Object-Oriented Design

CSE 5236: Mobile Application Development
Course Coordinator: Dr. Rajiv Ramnath
Instructor: Adam C. Champion, Ph.D.
Reading: Applying UML and Patterns, Chaps. 1, 6 (OO ref.); Big Nerd Ranch Guide, Chap. 2 (Android/MVC)
Elements of Good Object-Oriented Design

• Idea: Capture the complexity of real world problems and solutions in objects
  – Classes and responsibilities
  – Polymorphism helps represent the real-world
  – Achieve system goals through collaboration

• Principles: Loose coupling, high cohesion
  – Abstraction
  – Encapsulation and information hiding

• Methodology: Scenario-Driven Design
  – Implement objects needed for vertical slices of the system

• Technique: CRC-Card-Based Design
Terminology Check

- Class
- Object
- Method
- Function
- Class method
- Subclass (implementation)
- Subtype (interface)
- Interface
- Abstract class
- Virtual method

You should know what each term means!
OO Design Process

- Capture “narratives” of the environment, application (app)
  - Via observations, conversation with domain experts, other stakeholders (including end users!)
  - Identify specific “circumstances” where an app would be used
  - Capture using text, storyboards, sketches, descriptions
- Identify **domain** classes, assign responsibilities, identify collaborators.
- Evaluate using the OO Checklist
- Develop the application flow: use cases, screen flows
- Map domain model to Android framework model
- Map application flow to framework-specific model. Connect UI to domain model. Identify collaborators, update model and flow
- Add contracts
- Evaluate again using the OO Checklist
Identify Objects and Classes

- Examine nouns, noun phrases as candidates
- Adjectives: candidate attributes or subtypes
- Group into categories: potential abstract classes, super classes
- Write down the purpose of each class
- Eliminate redundant or non-domain classes
Identify and Assign Responsibilities

• Start with verbs and verb phrases (i.e. actions)
• Assign to the appropriate classes
  – Try to distribute evenly: don’t give a class too much work
  – Don’t break the class definition
  – Locate responsibility with information
  – Locate related information together
Map Domain Model to Core Framework Patterns

- Usually a variation of MVC
Identify Collaborations

- Examine narratives, storyboards, use cases
- Create scenarios
- Walkthrough scenarios
- Identify interactions between classes
  - These are the collaborations
Evaluate Using Design Checklist

• Each class must have:
  – Clear name
  – Cohesive description of responsibility
  – Long-lived state
  – Collaborators
Record on CRC Cards (1)

Class Name

- Responsibility 1
- Responsibility 2
- Responsibility 3

Collaborator 1
Collaborator 2

Record on CRC Cards (2)

Tic-Tac-Toe Exercise

• Tic-tac-toe, also spelled tick-tack-toe, or noughts and crosses, as it is known in the UK, Ireland, Australia, New Zealand, is a pencil-and-paper game for two players, who take turns marking the spaces in a $3 \times 3$ grid with the symbols X and O respectively. The X player usually goes first. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

• We extend this below by adding a narrative about playing the game on the Android device:
  – Tic-tac-toe for Android will implement the Tic-tac-toe paper game as an Android app. In it, human users will be able to play Tic-tac-toe against the computer. Multiple games may be played in each session, with either the computer playing first or the human playing first on an electronic board that will be displayed on the device’s touch screen. Scores for each session will be accumulated. If the user quits the session, scores will be reset.
Nouns and Verbs

• Nouns: pencil, paper, game, nought, cross, player, X, O, space, symbol, grid, mark, vertical row, horizontal row, diagonal row, human user, human, computer, session, board, touchscreen, score. These are candidate objects and classes.

• Verbs: take turn, mark, goes, place, win, implement, play, playing first, display, accumulate, quit, reset. These are candidate responsibilities.
Consolidate: Nouns

• Remove *pencil, paper, touchscreen* – physical objects
• *Symbol* and *mark* identical – retain *symbol.*
• *User vs. player* – retain *player*
• Remove one of *board* and *grid*
• Remove *touchscreen* – physical
• *Row* is a component
• *Session* is an instance of *game*
Consolidate: Verbs

- *Take turn, goes, play* – retain *play*
- *Mark vs. place vs. …? Use place symbol*
- *Remove implement* – irrelevant to game
- *Retain display, accumulate, exit and reset*
Candidate Classes and Responsibilities

• Classes: Symbol, Player, Human, Computer, Board, Row, and Game (with attribute Score)
• Instances: 0, X of the class Symbol
• Responsibilities: play, place, display, accumulate (scores), quit, and reset.
Allocate Responsibilities to Classes

• Class Game is allocated the responsibilities: play, accumulateScores, quit, and reset.

• Class Board has Display responsibilities.

• Class GameGrid has Place.

• Symbol, Player, Human, Computer, and Row have no responsibilities yet. Keep?
Map Domain Model to Core Framework Patterns

• Controller classes map to Activities, e.g. GameSession
• Visual elements (if any, remember we’re doing domain object design) map to views
• Pure domain objects map to “plain old Java object” (POJO) hierarchies
General Scenario

• Start a new game.
• Decide and set who plays first: the human or the computer. The first player gets assigned the X as a symbol, while the second player gets assigned the O.
• The first player places his (or its) symbol at an empty location on the board. The second player does likewise. Repeat until one player has three of his (or its) symbols in a row, column or diagonal, or there are no more squares to play, in which case the game ends in draw.
• Accumulate the scores for the players. The winning player gets 1 added to his/its score. No change is made to a losing player’s score. Also, both players’ scores stay the same if the game ends in a draw.
• If the user wishes, start a new game, else, quit.
Screens and Screen Flows in Tic-Tac-Toe
Scenario Walkthrough – Verification, Identifying Collaborators

• No class to respond starting a new game. Create one:
  – GameController?
  – GameController and Game collaborate
• Symbol creation and placement? Symbol and Board.
• placeSymbol invokes Play?
• Game needs checkResult?
• Board, GameGrid and Game are collaborators.
• Etc.
Final Classes and Responsibilities

• **Game**: Represents a single Tic-Tac-Toe game.
  – Responsibilities: play, checkResult
  – Collaborators: GameSession, GameView, Grid.

• **GameView**: Represents the visual display of a Tic-Tac-Toe game.
  – Responsibilities: placeSymbol, showScores
  – Collaborators: Game

• **GameGrid**: Represents a 3 × 3 Tic-Tac-Toe grid.
  – Responsibilities: placeSymbol, getEmptySquares
  – Collaborators: Game

• **GameSession**: Represents a Tic-Tac-Toe play session with multiple games.
  – Responsibilities: playNewGame, quit, decidePlayers, accumulateScores
  – Collaborators: Game, GameView

• **Symbol** – represents a Tic-Tac-Toe symbol (i.e., an X or an O)
  – Responsibilities: None
  – Collaborators: Game
Contracts

- **Game:**
  - play(Grid, Symbol, x, y) returns Success, Failure
  - checkResultAndSetState(Grid) returns nothing
  - isActive() returns true or false
  - isWon() returns true or false
  - isDrawn() returns true or false

- **GameView:**
  - placeSymbol(Symbol, X, Y) returns Success, Failure
  - showScores(PlayerOneScore, PlayerTwoScore) returns nothing.
Thank You

Questions and comments?