's Law: 80/20 rule and variants
- Software?
  - 80% of defects caused by 20% of code
- NSA study [Drake, IEEE Computer, 1996] on 25 million lines of code
  - 70-80% of problems were due to 10-15% of modules
  - 90% of all defects were in modules containing 13% of the code
  - 95% of serious defects were from just 2.5% of the code

16