Object Oriented Design and Analysis
CPE 372

Lecture 9

Evaluating OO Designs

Dr. Sally E. Goldin
Department of Computer Engineering
King Mongkut's University of Technology Thonburi
Bangkok, Thailand

Why create a design?

Explore the system in a simpler form than in code

Create a plan to make implementation faster and easier

Discover and solve problems at an early stage

Design is *always* iterative

You never get it right the first time!

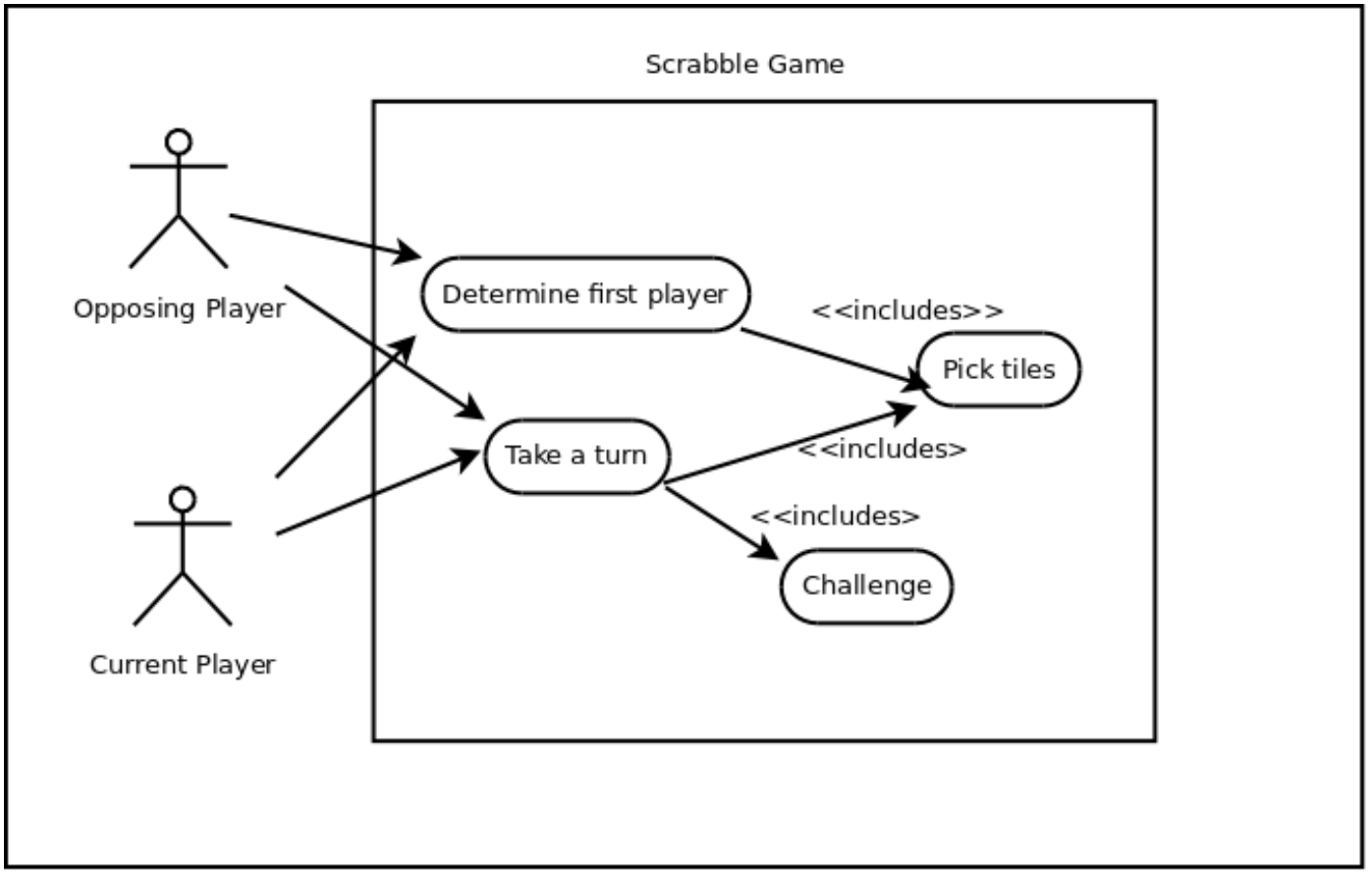
How do we know our design is “right”?

