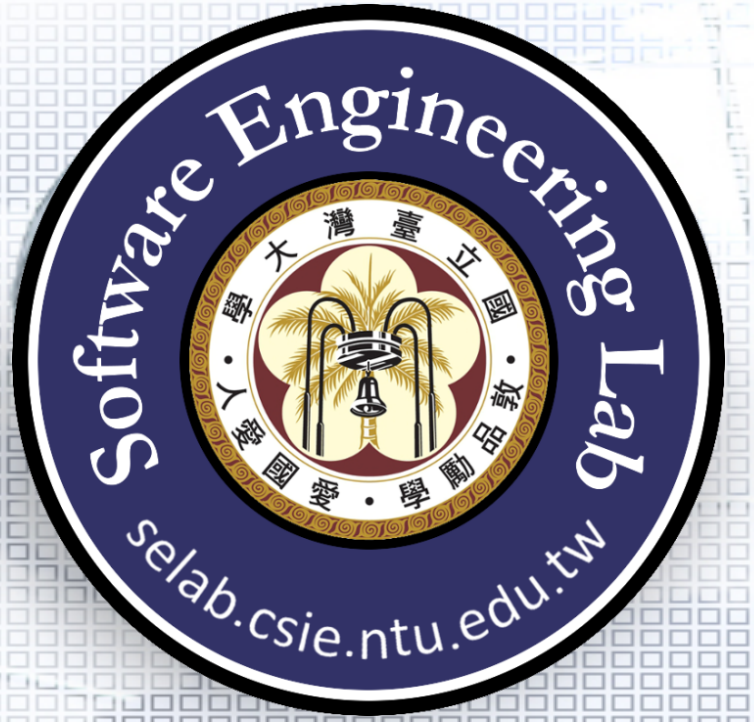


Strategy Pattern

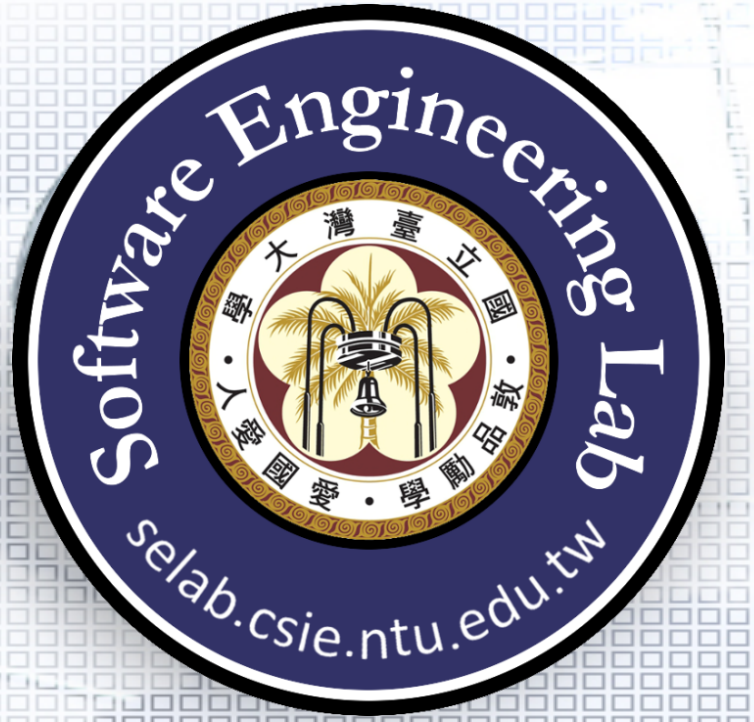
Prof. Jonathan Lee（李允中）
Department of CSIE
National Taiwan University

Design Aspect of Strategy

An algorithm or a family of algorithms

Outline

- Text Composition Design Requirements Statements
- Initial Design and Its Problems
- Design Process
- Refactored Design after Design Process
- Recurrent Problems
- Intent
- Strategy Pattern Structure
- Duck Game: Another Example
- Homework



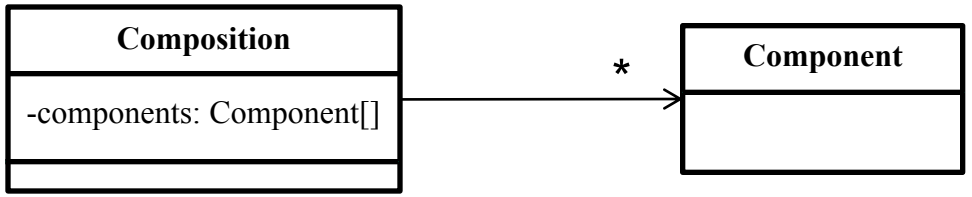
Text Composition Design (Strategy)

Prof. Jonathan Lee (李允中)

Department of Computer Science and Information Engineering
National Taiwan University

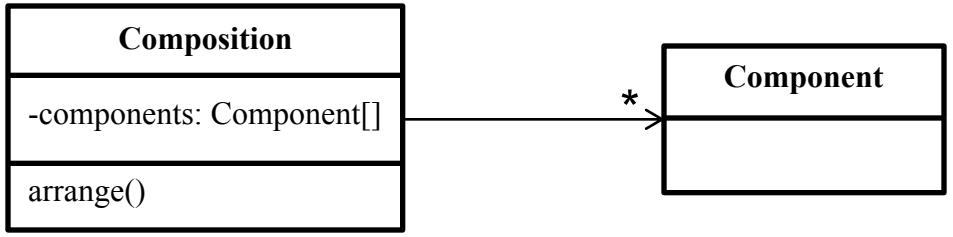
Requirements Statement$_1$

- The Composition class maintains a collection of Component instances, which represent text and graphical elements in a document.



