Objectives

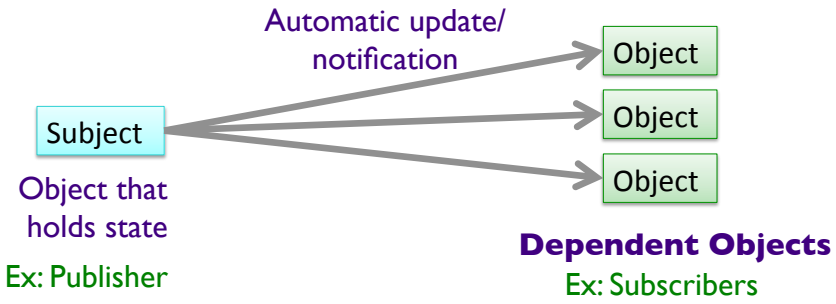
- Design Patterns
  - Observer
  - MVC
- Dependency Inversion Principle
  - Factory design pattern
  - Screensavers

Review

- What is a design pattern?
- What design patterns did we discuss?
  - What design principle(s) does it follow?
- Why do we prefer composition over inheritance?
- Any underlying commonalities?

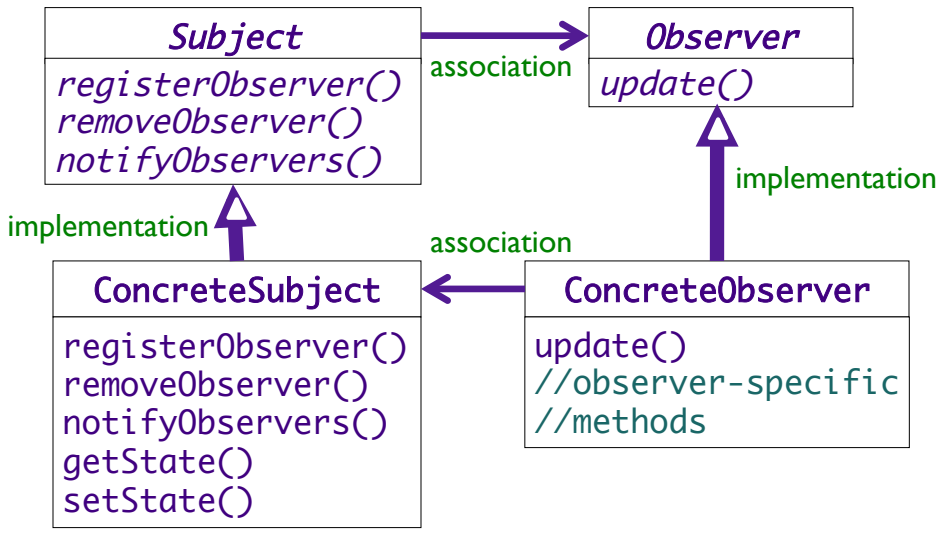
Design Pattern: **Observer**

- Defines a 1-to-many dependency between objects
- When one object changes state, all of its dependents are notified and updated automatically



Observer Pattern

Have we seen this pattern?



Design Principle: **Loose Coupling**

- A principle behind **Observer** pattern

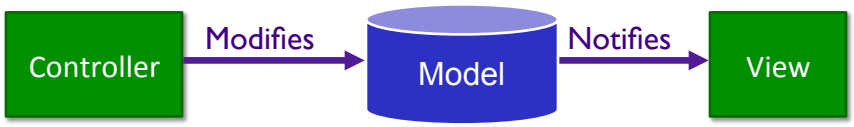
> Goal: loosely coupled designs between objects that interact

- Loosely coupled objects can interact but have very little knowledge of each other
  - Minimize dependency between objects
  - More flexible systems
  - Handle change

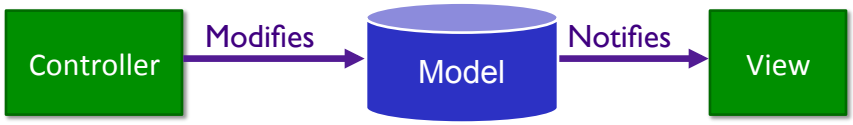
MVC DESIGN PATTERN

Model - Viewer - Controller (MVC)

- A common **design pattern** for GUIs
- Separate
  - Model: application data
  - View: graphical representation
  - Controller: input processing

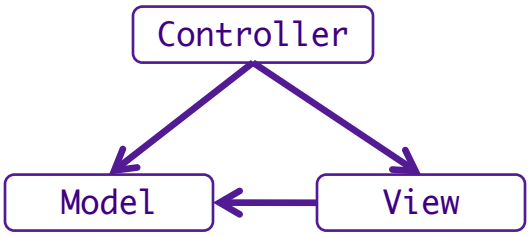


Model-Viewer-Controller

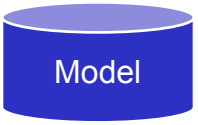


- Can have multiple viewers and controllers
- Goal: modify one component without affecting others

