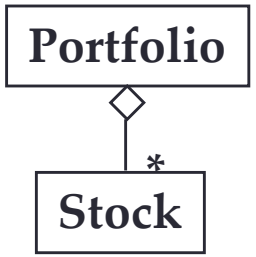
# Model in MVC

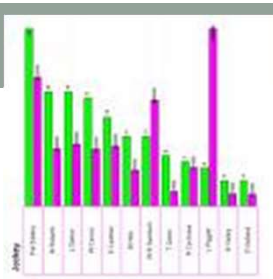
Dr. Youna Jung
Northeastern University
yo.jung@northeastern.edu

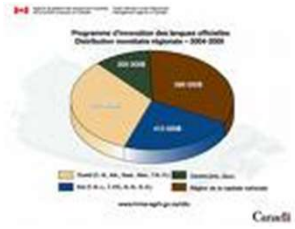
# OBSERVER PATTERNS

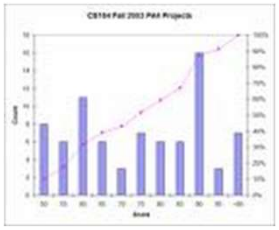

Copyright 2015 Youna Jung

# Motivation







## Problem

- ✓ We have an object that **changes** its state **quite often**
  - Ex) A **Portfolio** of **stocks**
- ✓ We want to **provide multiple views** of the current **state**
  - Ex) **Histogram** view, **pie chart** view, **timeline** view, **alarm**

## Requirements

- ✓ The system should maintain **consistency across** the (redundant) **views**
- ✓ The system design should be **highly extensible**
  - It should be **possible** to **add new views** without having to ~~recompile the observed object or the existing views.~~


Copyright 2015 Youna Jung

# **Observer** Design Pattern

- **Name**
  - ✓ Observer
- **Problem**
  - ✓ Need to notify a changing number of objects that something has changed
- **Solution**
  - ✓ Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically


Copyright 2015 Youna Jung

# Observer Pattern: Examples

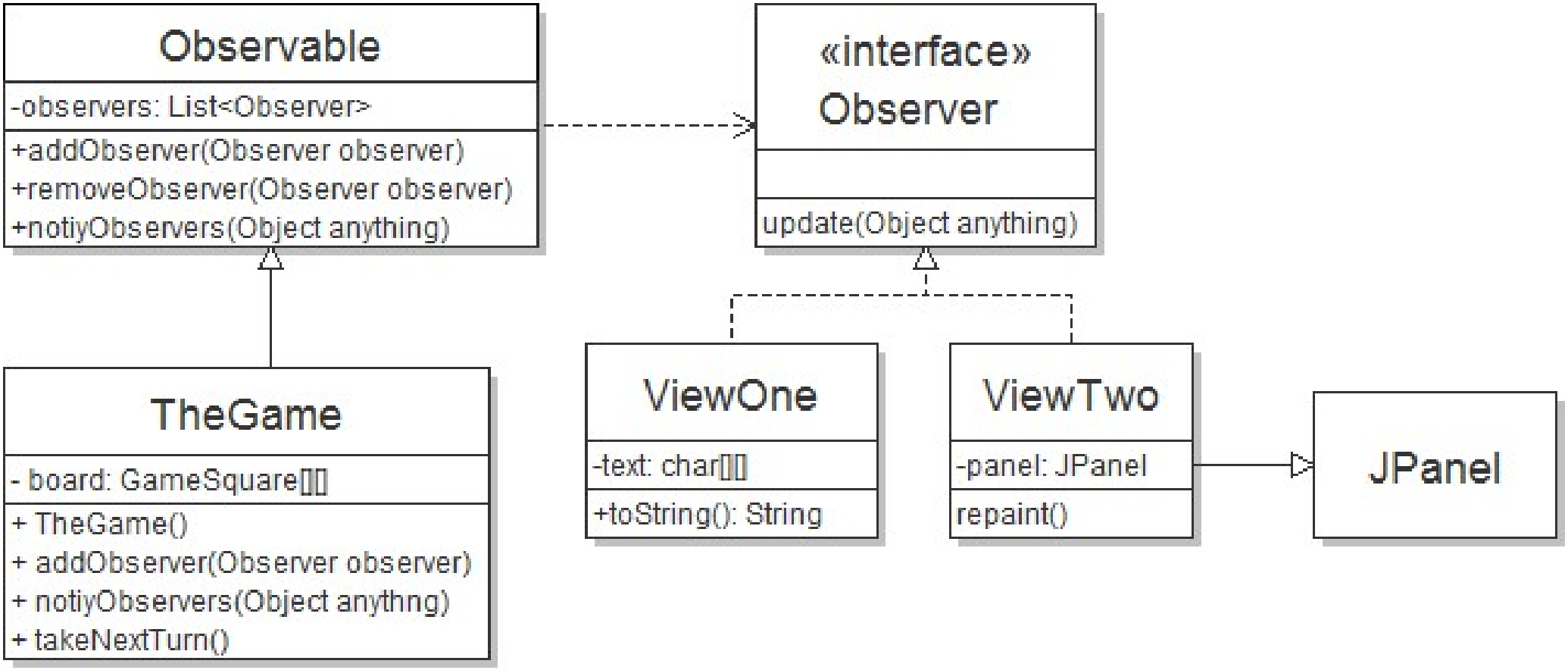
- **Charts** in Spreadsheet
  - ✓ Data frequently changing (Model)
  - ✓ Draw two charts (Observers = two Views)