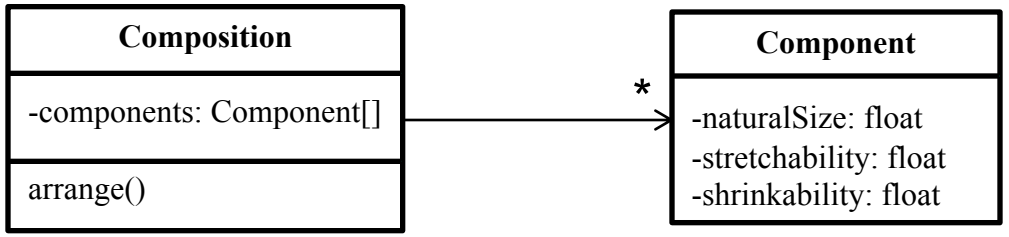
Requirements Statement$_2$

- A composition arranges component objects into lines using a linebreaking strategy.



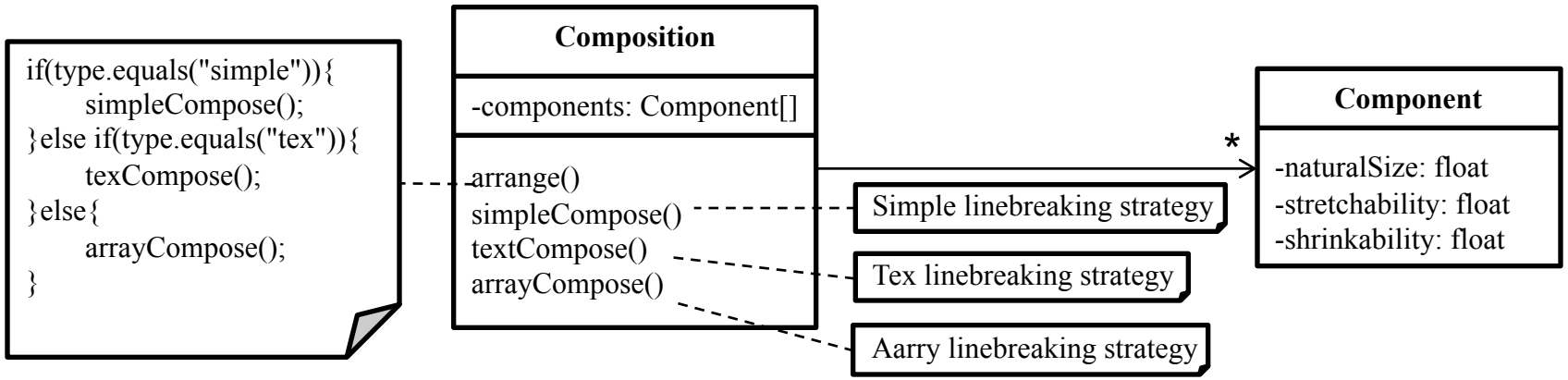
Requirements Statement$_3$

- Each component has an associated natural size, stretchability, and shrinkability.
- The stretchability defines how much the component can grow beyond its natural size; shrinkability is how much it can shrink.