High Level Goals

Manage (reduce, control, understand) complexity

Respond to change (people, requirements) over time

Evaluating Functional Designs

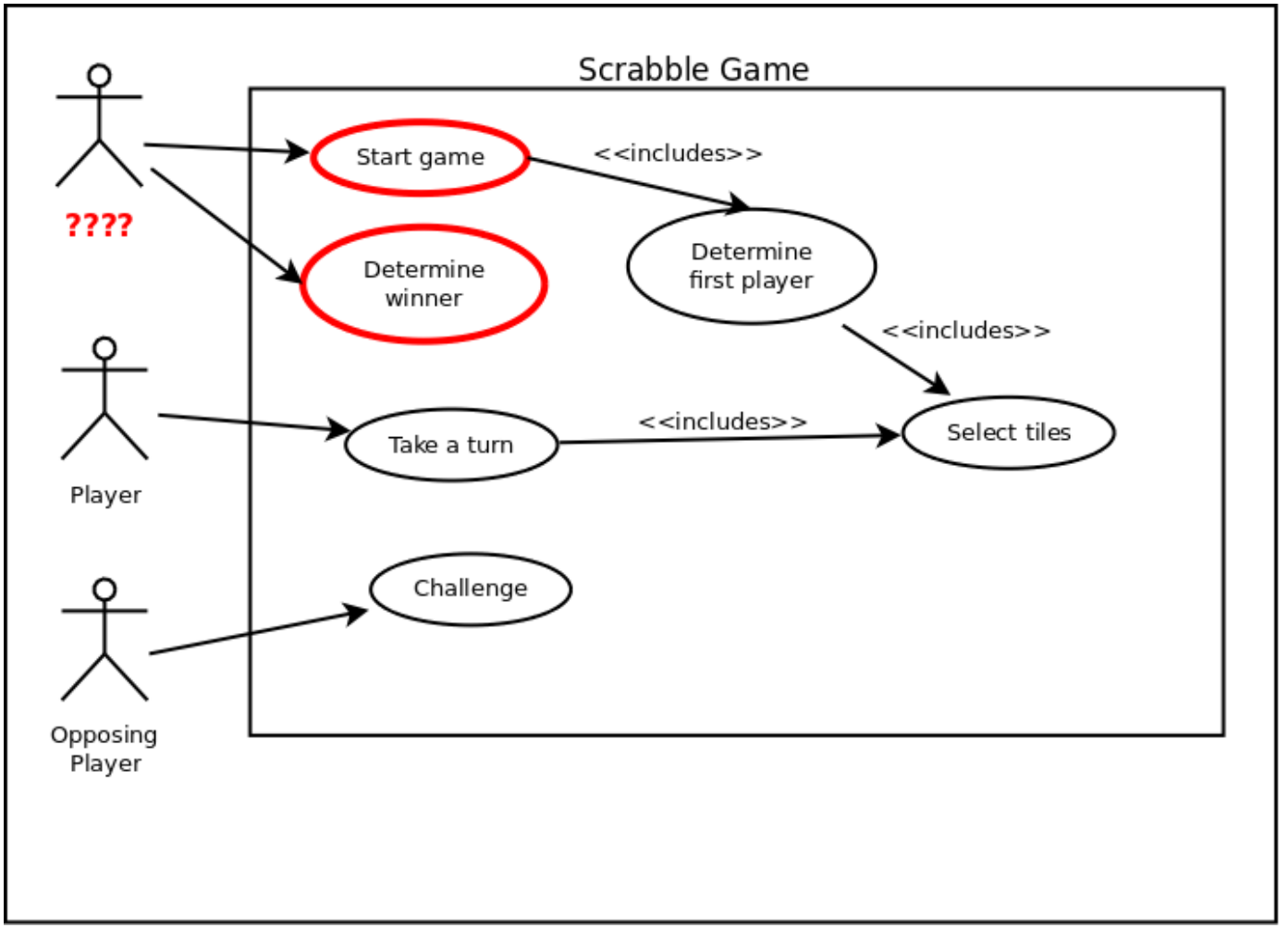


Complete?

Correct?

Sufficiently detailed and expressive?

