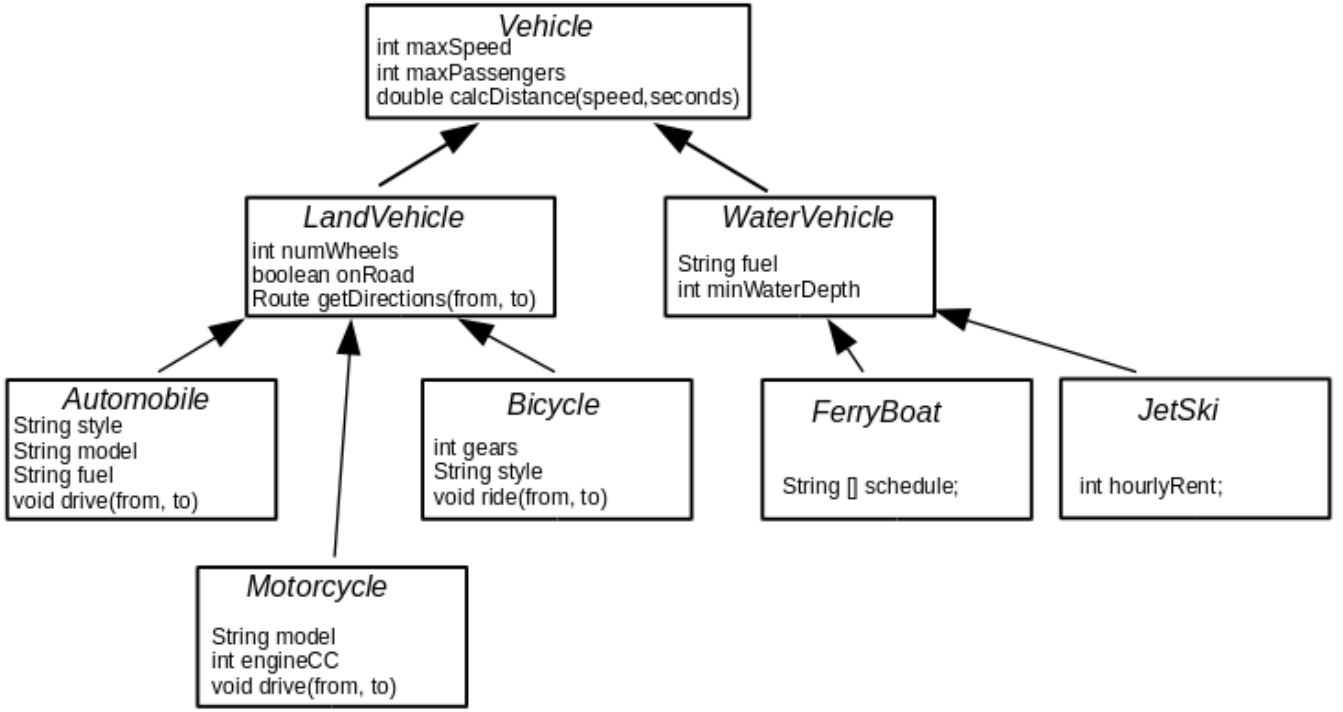
Object Oriented Design and Analysis
CPE 372

Lecture 5

Multiple Inheritance and Interfaces
Introduction to UML

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

IS-A Relationships



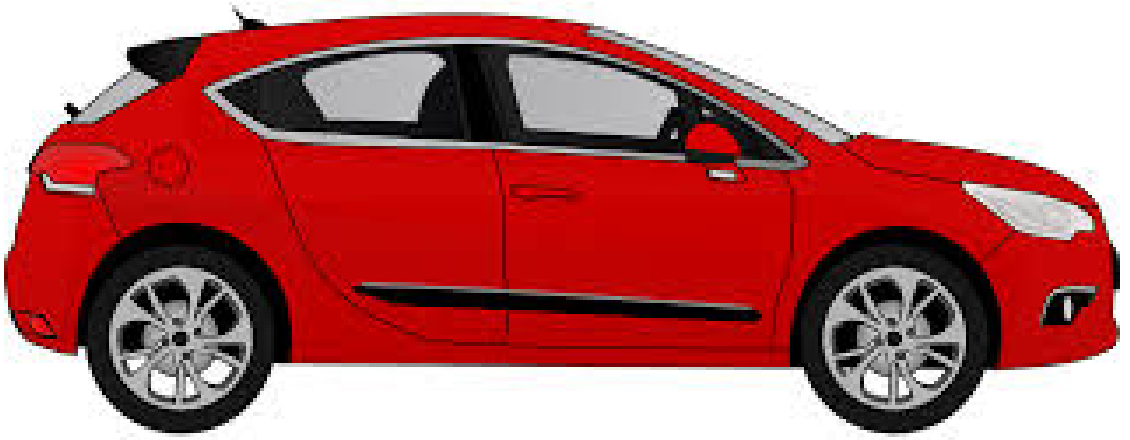
Subclass-superclass links in a class hierarchy represent “is-a” relationships