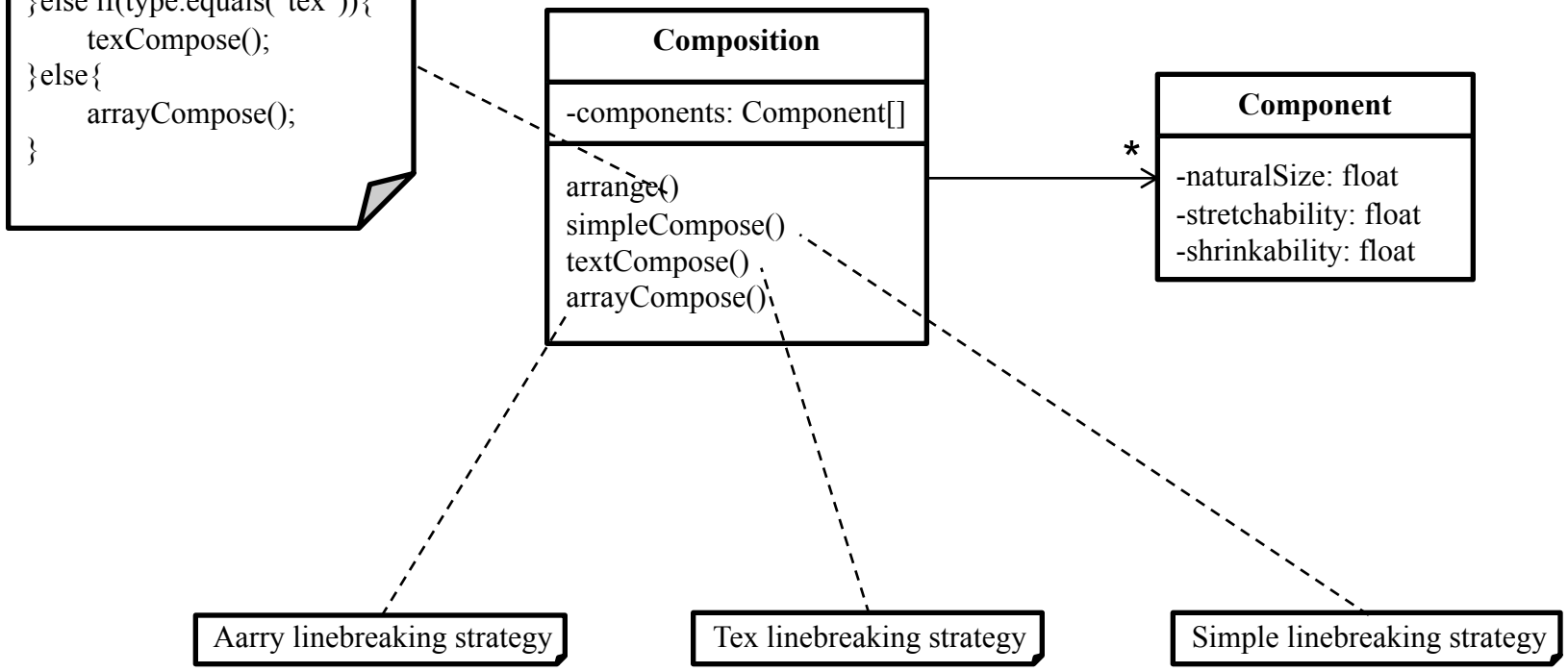
Requirements Statement$_4$

- When a new layout is required, the composition calls its compose method to determine where to place linebreaks.
- There are 3 different algorithms for breaking lines:
  - Simple Composition: A simple strategy that determines line breaks one at a time.
  - Tex Composition: This strategy tries to optimize line breaks globally, that is, one paragraph at a time.
  - Array Composition: A strategy that selects breaks so that each row has a fixed number of items. It's useful for breaking a collection of icons into rows, for example.