Direct associations

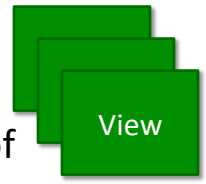


Model

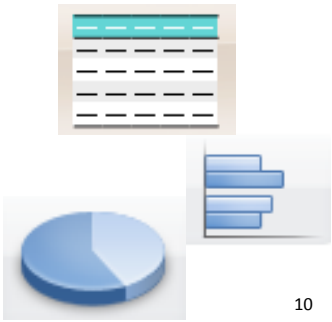


- Code that carries out some task
- Nothing about how view presented to user
- Purely **functional**
- Must be able to register views and notify views of changes

Multiple Views



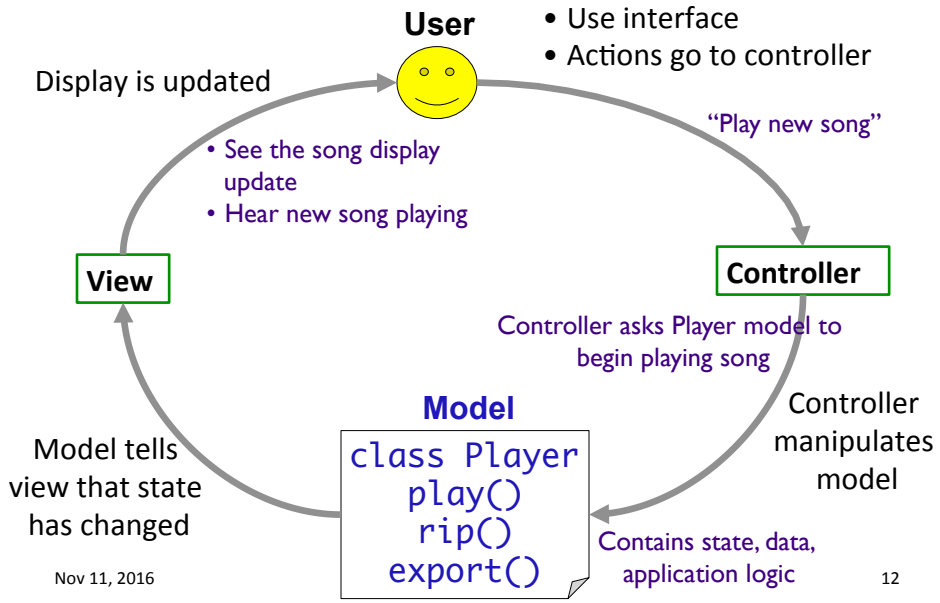
- Provides GUI interface components of model
  - Look & Feel of the application
- User manipulates view
  - Informs **controller** of change
- Example of multiple views: spreadsheet data
  - Rows/columns in spreadsheet
  - Pie chart, bar chart, …



Controller(s)

- Takes user input and figures out what it means to the model
  - Makes decisions about behavior of model based on UI
- Update **model** as user interacts with **view**
  - Calls model's mutator methods
- Views are associated with controllers

Example: Music Player



MVC: Combination of Design Patterns

- Observer
  - Views, Controller notified of Model's state changes
- Strategy
  - View can plug in different controllers
  - Different views of the same model
- Composite
  - View is a composite of GUI components
    - Top-level component learns about model update, updates components
    - A container computes its preferred size by combining all the preferred sizes of its components

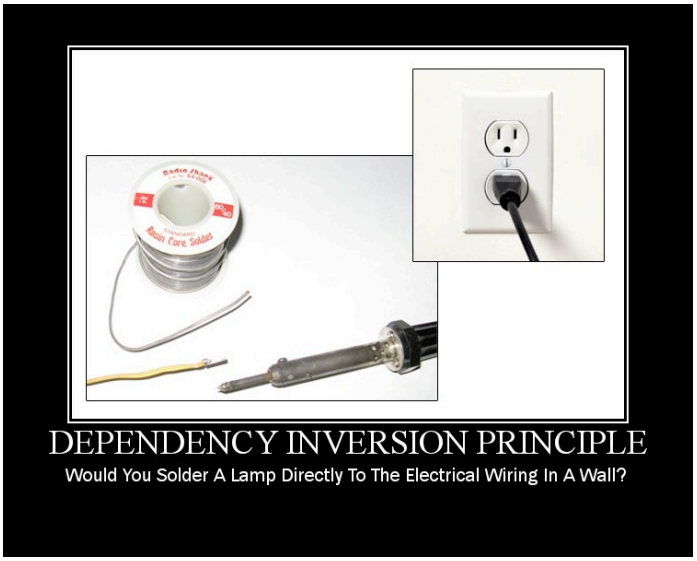
Dependency Inversion Principle

Depend upon abstractions

Dependency Inversion Principle

Depend upon abstractions.
Do not depend upon concrete classes.