- A *LandVehicle* is a *Vehicle*
- An *Automobile* is a *LandVehicle*
- An *Automobile* is also a *Vehicle*

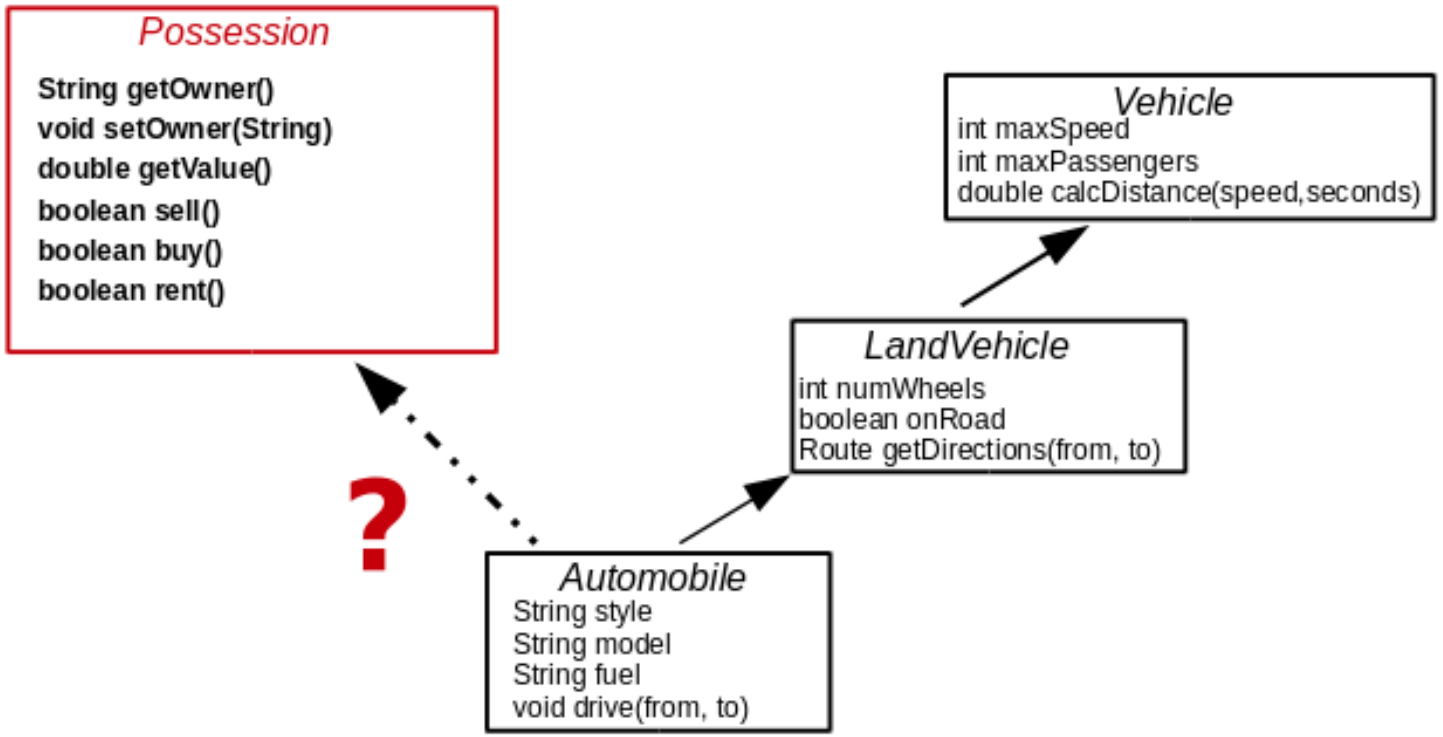
Many IS-A relationships are possible

An Automobile is also

- A machine
- A monthly expense
- A possession
- A status symbol



Can a class have multiple superclasses?



This depends on the programming language
In Java, the answer is NO
(In C++, the answer is yes...)

Multiple Inheritance has pros and cons

Advantages

- Allows you to represent and use common features from multiple different categories in defining your own classes
- Can increase opportunities for code reuse
- More closely mirrors the real world

Disadvantages

- Can get confusing, especially with multi-level hierarchies
- Introduces conflicts in determining correct use of overloading, overriding and polymorphism

Java Interfaces

Java has a language feature called ***interfaces*** that provides some of the benefits of multiple inheritance

Like an abstract class but:

- No member data items
- ***All*** methods are “abstract” and must be implemented by the “subclass”
- Captures similarities between (possibly) very different classes
- Permits polymorphism – can call interface methods even if you do not know the actual class

Possession Interface

```
public interface Possession
{
    String getOwner();
    void setOwner(String);
    double getValue();
    double sell();
    boolean buy(double price);
}
```

implements

extends *Vehicle*

implements

extends *Building*

implements

extends *Computer*

What good does this do us?