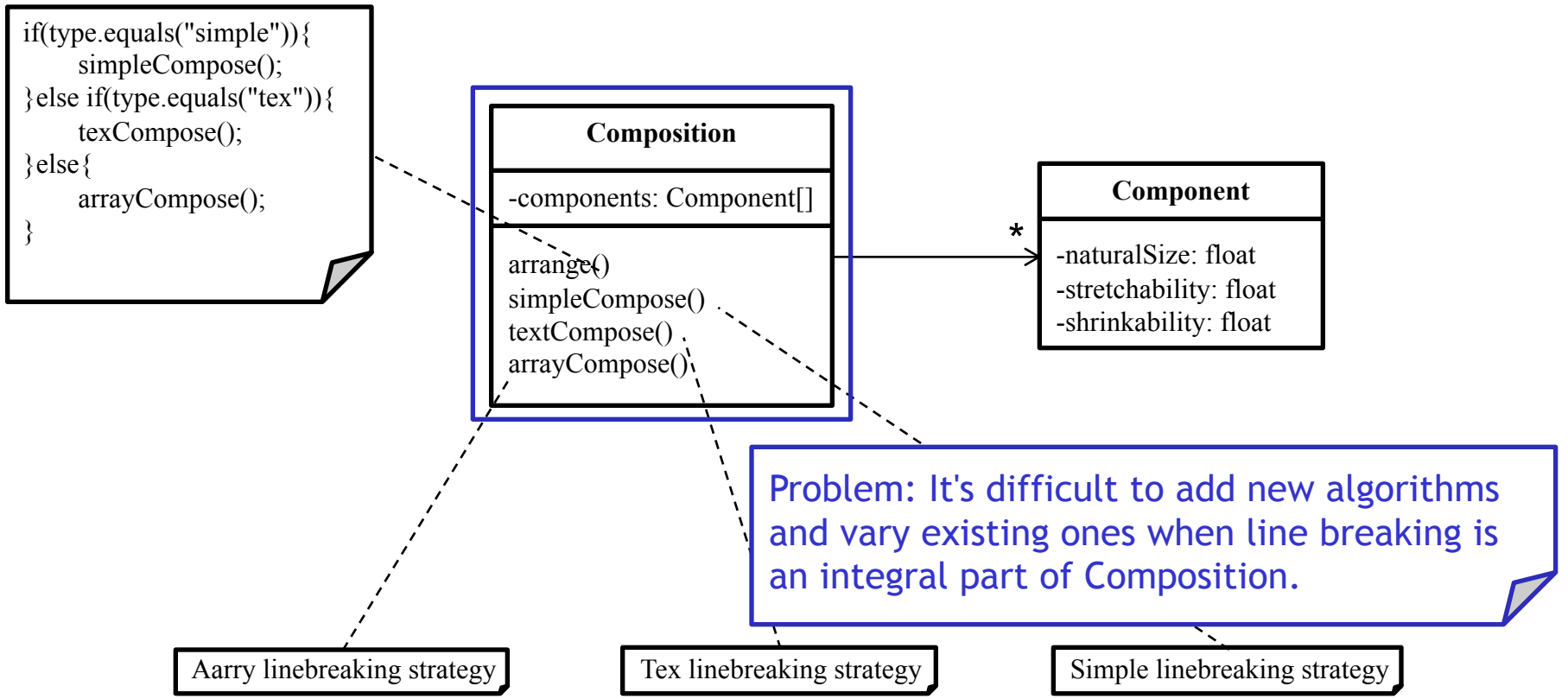


Initial Design

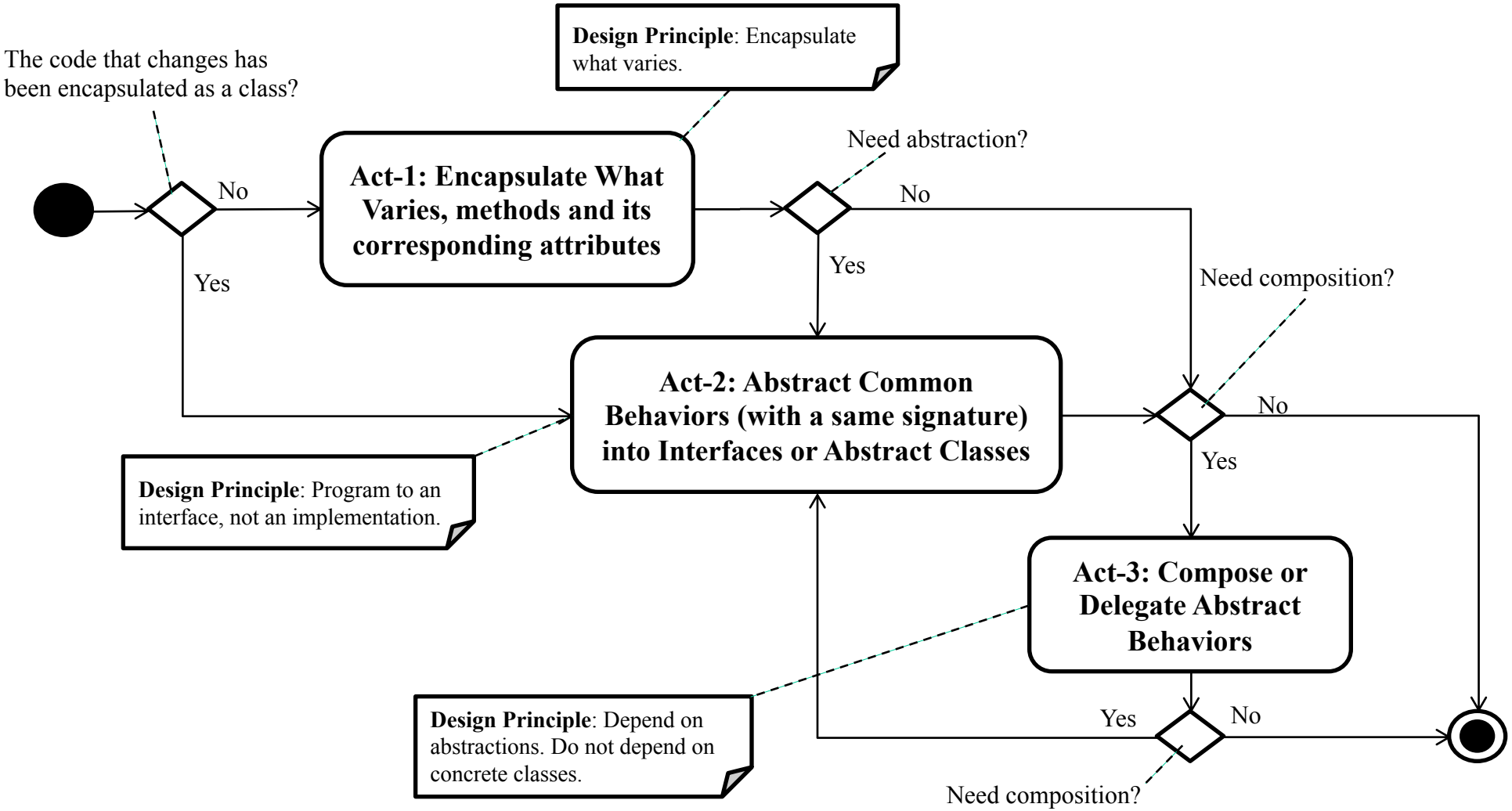
```
if(type.equals("simple")){
      simpleCompose();
}else if(type.equals("tex")){
      texCompose();
}else{
      arrayCompose();
}
```



