CMSC 131
Object-Oriented Programming I

Exceptions

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane
Exception Propagation

• When an exception occurs, Java looks in the current method for a catch clause that matches the exception. If one is found, the exception is handled; otherwise exception propagation takes place

• Exception Propagation
  • Java uses exception propagation to look for exception handlers
  • When an exception occurs, Java pops back up the call stack to each of the calling methods to see whether the exception is being handled in a catch block of the method. This is exception propagation
  • The first method it finds that catches the exception will have its catch block executed. Execution resumes normally in the method after this catch block
  • If we get all the way back to main and no method catches this exception, Java catches it and aborts your program

• Example: Propagation.java
Finally Block

- Code that is always run (finally → ALWAYS)
- Run no matter what
  - When no exception has been thrown
  - When exception is thrown
  - If you exit the method via return
    - For example if the return occurs before the finally clause, the clause will be executed and then we will return
- Example: Finally.java
Exceptions

• You can have multiple catch clauses
  • **Example:** Multiple.java
  • How do we know the exceptions thrown by the methods? By using the Java API
Exceptions

• Throwing Exceptions
  • You can throw exceptions using throw
  • Notice you can define your own exceptions and throw them
  • You can throw exceptions defined by Java
  • An exception can be rethrown
  • Example: Throwing.java

• Important: exceptions should be used for handling errors and not for implementing solutions to problems

• Never leave the catch clause empty
  • If you don’t know what to place, call printStackTrace
  • If you want code to compile, even though you have not implemented it, then use:
    throw new UnsupportedOperationException("You must implement this method.");
Immutable

- Immutable object
  - State is initialized when it is constructed but it will never change
  - Instance variables (state) cannot be changed
- Mutable object
  - Instance variables (state) can be changed
- There is no language construct that guarantees immutability
  - You have to design the class to enforce it
- Keep classes immutable (if possible)
  - If aliasing occurs it will not matter (no problem with sharing object if it cannot change)
- With mutable classes we need to make “defensive copies”
  - Changes made in one are not seen in another
- It is good practice to document whether a class is immutable
- How to define a class as immutable
  - Make all the instance variables private
  - Do not provide set methods
  - Only the constructor will initialize the instance variables
Anytime you are passing an object to a method or returning an object from a method you should ask whether a copy should be made.

Strings are immutable:
- Great! → Not need to make copies when we pass them to methods.
- Bad! → Inefficient to modify one.
  - Example: loop that reads characters one at a time and makes a string out of it (creates a lot of objects!)
A mutable String class
- Good → very efficient to modify them
- Bad → we have to worry about aliasing

Main methods
- **append** → add characters to end
- **insert** → add characters in middle
- **delete** → remove characters

Note
- **append, insert** → return object of type StringBuffer
  - This is alias to object that the methods belong to

API:
http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html

**Example:** StringBufferExample.java