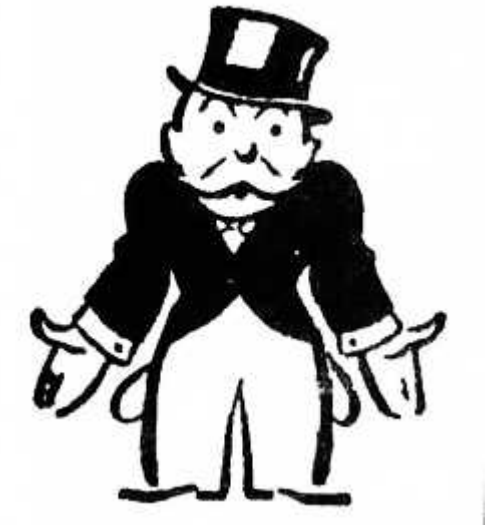
A class that implements an interface must provide code for all its methods

We cannot inherit any behavior

BUT the interface defines a “category” of behavior

We can use its methods without knowing what specific class we’re dealing with.

```
public double liquidateAssets( ArrayList<Possession> assets)
{
    double totalFunds = 0;
    for (int i=0; i < assets.size(); i++)
    {
        Possession p = assets.get(i);
        totalFunds += p.sell();
    }
    return totalFunds;
}
```



Iterator: a very useful interface

Interface Iterator<E>

- *E can be any type of object*

boolean hasNext()

- *Returns true if the iteration has more elements*

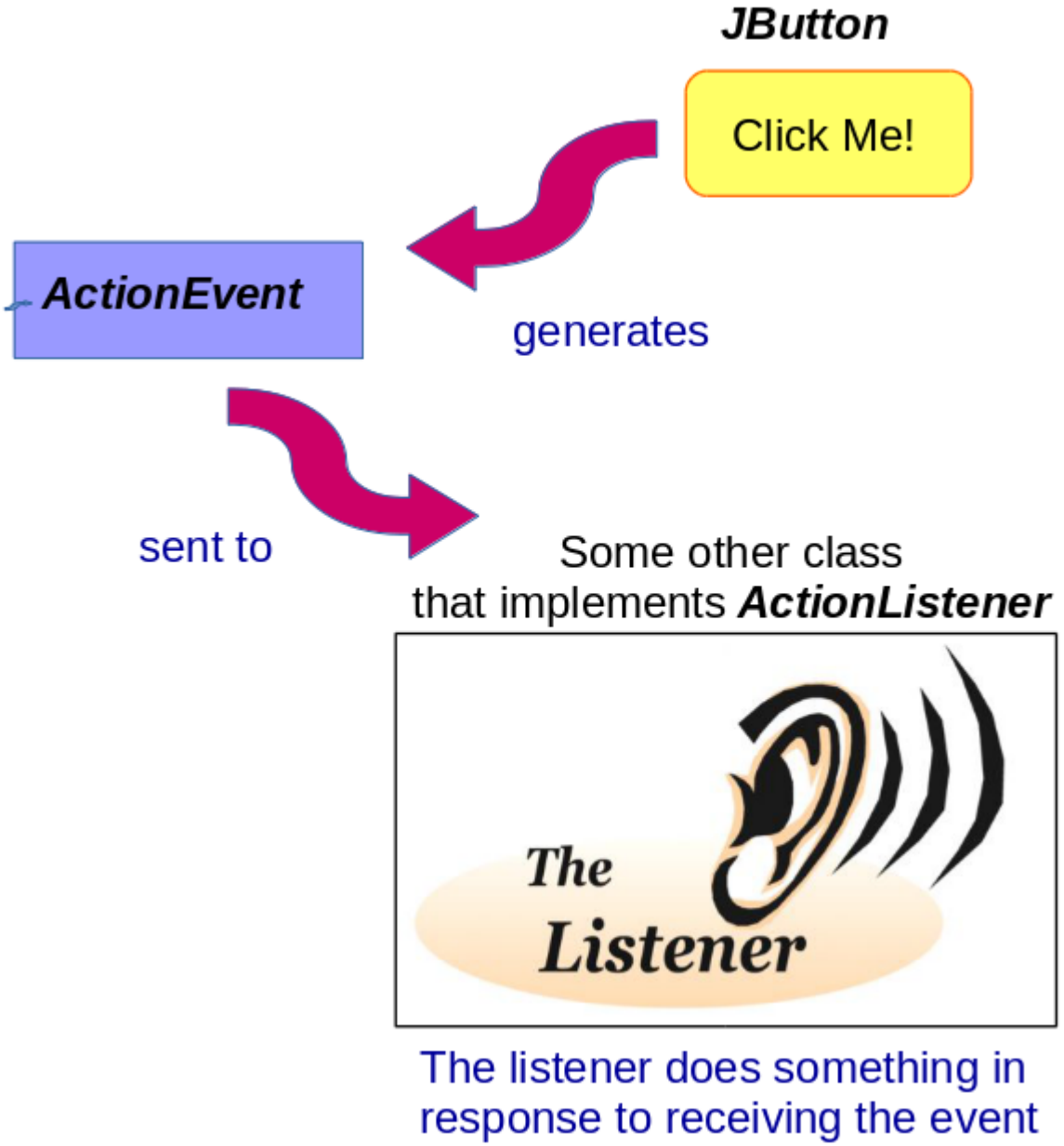
E next()