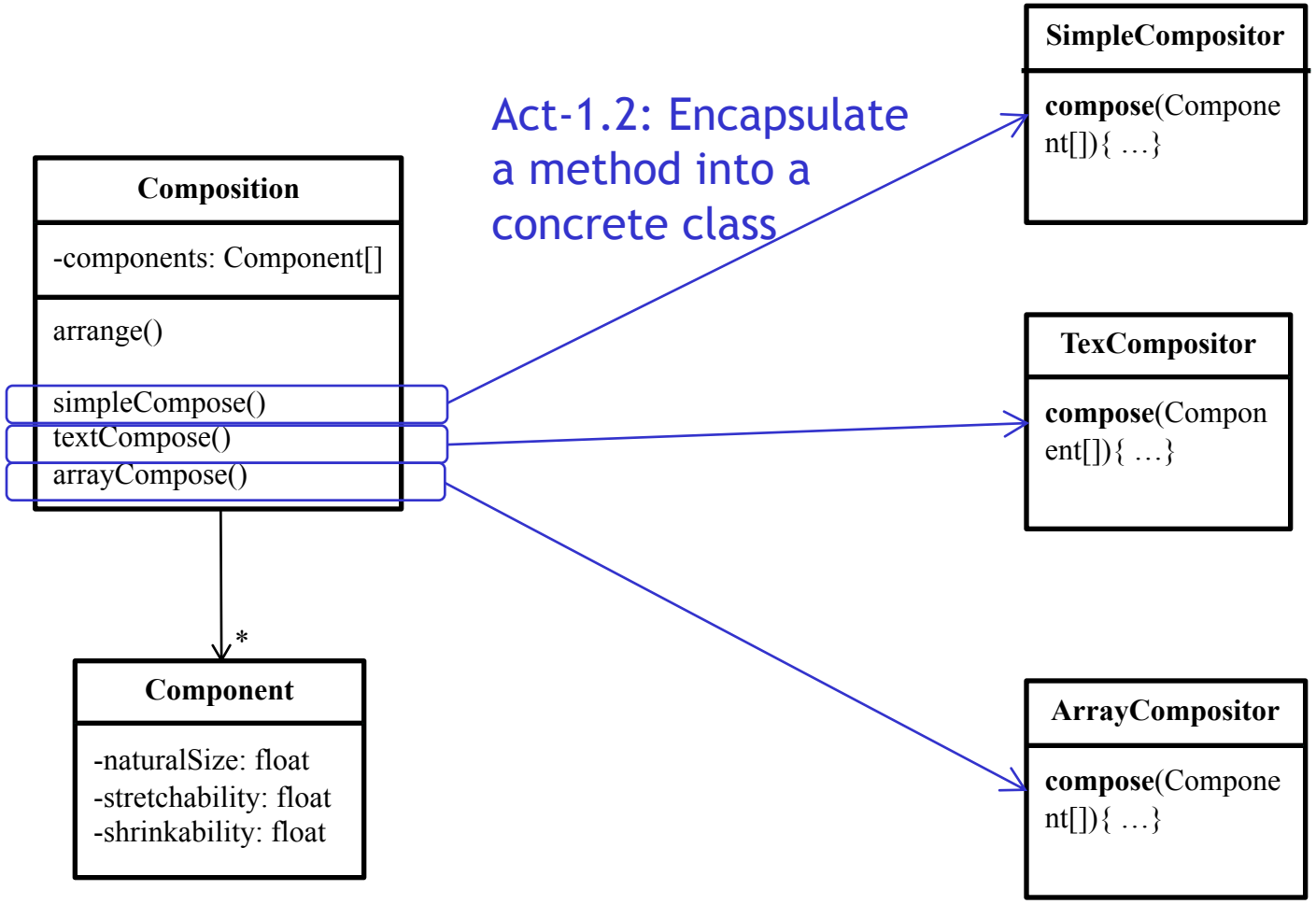
Problems with Initial Design



Design Process for Change

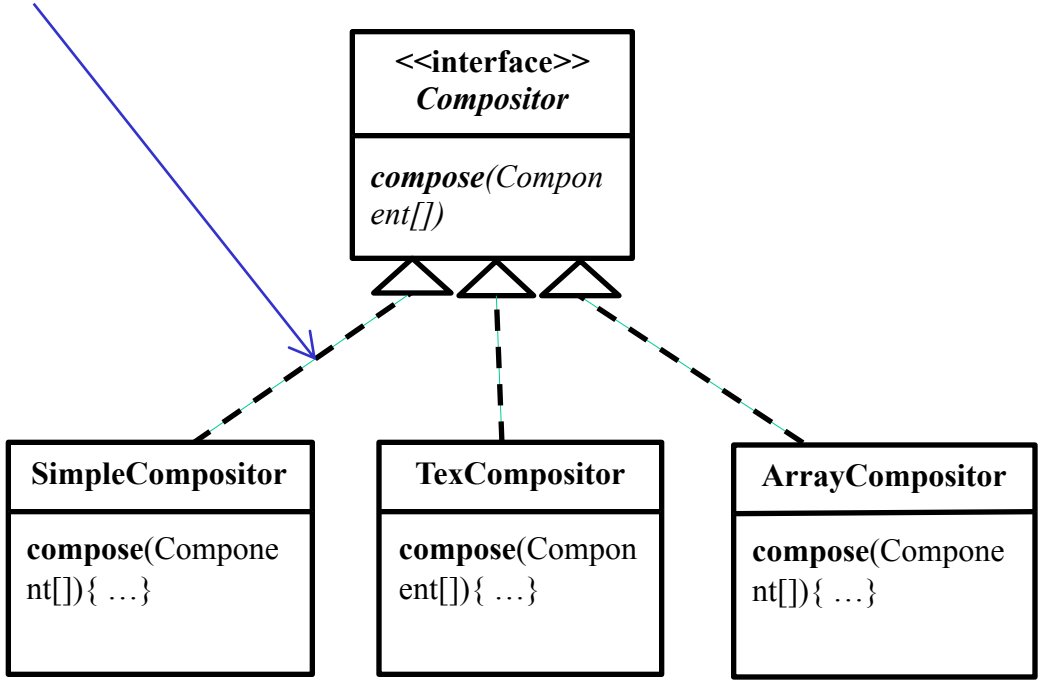