- **File Explorer**
  - ✓ File system (Model)
  - ✓ File explorer = Observers (View)



Copyright 2015 Youna Jung

# Observer Example



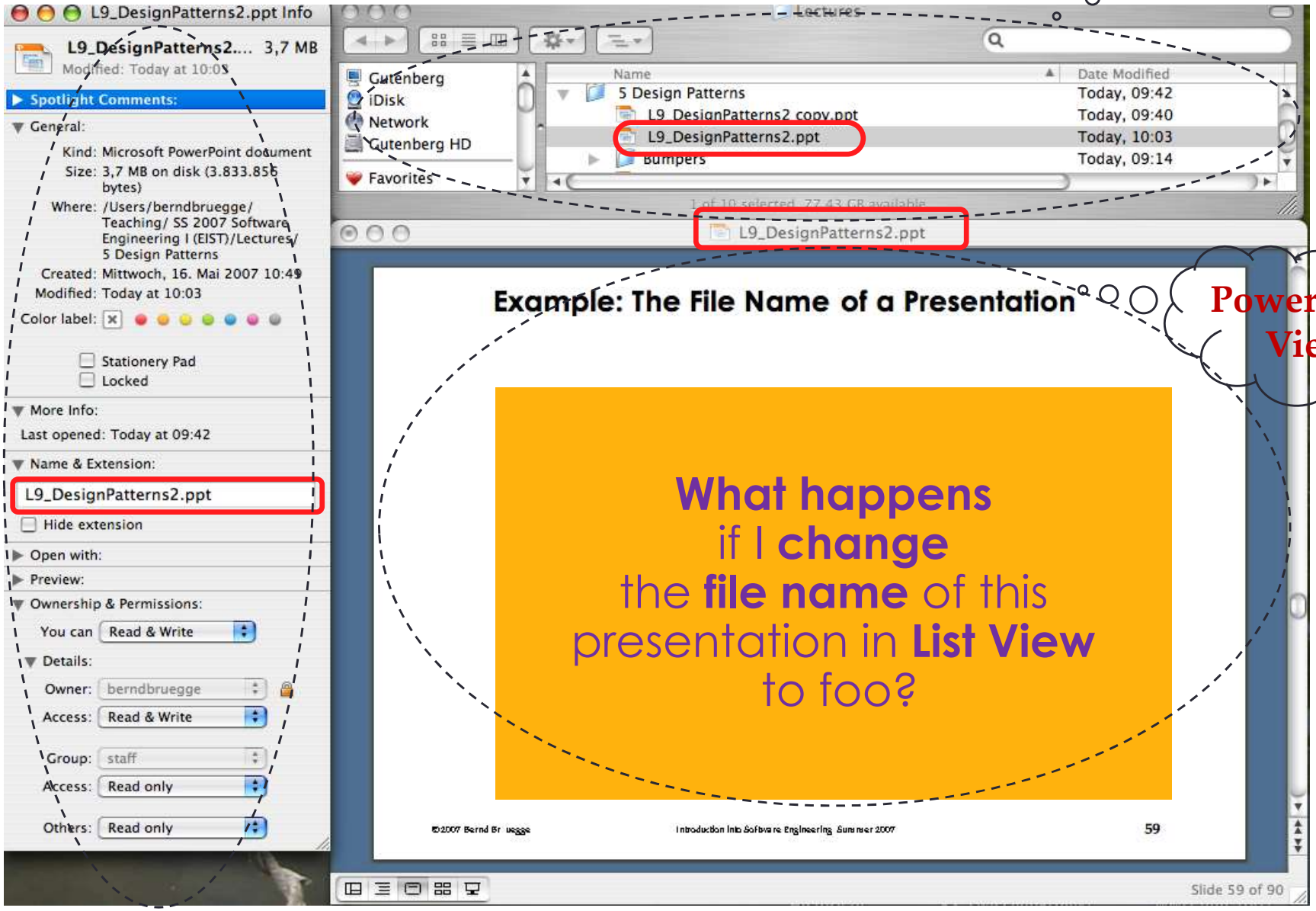


Copyright 2015 Youna Jung

# Example: The **File Name** of a Presentation

List View

***3 Possibilities to change the File Name***

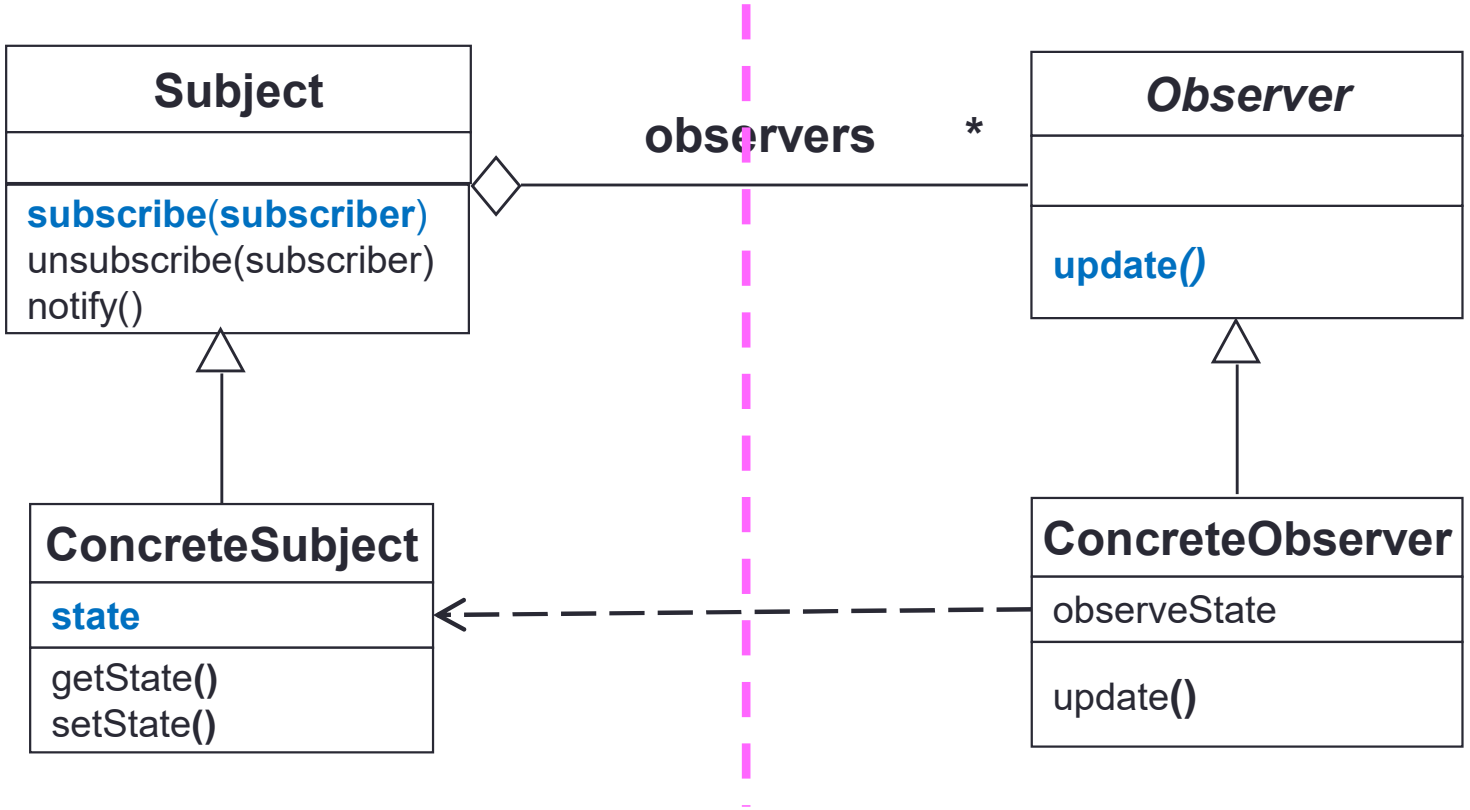



InfoView

Powerpoint View


Copyright 2015 Youna Jung

# Observer Pattern: Decouples from its Views




- **Subject** ("**Publisher**") represents the **entity object**
- **Observers** ("**Subscribers**") **attach** to the **Subject** by calling **subscribe**()
  - ✓ Each **Observer** has a **different view** of the **state** of the **entity object**
    - The state is contained in the subclass **ConcreteSubject**
    - The state can be obtained and set by subclasses of type **ConcreteObserver**.