Revised Use Case Diagram

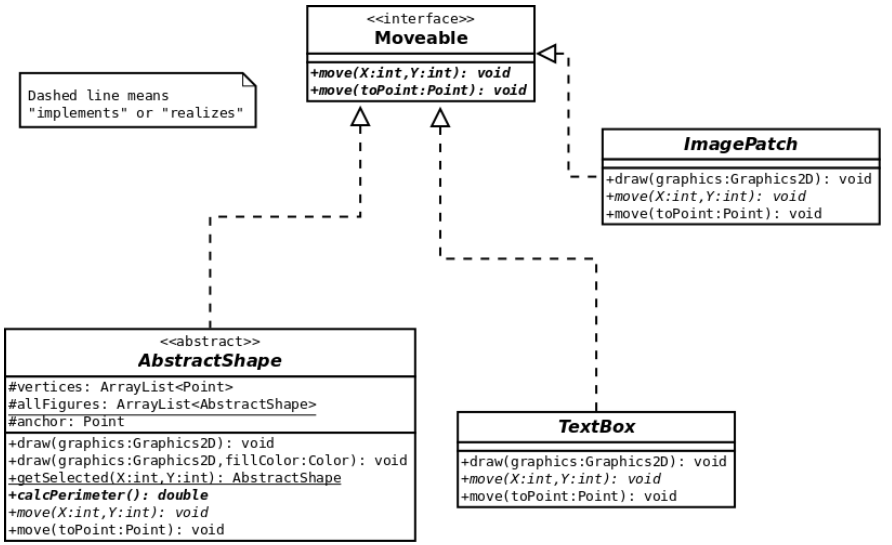


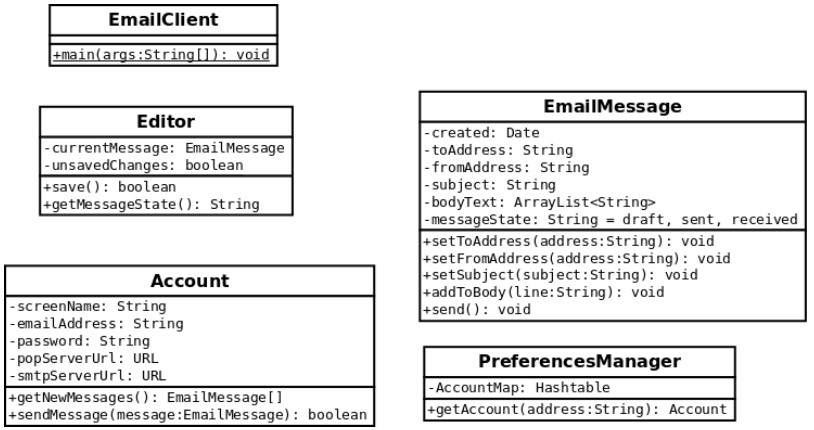
What actor starts the game? Determines the winner?

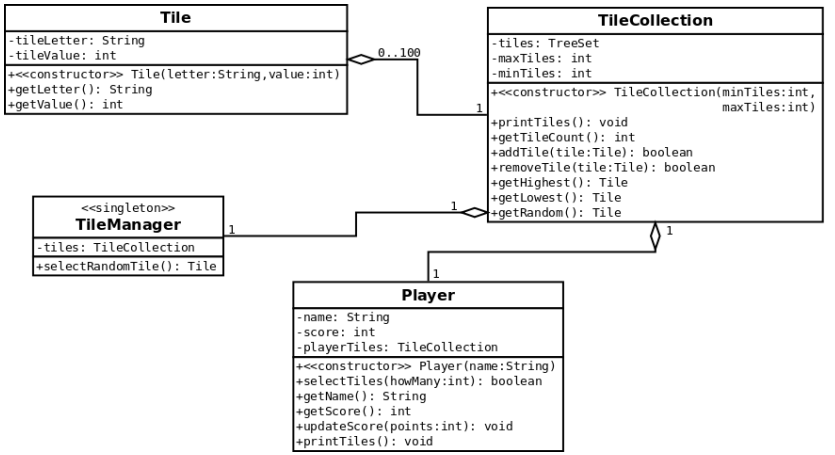
Evaluating Class Structure

We want our classes to have

- Low coupling
- High cohesion
- Sufficiency
- Primitiveness

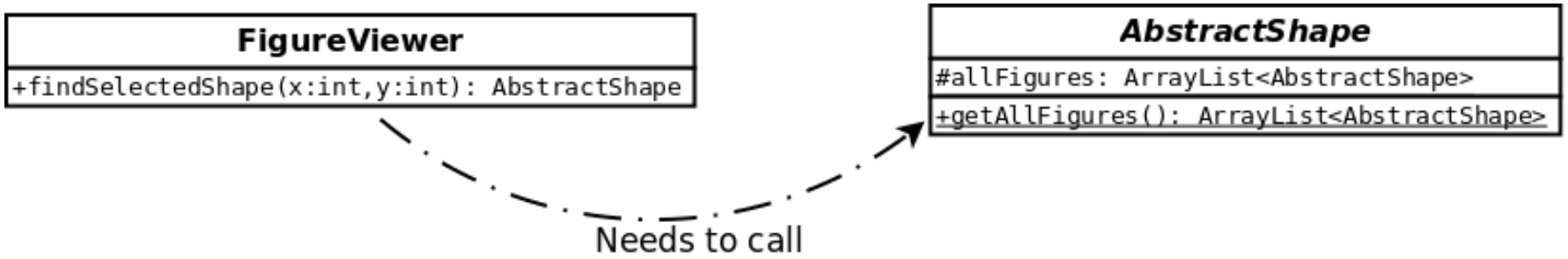






Reducing Coupling (1)