- High-level components should not depend on low-level components
  - Both should depend on abstractions
- Abstractions should not depend upon details. Details should depend upon abstractions
- "Inversion" from the way you think



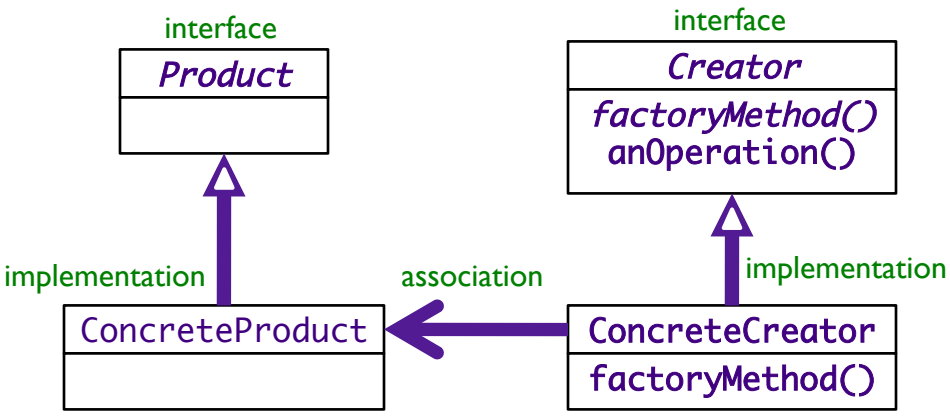
FACTORY DESIGN PATTERN

Design Pattern: **Factory Methods**

- Allows creating objects without specifying exact (concrete) class of created object
- Often used to refer to any method whose main purpose is creating objects

- How it works:
  1. Define a method for creating objects
  2. Child classes override method to specify the derived type of product that will be created

Factory Method Pattern



UML Class Diagram

Guidelines to Follow DIP

- No variable should hold a reference to a concrete class
  - Using new → holding reference to concrete class
  - Use factory instead
- No class should derive from a concrete class
  - Why? Depends on a concrete class
  - Derive from an interface or abstract class instead
- No method should override an implemented method of its base class
  - Base class wasn't an abstraction
  - Those methods are meant to be shared by child classes

What's a problem with following all of these guidelines?

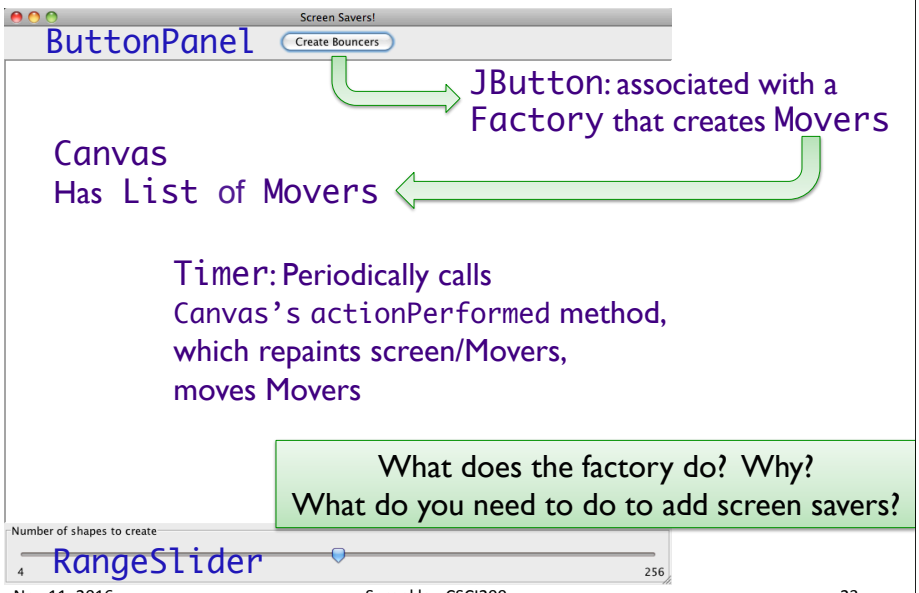
GRAPHICS PROGRAMMING

Understanding Code

Import existing Java project:
`/csdept/local/courses/cs209/handouts/screensavers.tar.gz`

- Simple Bouncers
  - How draws
  - How animates
- Screen Savers
  - What represents an object in the screen saver?
  - How are screen saver objects generated?
  - How is animation handled?
  - How are events handled?