Act-1: Encapsulate What Varies



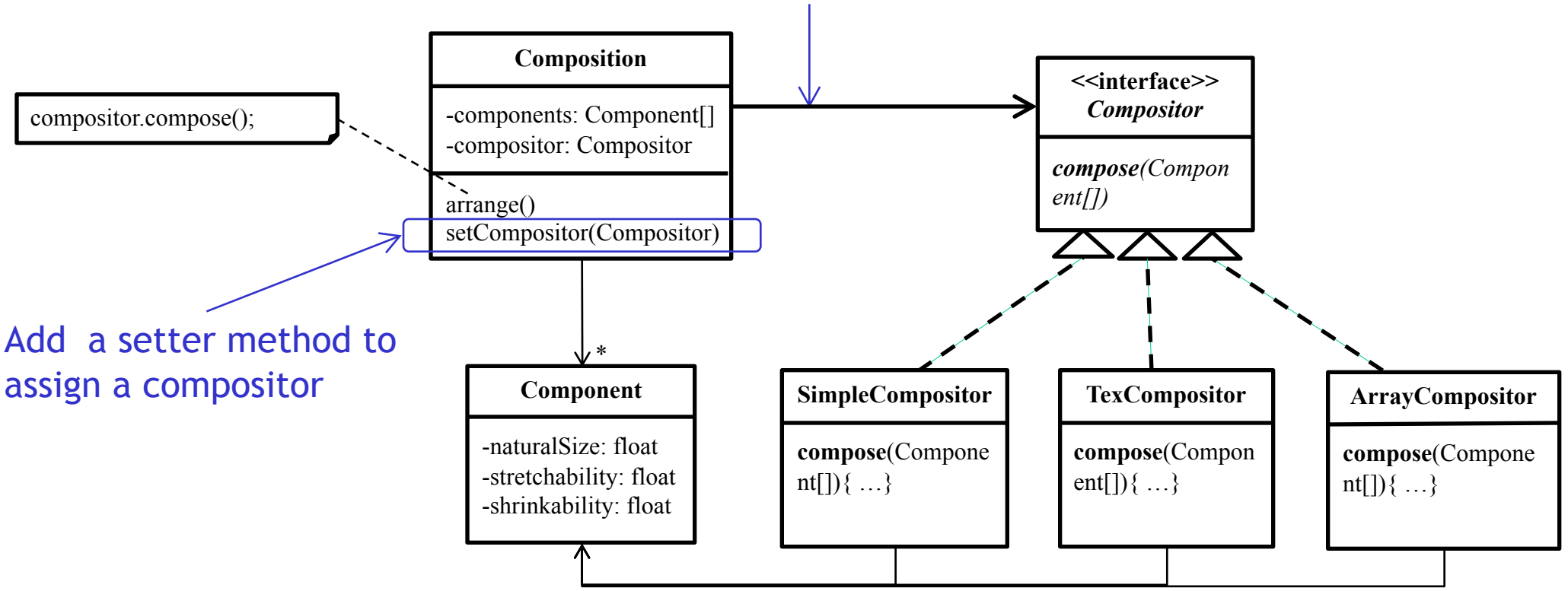
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