The methods in one class should not depend on the structure of another class

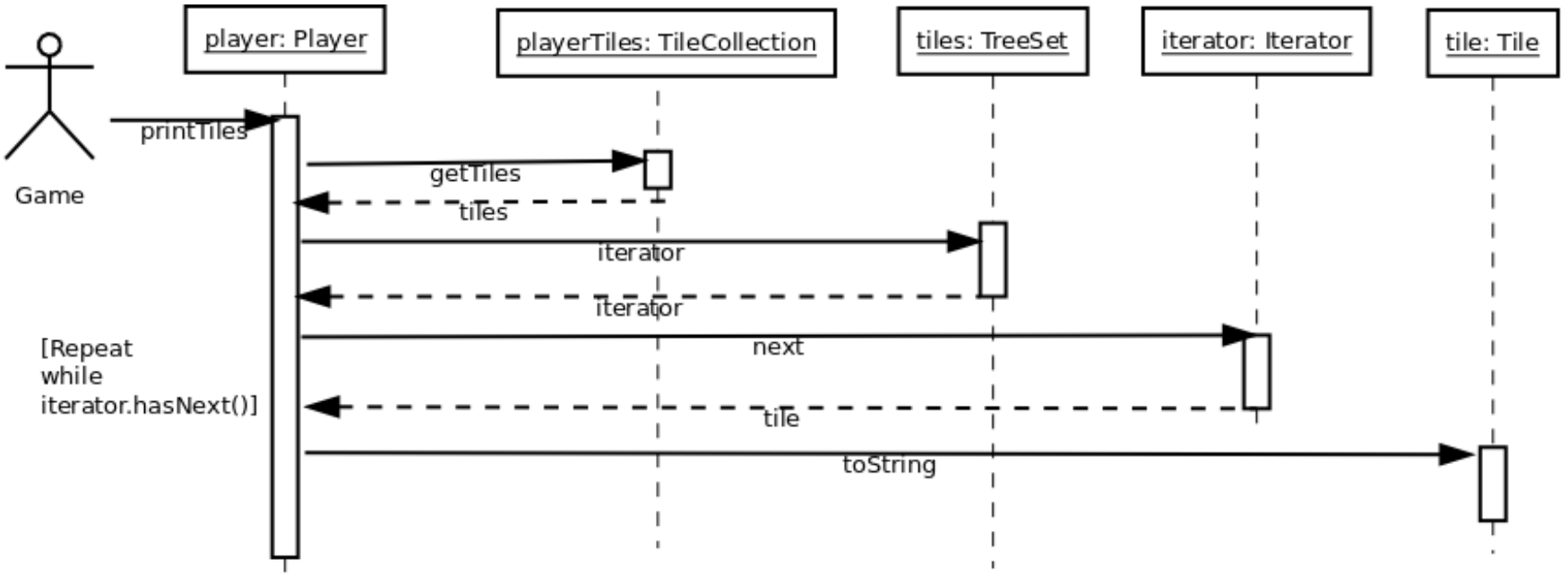


In Exercise 5, some people put the findSelectedShape() method in the ***FigureViewer*** class.

This created undesirable coupling between the two classes since we couldn't change the data structure in ***AbstractShape*** without also changing ***FigureViewer***

Reducing Coupling (2)

A class should communicate with as few other classes as possible.



Sequence diagram from last week, before using delegation.
Player class is communicating with many other classes.
This is not a good design.

Increasing Cohesion

Each class should do one thing, and do it well.

You should be able to summarize the task or purpose of any class in a single sentence!

Common Problem: The “God” Class

Classes have a tendency to acquire more functionality as you work on the design.

Watch out for “bloated” classes that have too many methods, too much power.

This sort of class is also called a “Swiss Army Knife class” because it tries to do everything!

Example:
Expansion of the Game Facade Class

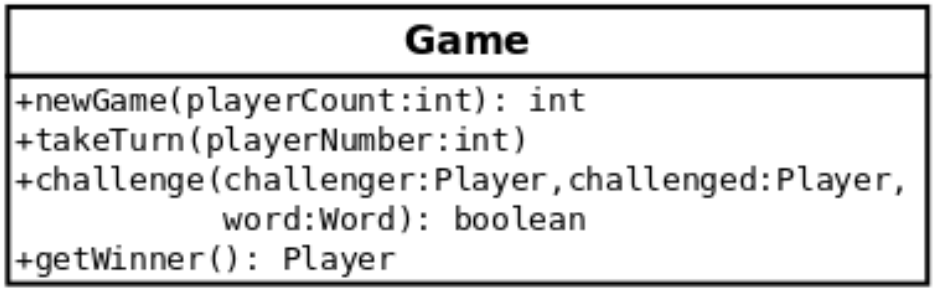
Suppose we decide we want to distribute our Scrabble game across different network nodes. Different players can be on different remote computers.

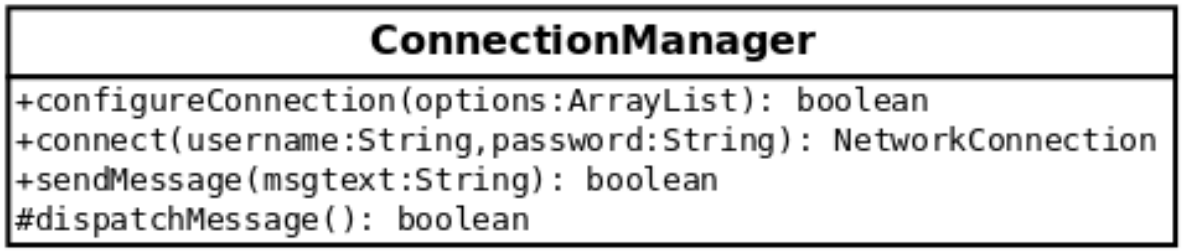
We need new methods to handle network communication.