- *Returns the next element in the iteration*

This interface captures the important general idea of going through a group or set of “things” one at a time.

Unlike a “for” loop iteration, it does not require the code to know how many things are in the set.

Java Swing Event Listeners

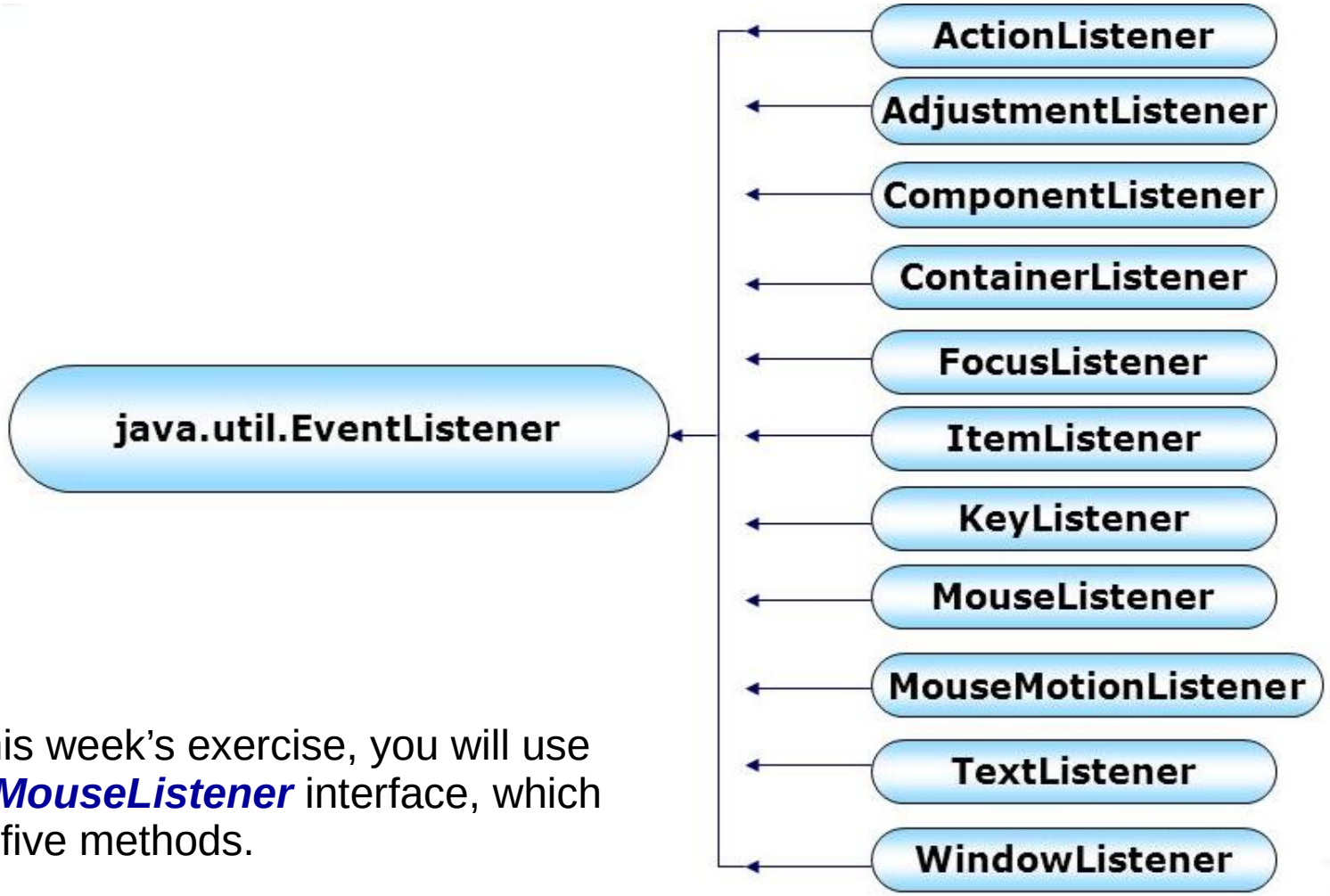


For an example, take a look at my ***FigureViewer*** class.