Copyright 2015 Youna Jung

# **Observer** Pattern (Publish and Subscribe)

- Models a **1:N dependency** between **objects**
  - ✓ **Connects** the state of an **observed object** (the **subject** with many observing objects) the **observers**
- Usage
  - ✓ Maintaining **consistency across** redundant **states**
  - ✓ **Optimizing** a batch of changes to **maintain consistency**
- 3 Ways to maintain the consistency
  - ✓ **Push** Notification
    - – **Every time** the **state** of the **subject changes**, **ALL** the **observers** are **notified** of the **change**
  - ✓ **Push-Update** Notification
    - – The **subject** also **sends** the **state** that **has been changed** to the observers
  - ✓ **Pull** Notification
    - – An **observer inquires** about the **state** the of the subject

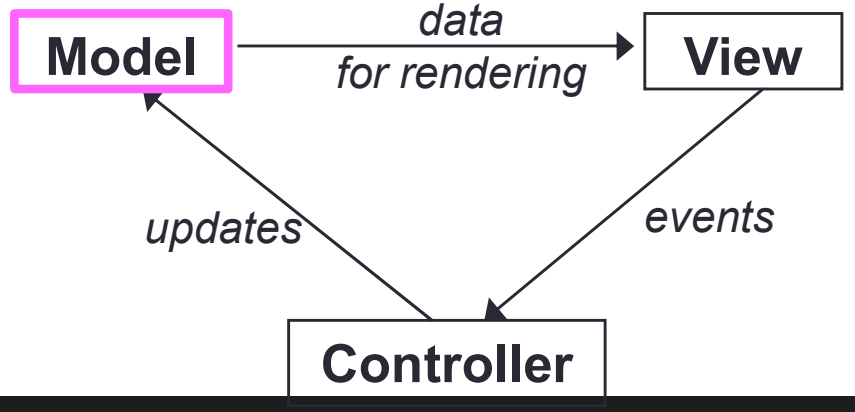

Copyright 2015 Youna Jung

# OBSERVER & MVC


Copyright 2015 Youna Jung

# Model-View-Controller Pattern

- **Model**
  - ✓ **Classes** in your system that are related to the internal representation of **data** and **state** of the **system**
    - often part of the model is connected to **file**(s) or **database**(s)
    - Ex) **Card** game - **Card**, **Deck**, **Player**
    - EX) **Bank** system - **Account**, **User**, **UserList**
  - ✓ What it does
    - implements all the **functionality**
  - ✓ Does not do
    - does not care about **which functionality** is used **when**, **how results** are **shown** to the user

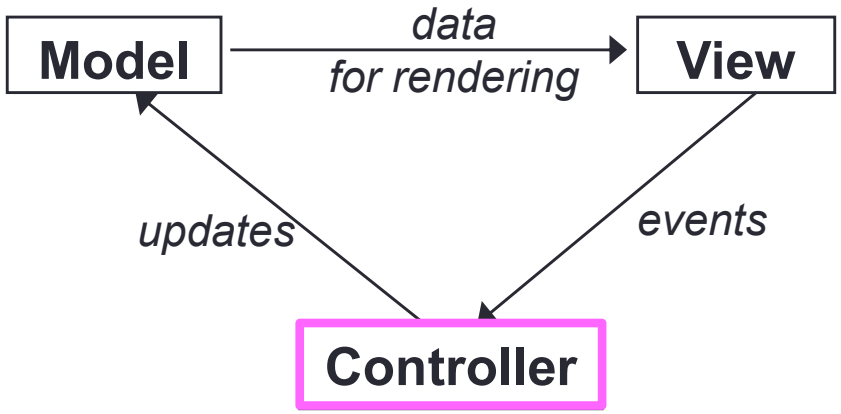




Copyright 2015 Youna Jung

# MVC Pattern

- **Controller**
  - ✓ Classes that connect **model** and **view**
    - defines how user interface reacts to user input (events)
    - receives messages from **view** (where events come from)
    - sends messages to **model** (tells what data to display)
  - ✓ What it does
    - **Takes** user inputs, tells **model** what to **do** and **view** what to **display**
  - ✓ Does not do
    - does **not care how model implements** functionality, screen layout to **display results**