It may seem natural to add these to our facade class ***Game***.

Game
+newGame(playerCount:int): int +showBoard() +takeTurn(playerNumber:int) +showPlayerTiles(playerNumber:int) +challenge(challenger:Player,challenged:Player, word:Word): boolean +getWinner(): Player +configureConnection(options:ArrayList): boolean +connect(username:String,password:String): NetworkConnection +sendMessage(msgtext:String): boolean #dispatchMessage(): boolean

But this greatly reduces cohesion!





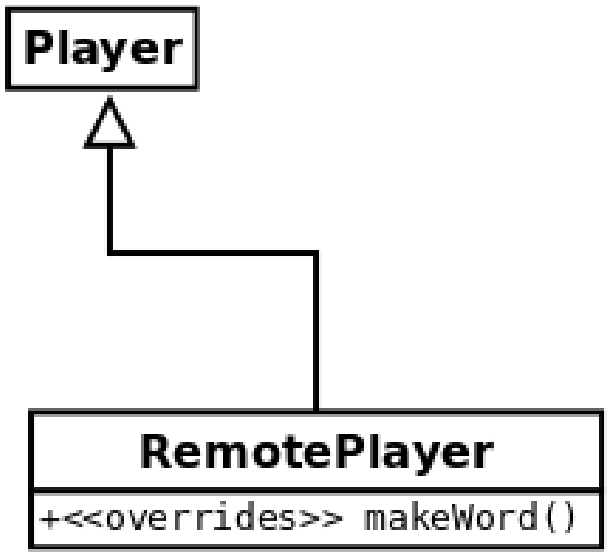
Primary task of Game is to manage game play.

Break out the new functionality into a new class.

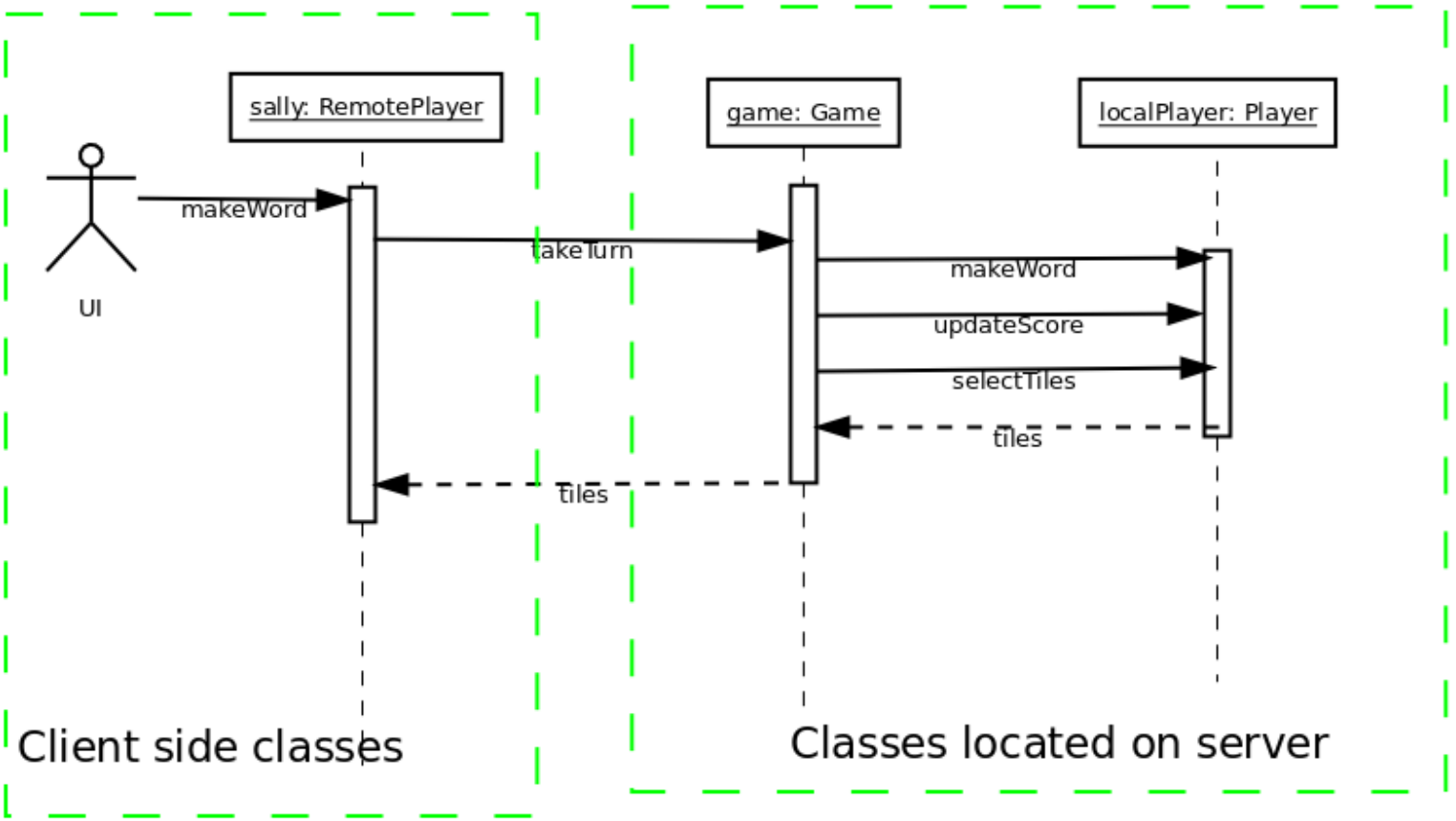
In fact some previously defined methods related to displaying information also make more sense in a new, separated class.