This class implements the ***ActionListener*** interface, which defines one method:

```
public void
actionPerformed(Event e);
```

Swing Event Listeners



In this week's exercise, you will use the ***MouseListener*** interface, which has five methods.

Introducing UML

Unified Modeling Language (UML) is a standardized notation for developing and documenting software designs

Uses diagrams plus text

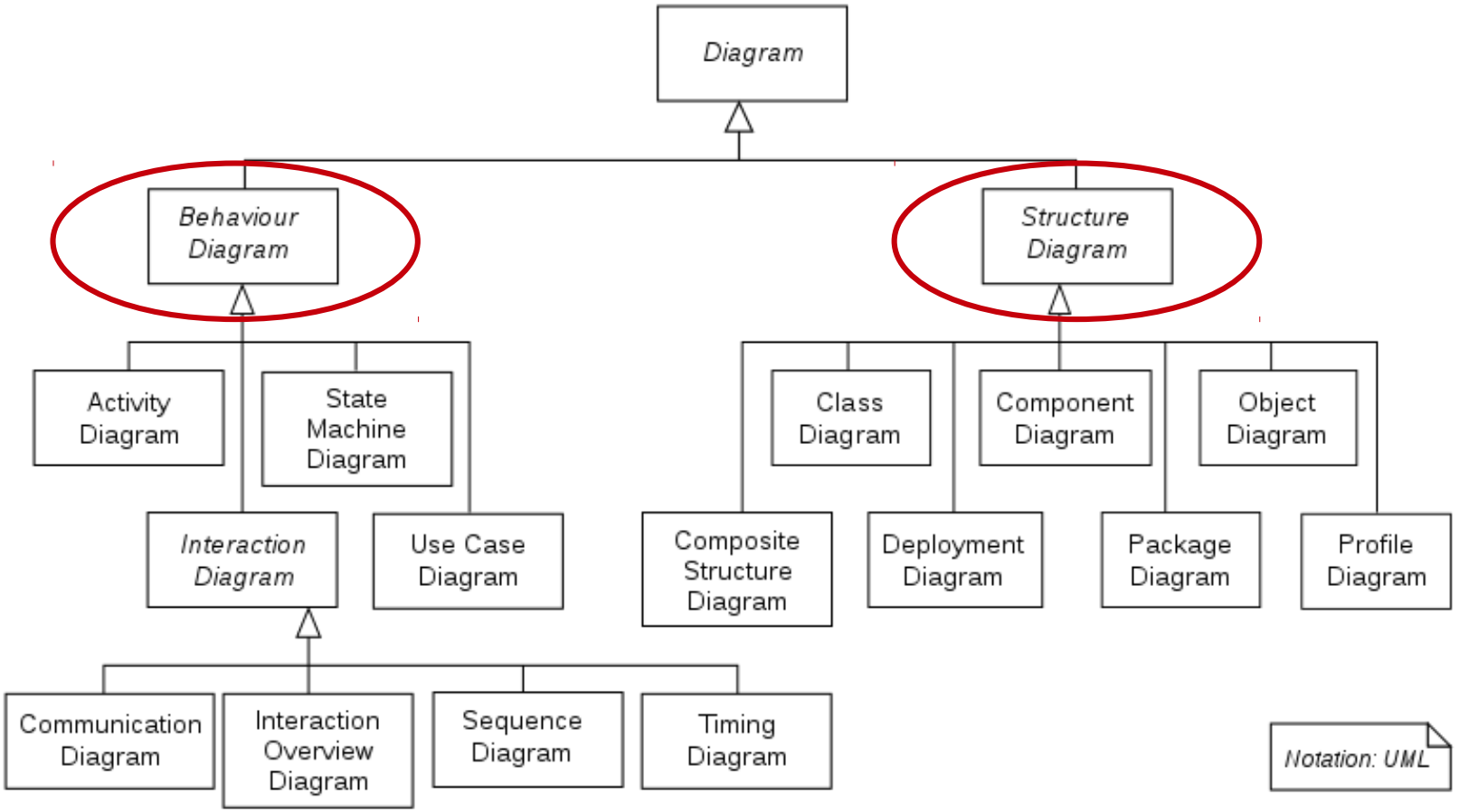
Meaning of diagram symbols is defined by the standard

