CLASS 6: CLASSES AND DESIGN

- Rewrite: Assignment 3 Review, Assignment 4
- Nested Classes
- Arrays and Object References
- OO Design Process
- OO Design Principles

- Questions and Examples
NESTED CLASSES

• Contrast to top level classes in the same file
• Nested classes are always inside the scope of a given class
• No preset limit to the number of nested levels possible
• There are two types of nested classes:
  • Static, nested classes
  • Non-static nested classes, also called “inner classes”

  • For static nested classes, “static” means: instances of the static nested class are not associated with any instances of their enclosing class.
  • Static nested classes are useful when you want a class that is related to its enclosing class but not to a particular instance. This provides a way of grouping related classes.
  • Methods in the enclosing class have access to all static member variables (private or otherwise) of a static nested class and vice-versa. The nested class does NOT have to qualify the varaible name with the enclosing class name.

<Examples>
NON-STATIC NESTED (INNER) CLASSES

• Require an instance of their enclosing class to exist before they can be created
• These have access to their enclosing class’s instance variables.
• Come in three varieties:
  Anonymous
  Method-local scope
  Class-local scope

The syntax for these takes some getting used to.
• Anonymous classes are useful when you need an object in a very specific scope and don’t want the full overhead of writing another class.
• Method and Class-local scoped classes are useful when you want objects that support the enclosing class only.

<Examples>
Arrays may contain object references if declared to do so.

Example:

```java
Object[] someObjects = new Object[10];

But methods cannot yet be invoked:

    someObjects[5].toString();  // Runtime error: NullPointerException

All 30 array positions in someObjects are null.

for (int i=0; i < 10; i++) {
    someObjects[i]=new Object();
    System.out.println("Object[" + i + "] is: " + someObjects[i].toString());
}
```
OBJECT ORIENTED DESIGN PROCESS

• Learn the language of the target domain.
• Find the “abstractions”. In this context it means the members of the target domain that are relevant to doing work that you want to model with software.
• A good way to start is with brainstorming and speaking to subject matter experts.
• Once you have candidate abstractions, reduce the set based on where you think you can make the best start.
• Determine the software artifacts (classes, abstract classes, interfaces, enums, etc.) you will need to implement those abstractions.
• Decide the relationships between those artifacts, as this will determine key pieces of their implementation.
• Decide upon responsibilities of your classes. Write a brief description of each class, then write and test them independently of each other.
• Test them together to ensure the system is working as expected and meeting its requirements.
FIVE KEY OBJECT ORIENTED DESIGN PRINCIPLES

• Single Responsibility Principle
  • Design classes to have one coherent responsibility.

• Open-Closed Principle
  • Design classes to be open for extension, closed for modification.
    • Practically, this means to make code interface-driven and extendable by adding new implementations.

• Interface Segregation Principle
  • Partition large interfaces into smaller ones that are likely to be used or change together.

• Substitution Principle
  • Any subtype can be assigned to a superclass object reference without breaking the program.

• Dependency Inversion Principle
  • Use the most general object reference available (except Object). Do not use subclass references when a superclass reference will work.
THE REST OF THIS CLASS...

Multiple choice questions to highlight key ideas
Concept Examples to highlight our ‘class’ discussions
‘What happens here’ questions

Audience Participation Night...