Inner Classes: Overview

- In Java 1.1.x, a class can be defined within the definition of another class
- This can sometimes enhance encapsulation and make for cleaner designs
- Four different scenarios: nested classes, member classes, local classes, and anonymous classes

Inner Classes: Nested Classes

- Class defined as static within another class
- Cannot use methods or fields from enclosing class
- Organizational convenience
- Can be private
- Cannot themselves contain nested classes or static members

Inner Classes: Member Classes

- Class not defined as static within another class
- Can use methods or fields from enclosing class
- Can be private

Inner Classes: Member Classes(cont)

- Use a member class instead of a nested class when you need to refer to members of the enclosing class
- Use *classname.this.membername* to access enclosing class members when name conflicts exist

Inner Classes: Local Classes

- Class defined within an arbitrary block of code
- Same rules as member classes, except they are not declared with an access modifier (just like other local variables)
- Use *classname.this.membername* to access enclosing class members when name conflicts exist

Inner Classes: Anonymous Classes

- Class defined within an expression
- Exactly like a local class, except it doesn't have a name
- No constructor! (Only the compiler-supplied no-args constructor)
- Appropriate when you only need one instance of a class, and defining the class with a name doesn't clarify your code
- Class must implement a known interface or extend a known class

Inner Classes: Anonymous Classes

- Most often used in awt, where you need to create an event handler for a GUI widget
- Generally, only one instance is ever created, and is only used in that one place
- You'll use inner classes (and especially anonymous classes) next quarter

Design Patterns

Design Patterns

- **Originated in early 1980's or earlier**
- **Gained prominence in 1995**
  - **Design Patterns: Elements of Reusable Object-Oriented Software; Gamma, Helm, Johnson, Vlissides (GoF)**
- **Identify proven solutions to common or recurring problems**
- **Document inter-class relationships and their purposes**

Essential Elements

- Name
  - Identifies a pattern from others
  - Provides a "design vocabulary"
- Problem
  - Design context
  - List of conditions indicating applicability
- Solution
  - "elements that makeup the design, their relationships, responsibilities, and collaborations." [GoF]
  - Does not present implementation details

Essential Elements

- Consequences
  - The trade-offs
    - Flexibility
    - Extensibility
    - Portability

Singleton

- Ensures only a single instance of a class exists
  - All clients use the same object.
- Constructor is private to prevent external instantiation
- Single instance obtained via a static `getInstance` method

Singleton Structure