Provides a way to record the results of design brainstorming

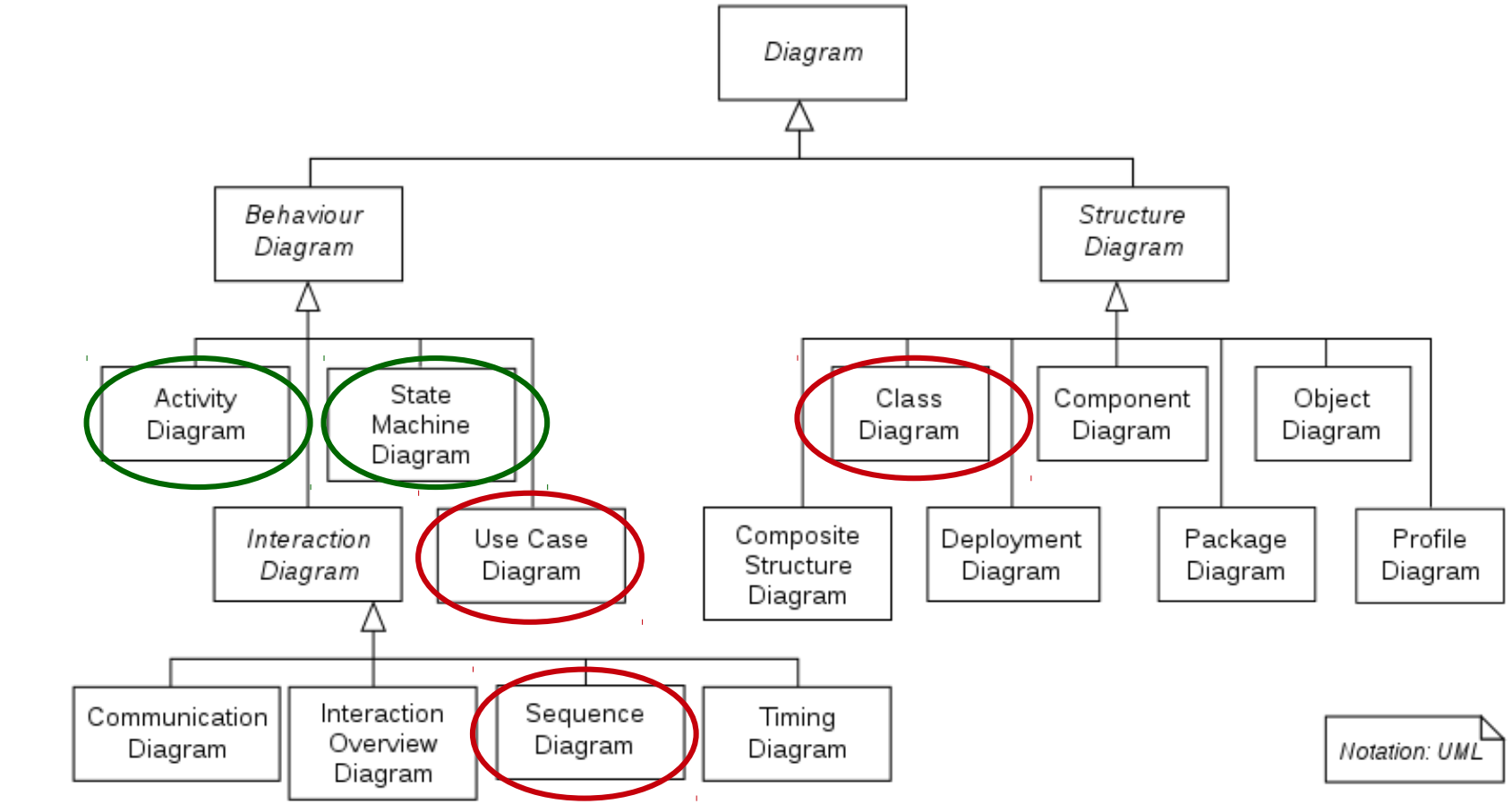
Facilitates communication among developers

Not limited to OOD approach but strongly associated with OOD

UML considers both structure and behavior



We will use a subset of UML for your project designs



Required

Optional

Don’t get hung up on the diagrams!

The purpose of creating (and revising) UML diagrams is to help you clarify and communicate your ideas

But don’t expect to get it “right” the first time...

Design is an iterative process

UML tools are often quite complicated and hard to use

Some are very expensive also

Two that I have used:

- Visual Paradigm

  https://www.visual-paradigm.com/solution/freeumltool/

- MagicDraw

  https://www.nomagic.com/products/magicdraw

  Both have free “community” or “trial” editions

Tools that can create diagrams without enforcing semantics:

- Dia (comes with many Linux distributions)
- Visio (Microsoft)

Use Cases

"Use cases are a technique for capturing the functional requirements of a system. Use cases work by describing the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used." *Martin Fowler, UML Distilled (3rd edition, 2004)*

A use case is a set of different usage scenarios tied together by a **common goal**.