Duplicate Code Reduces Cohesion

In building networked Scrabble, we might decide to create a subclass of ***Player***, to run on the remote computer.



Not a good idea!



Both players have ***TileCollections***. How can we guarantee their contents are the same?
Making a word should be the responsibility of one or the other.

Sufficiency

What's missing?

Do you have all the classes, data and methods you need?

Hard to tell!

Creating behavioral UML diagrams (sequence, state diagrams) can help identify gaps in your design.

Primitiveness

A class is considered “primitive” if all its public methods access the class data directly

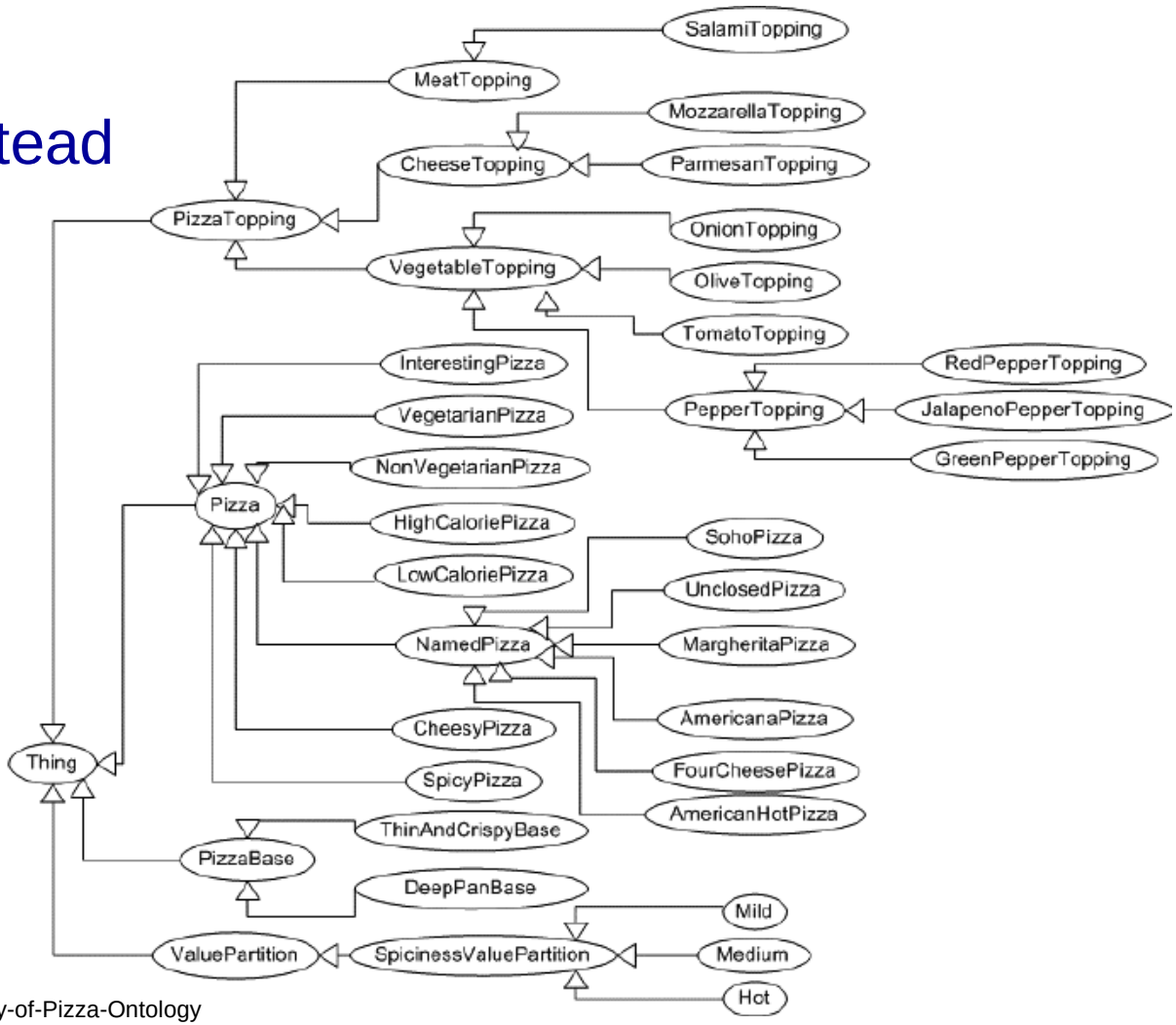
This tends to lead to simpler, cleaner designs than when class methods call lots of other public class methods.