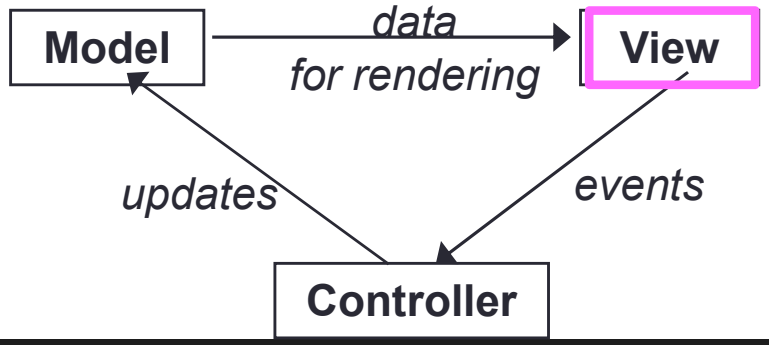




Copyright 2015 Youna Jung

# MVC Pattern

- **View**
  - ✓ Classes in your system that **display** the state of the model to the user
    - generally, this is your **GUI** (could also be a **text UI**)
    - should not contain crucial application data
    - **Different views** can represent the same data in different ways
      - Ex) Bar chart vs. pie chart
  - ✓ What it does
    - **display** results to user
  - ✓ Does not do
    - does not care **how** the **results** were **produced**, **when** to **respond** to user action





Copyright 2015 Youna Jung

# Observer Pattern

- **Observer**
  - ✓ An object that "**watches**" the **state** of **another object** and takes **action when** the state **changes** in some way
    - Ex) In Java: **event listeners** or **`java.util.Observer`**

- **`observable` object**
  - ✓ An **object** that allows **observers** to **examine** it and **notifies** the **observers when** it **changes**
    - permits customizable, extensible event-based behavior for data modeling and graphics

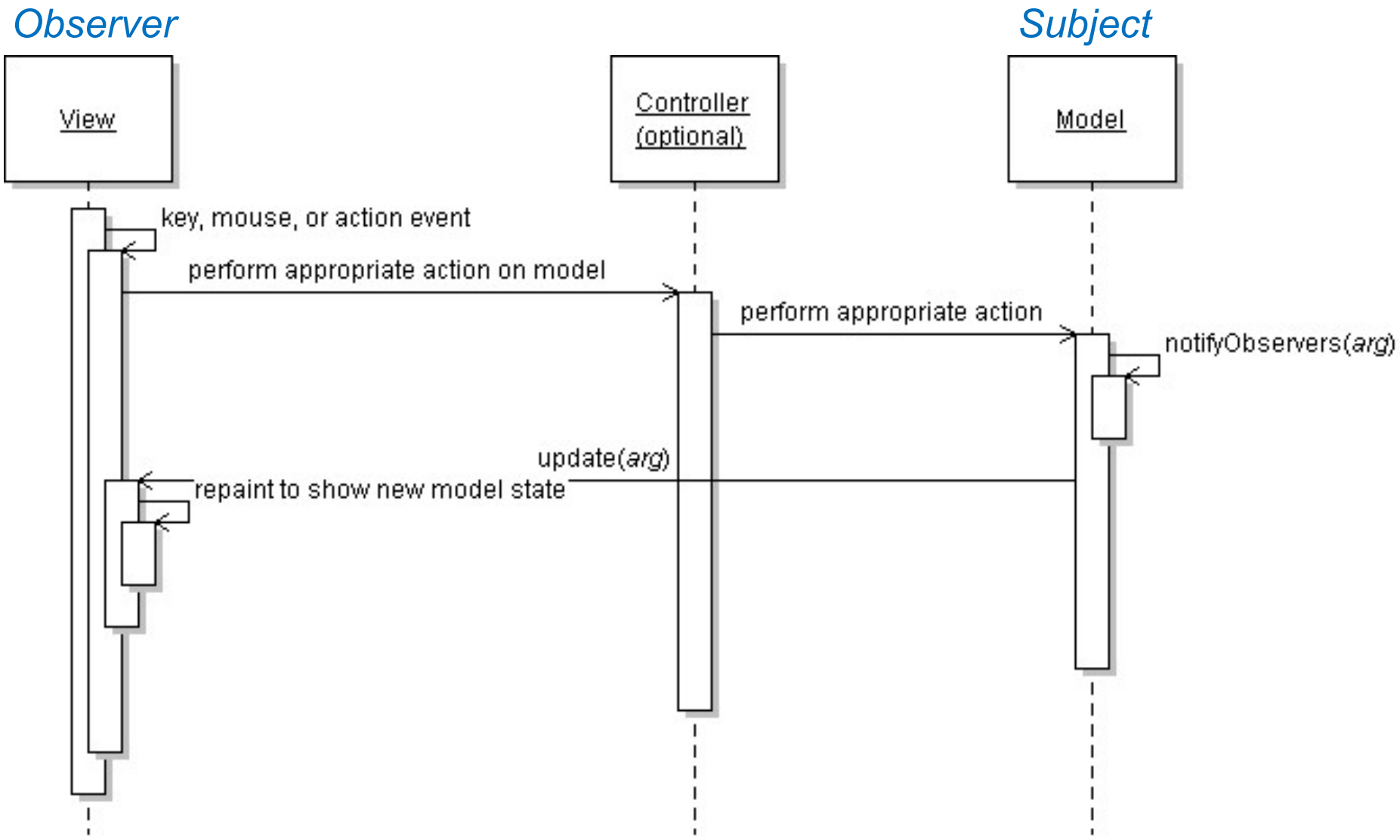

Copyright 2015 Youna Jung

# Benefits of Observer

- **Abstract coupling** between subject and observer
  - ✓ Each can be extended and reused individually
- **Dynamic relationship** between subject and observer
  - ✓ can be established at run time → gives a lot more programming flexibility
- **Broadcast** communication
  - ✓ **Notification** is **broadcast** automatically to **all** interested objects that subscribed to it
- **Observer** can be used to implement **model-view separation** in Java more easily