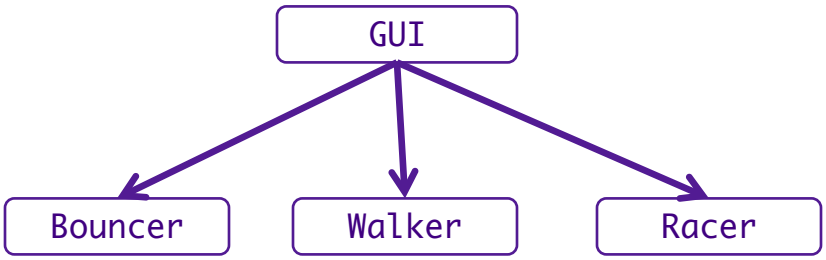
Screensavers GUI/Architecture



Dependency Inversion Principle

- How would you typically build/design the screen saver application?
  - Know we need to view/display a screen saver
    - Buttons, slider, objects that move
    - Top-down
  - Know we need to create a bunch of types of screen savers
    - Abstraction
    - Bottom-up

One Option for Screen Saver Design



Violates Dependency Inversion Principle:
High-level component is dependent on concrete classes.
If implementations change, GUI may have to change

Mapping Factory Design Pattern to Screen Savers

- How does the screen saver application use factory methods?
- What would be the alternative solution?
- What problems are the factories addressing?

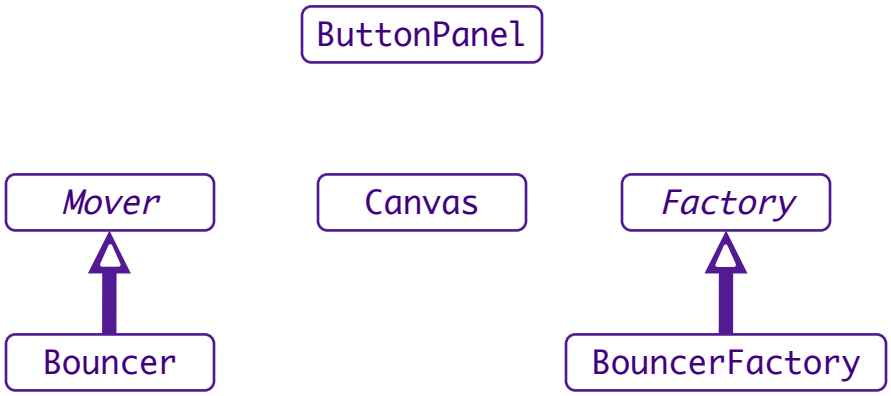
Mapping Factory Design Pattern to Screen Savers

- What problems are the factories addressing?
  - Delegate creation of concrete Movers
    - Likely to change
    - Encapsulate change in factory
  - Using abstraction instead of specifying concrete classes
    - Reduces dependencies to concrete classes

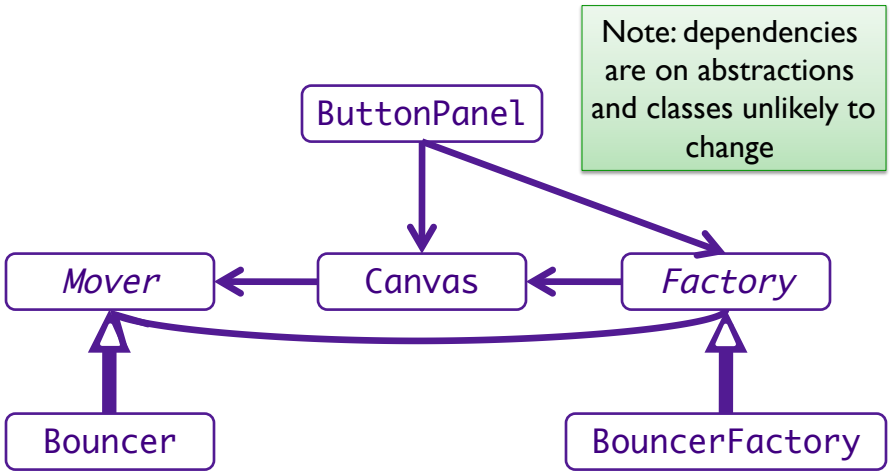
Thoughts

- Didn't need to know design pattern to understand code
  - Helps to know the **terminology** to understand the naming
- Design principles all come down to **where there is change, use abstraction**

Our Screen Saver Dependencies



Our Screen Saver Dependencies



TODO

- Assignment 9 due Wednesday