Relationships between Classes

Use inheritance to represent true “is-a” relationships

Otherwise, choose association/composition instead of inheritance

Beware of very deep class hierarchies



https://www.researchgate.net/figure/228408042_fig1_Fig-2-Class-hierarchy-of-Pizza-Ontology

Separation of Concerns

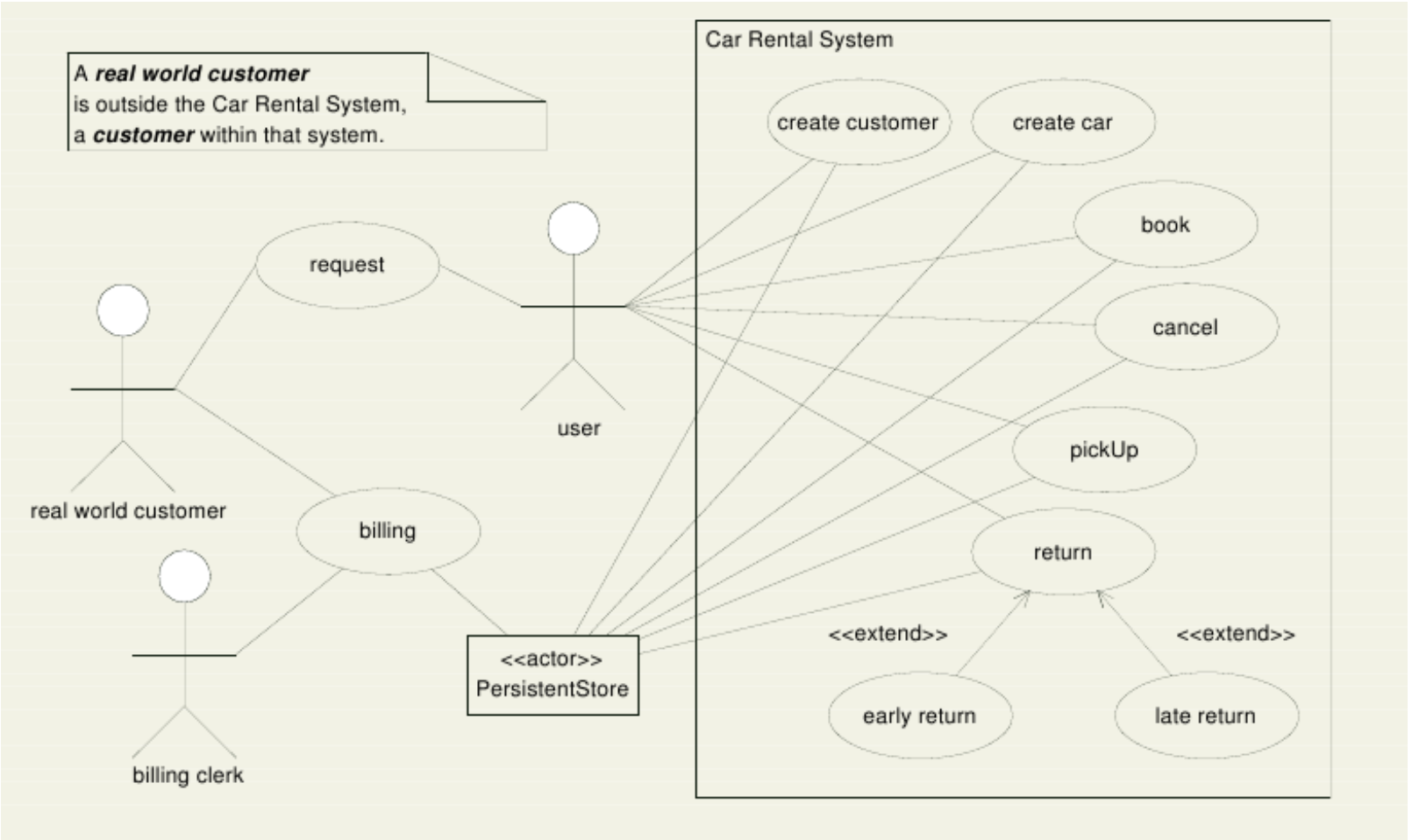
Important software design principle

Group information and behavior into discrete packages

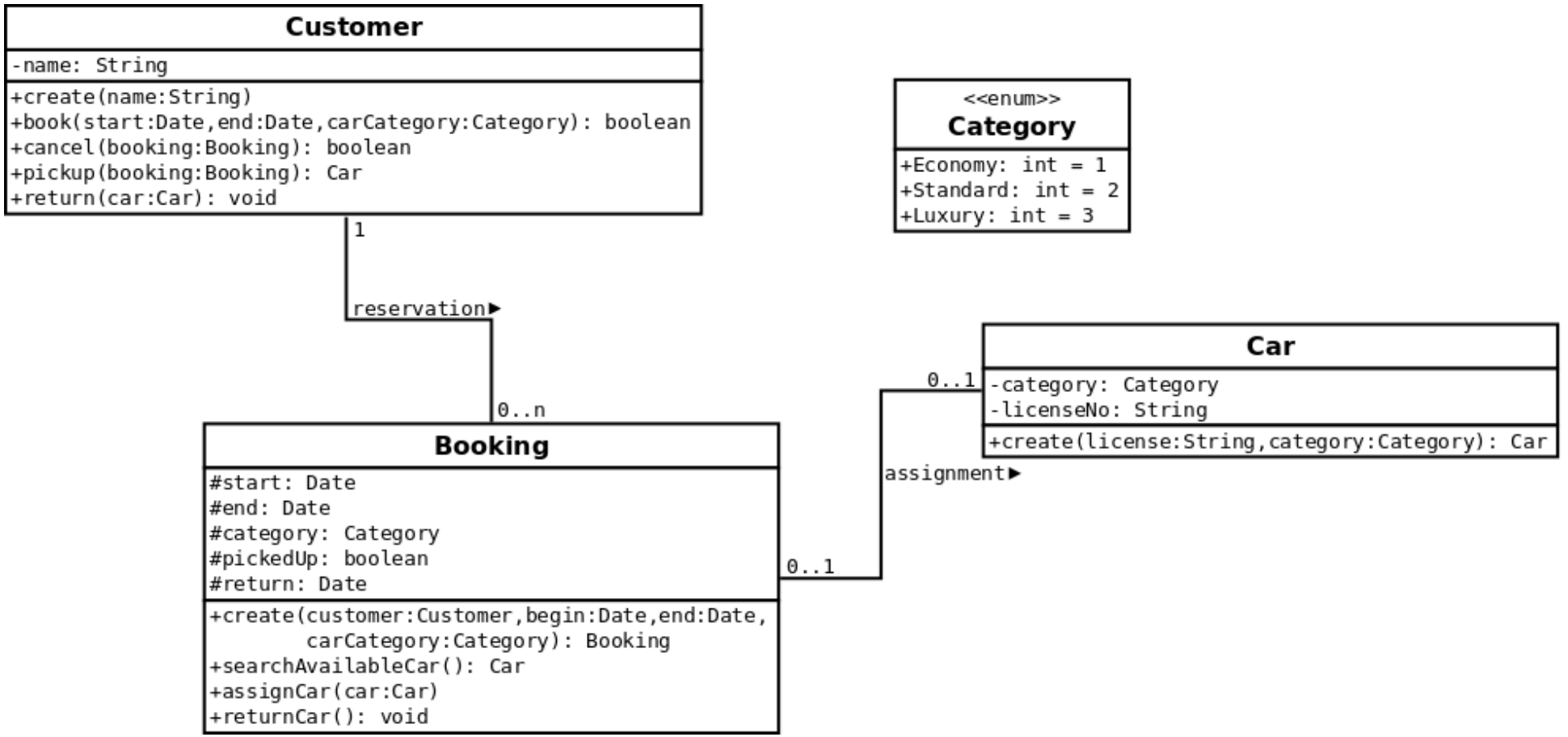
Minimize dependencies between these packages

Group Exercise: Car Rental System

Based on: http://www.db.informatik.uni-bremen.de/teaching/courses/ss2002_oose/lecture05.pdf



Class Diagram



Questions to Ask

Are there undesirable dependencies?

Does each class do one thing and do it well?

Are there responsibilities or code duplicated across classes?

Are there any classes, data items or methods missing?

Are the classes primitive?

Do you see any opportunities for reuse via inheritance?

What sort of changes can you imagine this system might need in the future? How difficult will it be to make these changes?

Assignments

- **Read the brief article on OO design quality, here:**

https://atomicobject.com/resources/oo-programming/oo-quality

Next week (28 March) no class

- **Prepare to present your project design to the class for review**
  - Eight teams on 4 April, eight teams on 11 April, randomly chosen (which means everyone needs to be ready on 4 April!)
  - Twenty minutes per project – 10 minutes to present the design, 10 minutes for critiques and suggestions
  - You can use PowerPoint, UML tools, the whiteboard, whatever works
  - You might want to install your UML tool on your laptop so you can make changes as they are suggested
  - Everyone in the class is expected to comment – 10% of your grade is participation!