Copyright 2015 Youna Jung

# Observer - Sequence diagram





Copyright 2015 Youna Jung

# *Observer* Interface

- The **`update()`** method will be called **when** the **observable model changes**
  - ✓ Need to put the appropriate code to handle the change inside **`update()`**

```
package java.util;

public interface Observer {
   public void update(Observable o, Object arg);
}
```


Copyright 2015 Youna Jung

# `Observable` class

- **`public void addObserver(Observer o)`**
- **`public void deleteObserver(Observer o)`**
  - ✓ Adds/removes **`o`** to/from the **list** of **Observer** objects that will be **notified** (via their update method) when **`notifyObservers()`** is called.

- **`public void notifyObservers()`**
- **`public void notifyObservers(Object arg)`**
  - ✓ **Inform all observers** listening to this `Observable` object of an event that has occurred.
  - ✓ `Object` **`arg`**
    - An **optional** argument may be passed to provide more information about the **event**.

- **`public void setChanged()`**
  - ✓ **Flags** the `observable` **object** as having **changed** since the last event; must be called each time before calling **`notifyObservers()`** .


Copyright 2015 Youna Jung

# Common Usage of Observer

1. write a **model** class that **extends** `Observable`
   - ✓ have the model notify its observers when anything significant happens

2. make **all views** of that model into **observers**
   - ✓ Model like GUI panels that draw the model on screen
   - ✓ have the panels take action when the model notifies them of events (e.g. `repaint`, play sound, show option dialog, etc.)


Copyright 2015 Youna Jung

# Using **Multiple Views**

- make an **`Observable` model**
- write a **View** interface or abstract class
  - ✓ make View an observer
- extend/implement **View** for **all actual** views
  - ✓ give each its own unique inner components and code to draw the model's state in its own way
- provide mechanism in GUI to **set view**
  - ✓ perhaps through **menus**

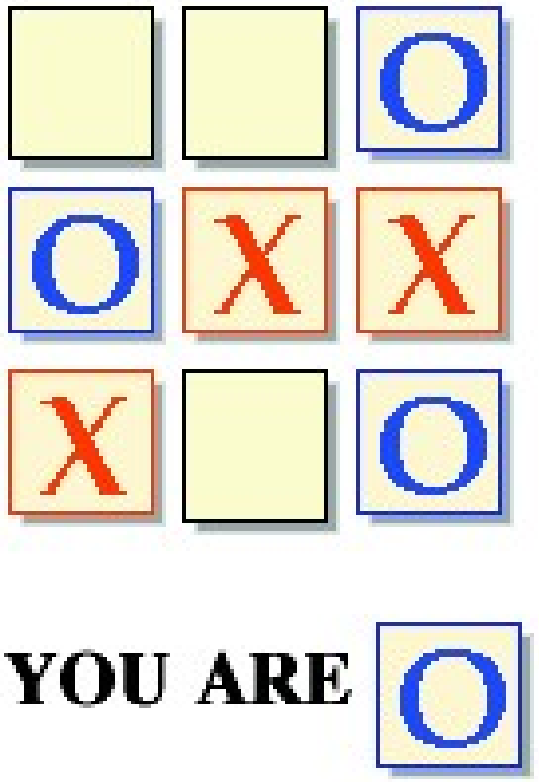

Copyright 2015 Youna Jung

# TIC-TAC-TOE


Copyright 2015 Youna Jung

# Tic-Tac-Toe Game

- Let us first play the game!
  - ✓ https://boulter.com/ttt/index.cgi

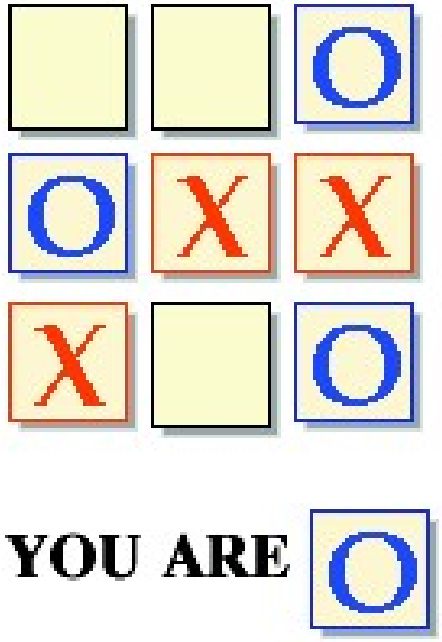




Copyright 2015 Youna Jung

# Rules

- The game is played on a 3 X 3 grid.
- 2 players, **X** and **O**.
- Players take turns putting their marks in empty squares.