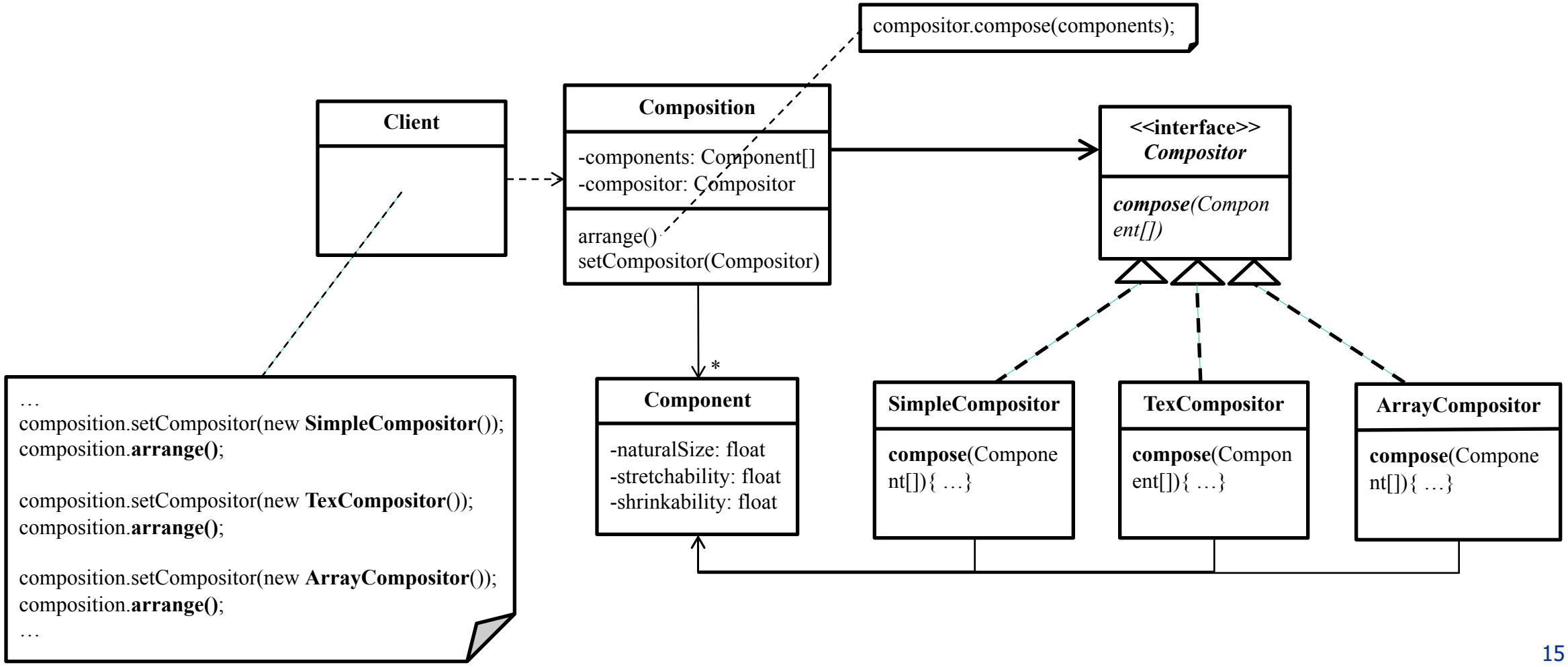
Act-3: Compose Abstract Behaviors

Act-3.1: Compose behaviors of an interface or an abstract class



Add a setter method to assign a compositor

Refactored Design after Design Process



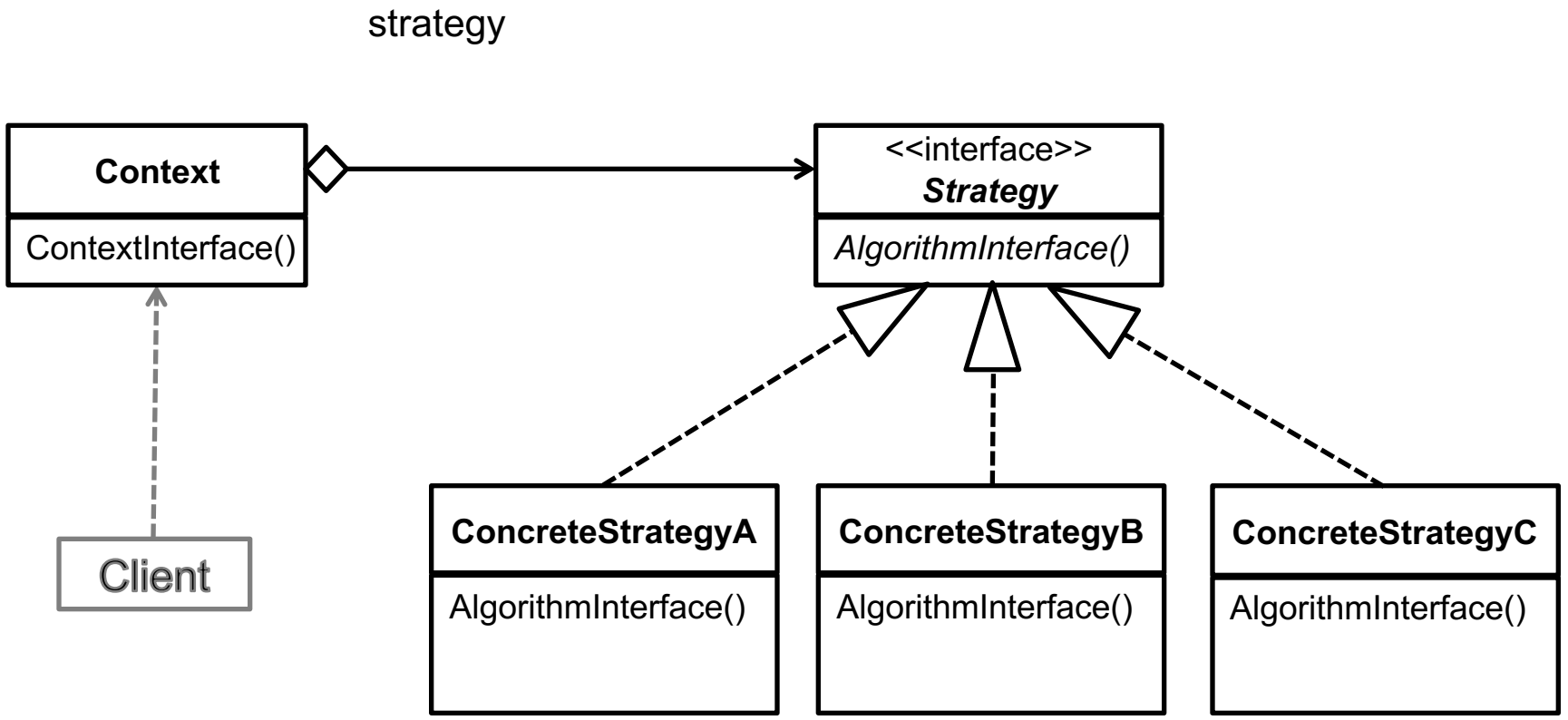
Recurrent Problems

- Multiple classes will be modified if new behaviors are to be added.
  - It's difficult to add new algorithms and vary existing ones.
- All duplicate code will be modified if the behavior is to be changed.
  - Different algorithms will be appropriate at different times.

Intent

- Define a family of algorithms, encapsulate each one, and make them interchangeable.
- Strategy lets the algorithms vary independently from clients that use it.