Components of a Use Case

Name - Simple label usually starting with a verb that expresses the purpose of the use case

Actors – Users or external systems that will interact with the system under design

Goal – What the actor(s) are trying to accomplish

Preconditions – Statements that must be true before the use case can begin

Main success scenario – Step by step description of what the actor does and how the system responds, if things go as planned

Extensions – Alternative scenarios that could occur if there is some unusual condition

Postconditions – Statements that are true if the use case completes successfully

Example: Scrabble Game

Name: Take a turn

Actor(s): Current player (CP), opposing players (OP)

Goal: Create a new word on the board

Preconditions: None

Scrabble Example continued

Main success scenario:

1. CP locates a word in her letter tiles
2. CP finds a location where the word will fit
3. CP places the tiles on the board
4. System calculates the score by adding letter values while applying any special bonuses due to letter location
5. System adds the turn score to the overall score for CP
6. CP draws new letters to replace those used

Scrabble Example continued

Extension scenario (a):

4. OP challenges the legality of the word
5. System looks up word in the dictionary
6. Word is illegal
7. CP removes word and loses turn

Scrabble Example continued

Extension scenario (b):

4. OP challenges the legality of the word
5. System looks up word in the dictionary
6. Word is legal
7. OP loses her turn
8. Return to 4 in main success scenario

Scrabble Example continued

Extension scenario (c):

1. CP cannot make a word with her available letters
2. CP turns in and replaces some or all of her tiles
3. CP loses her turn

Extension scenario (d):

3. CP uses all seven letters
4. System adds 50 points to the CP score
5. Return to 5 in main success scenario

There are more extensions, too!

Extension e. No more tiles to replace the ones used by CP (last round of the game)

Extension f: First play - can be located anywhere so skip step 2.

Extension g: At step 3, system detects that CP has made an error in placing tiles...

This is all a single use case!

Sorry if this is making your head hurt!

You will need to think about functionality at this level of detail if you are writing a Scrabble program.

Use Case Dos and Don’ts

Do

✔Be specific about information provided to system, system response

✔Think carefully about all possible paths through the scenario

✔Include all steps that are necessary to accomplish the Use Case goal

✔Start at a more general or abstract level, then refine to provide more details

Don’t

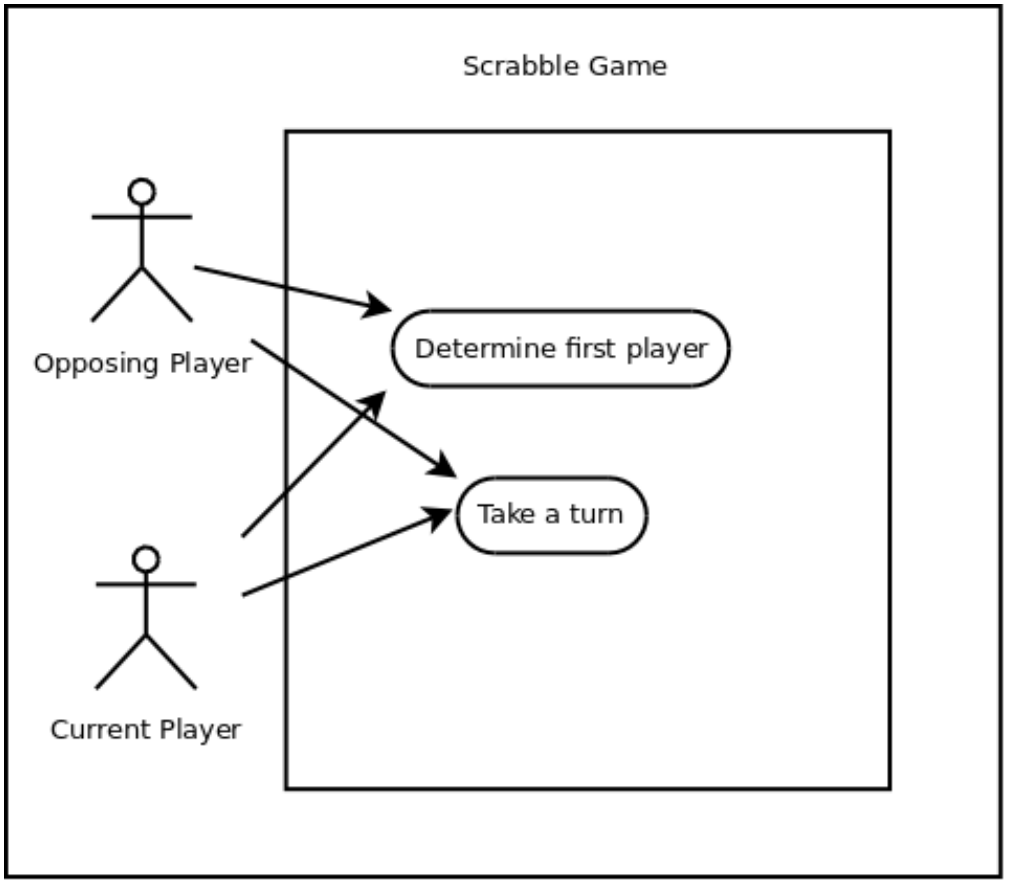
✗ Make assumptions about user interface implementation