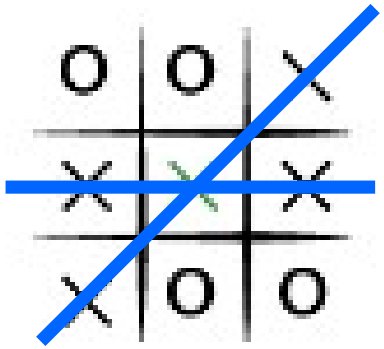


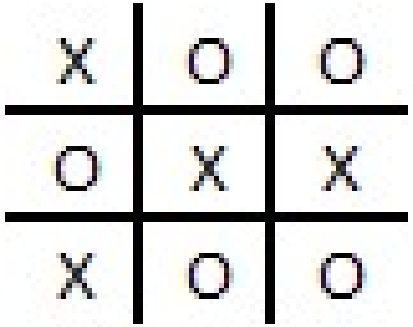


Copyright 2015 Youna Jung

# Win, Lose, or Tie?

- How to **win**?
  - ✓ The first player to get 3 of her marks in a row is the winner.
    - Vertically, horizontally, or diagonally
- **End** of the game?
  - ✓ When a player wins
  - ✓ When all 9 squares are full
- Possible to **tie**?
  - ✓ If no player has 3 marks in a row, the game ends in a tie.

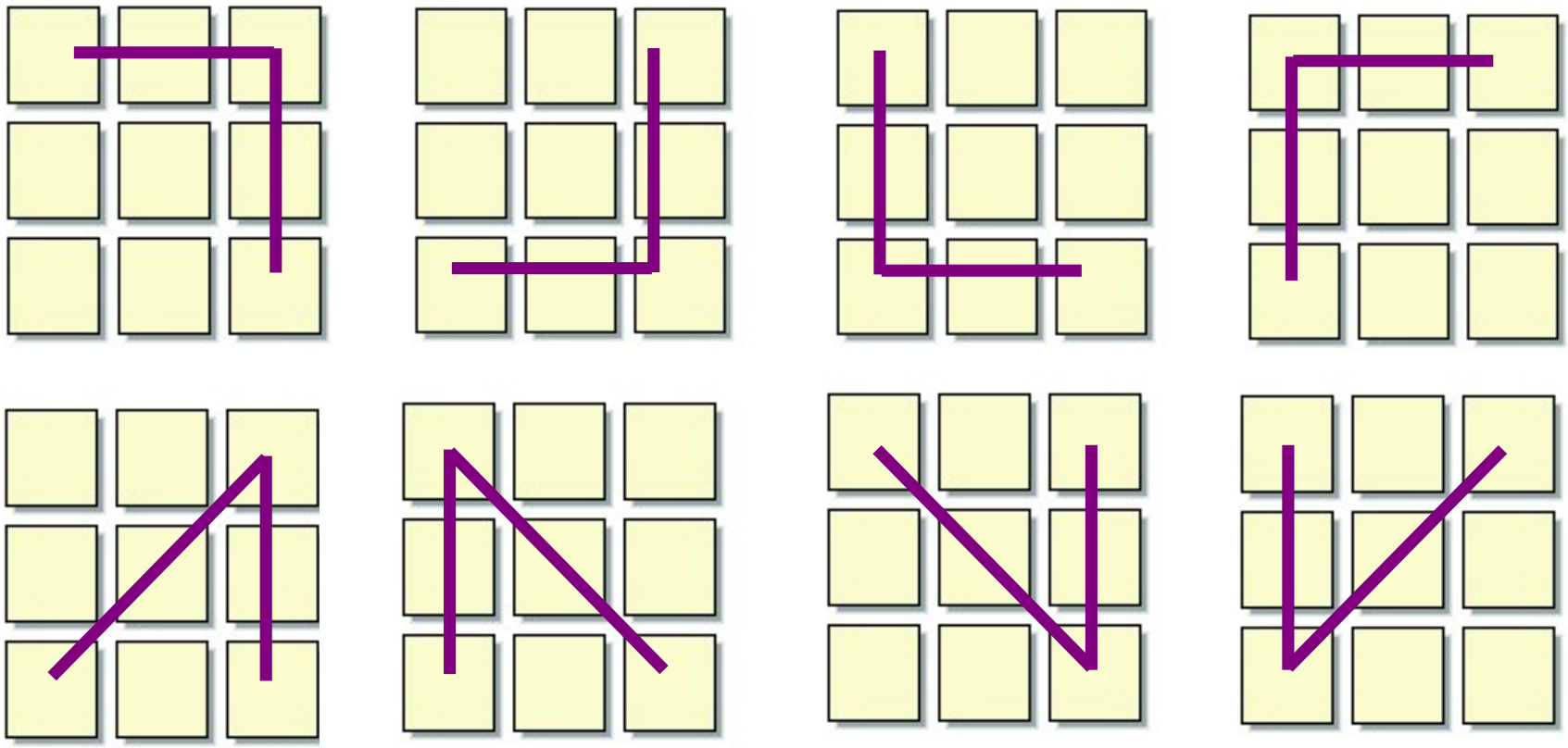




Copyright 2015 Youna Jung

# Winning Strategy for Tic-Tac-Toe





Copyright 2015 Youna Jung

# Tic-Tac-Toe: **Model**

- **Model** in MVC
  - ✓ Creates and maintains **data**
  - ✓ implements all the **functionality**

- To design the model
  - ✓ specify what **functionality** has to be offered by the program
  - ✓ distill that information into a set of operations
    - – which can be divided into one or more classes


Copyright 2015 Youna Jung

# Tic-Tac-Toe: **Model**

- Expected **Functionality**
  - ✓ Play a **move** as X or O.
  - ✓ **Find** out whose **turn** it is.
  - ✓ **Find** out the **contents** of the grid, perhaps in order to **display** it.
  - ✓ **Find** out **whether** the game is **over**, and if so, **who** the **winner** is, if any.

**Define Interface, Classes, and Methods**

- For the model to enforce the rules of the game, it should **signal an exception** in these cases
  - ✓ Attempting to play **out of turn**.
  - ✓ Attempting to **move** on an **occupied cell**.
  - ✓ Attempting to **play** after the **game** is **over**.
  - ✓ Attempting to **play** in a **cell** that **doesn't exist**.


Copyright 2015 Youna Jung

# How to move?

- **`moveAsX()`** and **`moveAsO()`** ? ➔ **`move()`** and update a turn

```
/**
 * Places an X or O mark in the specified cell. Whether it places an X
 * or O depends on which player's turn it is.
 *
 * @param column the column of the cell
 * @param row    the row of the cell
 * @throws IllegalStateException if the game is over, or if there is
 * already a mark in the cell.
 * @throws IndexOutOfBoundsException if the cell is out of bounds.
 */
void move(int column, int row);
```

- The model must throw an **exception** if
  - ✓ Attempting to play out of turn, move on an occupied cell, play after the game is over, or play in a cell that doesn't exist ➔ **IllegalArgumentException**, **illegalStateException**


Copyright 2015 Youna Jung

# Whose turn is it?

- Option 1:

```
boolean isXsTurn();

/** [mutatis mutandis] */
boolean isYsTurn();
```

- Option 2:

```
public class TicTacToeModel implements TicTacToe {
  private final Player[][] board;
  private Player turn;
```

```
@Override
public Player getTurn() {
  return turn;
}
```

- Must throw an **IllegalStateException** if a game is already over.


Copyright 2015 Youna Jung

# Getting the Grid?

- Need to offer a way for a client of our model to get the current state of the game
  - ✓ `Player[][] getBoard()` ? ➔ Querying a specific cell

```
/**
 * Returns the {@link Player} whose mark is in the cell at the given
 * coordinates, or {@code null} if that cell is empty.
 *
 * @param column the column of the cell
 * @param row    the row of the cell
 * @return a {@code Player} or {@code null}
 * @throws IndexOutOfBoundsException if the cell is out of bounds.
 */
Player getMarkAt(int column, int row);
```


Copyright 2015 Youna Jung

# Game is over?

- Need some way to find out when the game is over

```
/**
 * Determines whether the game is over.
 *
 * @return whether the game is over
 */
boolean isGameOver();
```


Copyright 2015 Youna Jung

# Who is the winner?

- Need some way to find out who, if anyone, won

```
/**
 * Returns the winner of the game, or {@code null} if the game is a
 * tie.
 *
 * @return the winner or {@code null}
 * @throws IllegalStateException if the game isn't over
 */
Player getWinner();
```

- Return **NULL** if there is **NO winner** (game is a tie)


Copyright 2015 Youna Jung

# Tic-Tac-Toe: **Model**

- Please complete the three java files provided
  - ✓ **TicTacToe.java**
  - ✓ **TicTacToeModel.java**
  - ✓ **TicTacToeModelTest.java**


Copyright 2015 Youna Jung