✗Discuss details of system-side algorithms or processing

✗Include steps unrelated to the Use Case goal

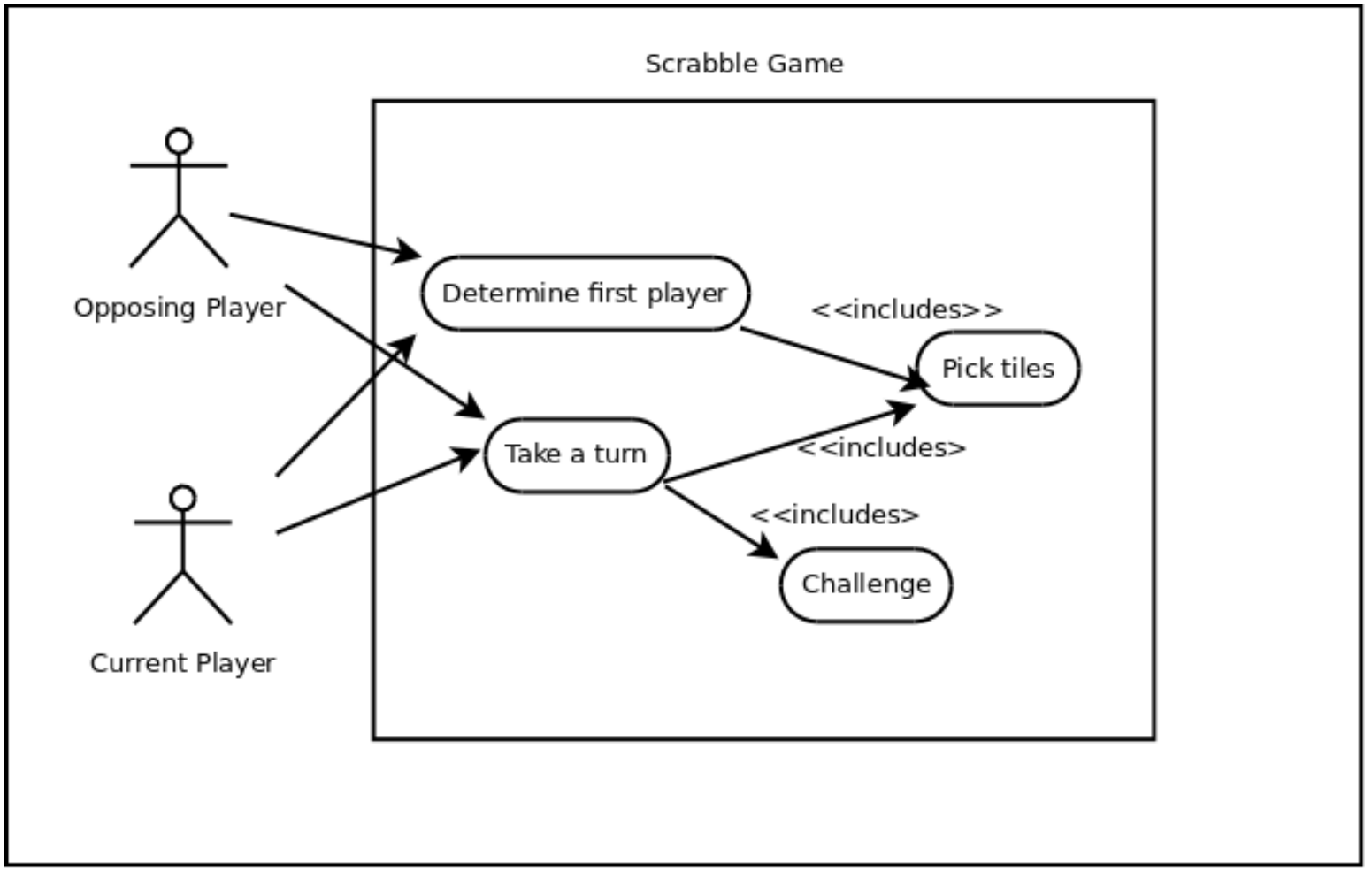
What about use case diagrams?

A use case diagram summarizes all the use cases for a system.



Can you think of any other use cases?

“Included” Use Cases



Specify included use cases if some subset of actions is part of more than one main use case

Assignments

1. Create a use case diagram for your project – print out and hand in next Thursday 28 February
2. Do Exercise 5 – due Monday 25 February at 17:00