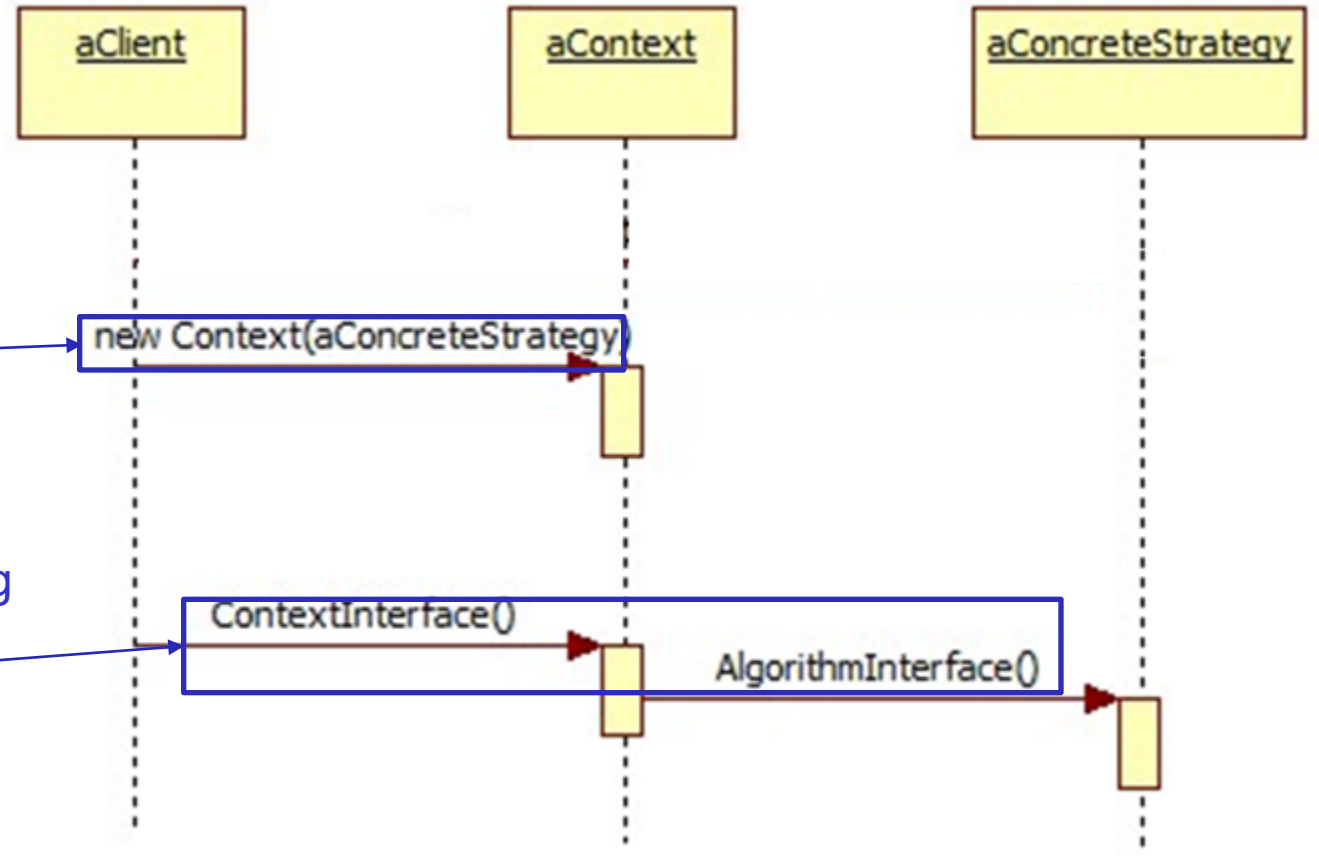
Strategy Pattern Structure$_1$



Strategy Pattern Structure$_2$

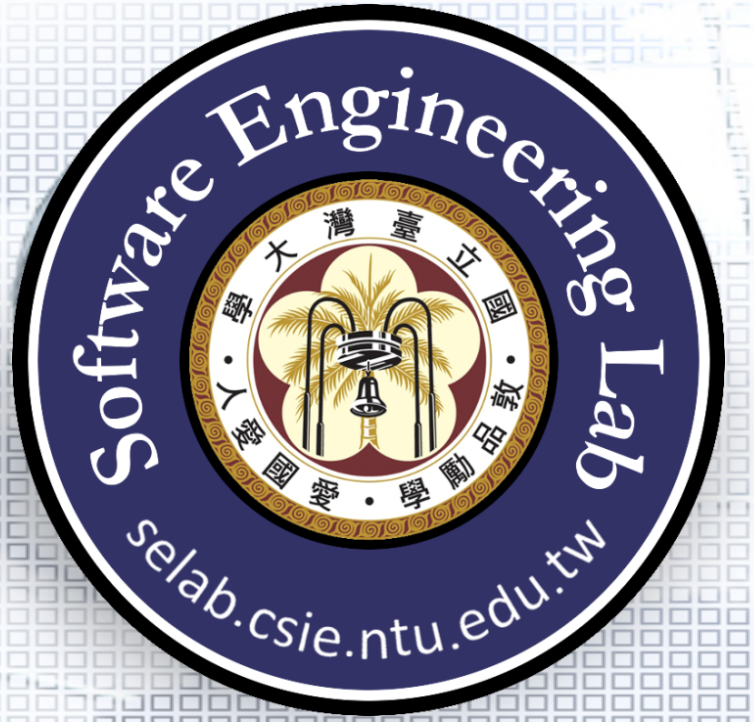
1. Client assigns a concrete strategy to Context.

2. Client perform the strategy through invoking Context's method.



Strategy Pattern Structure$_3$

	Instantiation	Use	Termination
Client	Other class except classes in the strategy pattern	Other class except classes in the strategy patterns	Other class except classes in the strategy pattern
Context	Other class or the client class	Client passes a ConcreteStrategy reference to this class and delegates request to it	Other class or the client class
Strategy	X	Context uses this interface to call the algorithm defined by a ConcreteStrategy	X
Concrete Strategy	The client class or other class except classes in the strategy pattern	Context uses this class which is passed through reference by client through polymorphism	Classes who hold the reference of ConcreteStrategy



Duck Game (Strategy)

Prof. Jonathan Lee (李允中)

Department of Computer Science and Information Engineering
National Taiwan University

Requirements Statement

- There are four types of ducks in the game: MallardDuck, RedheadDuck, RubberDuck, and DecoyDuck.
- All types of the ducks have the same swim behavior but are with different displays.
- Some ducks can fly with wings, but some cannot fly.
- A duck can quack, squeak, or be silent.
- A duck can change its fly or quack behavior at run time.
- New fly or quack behaviors can be added, and the existing behaviors can be modified at compile time.

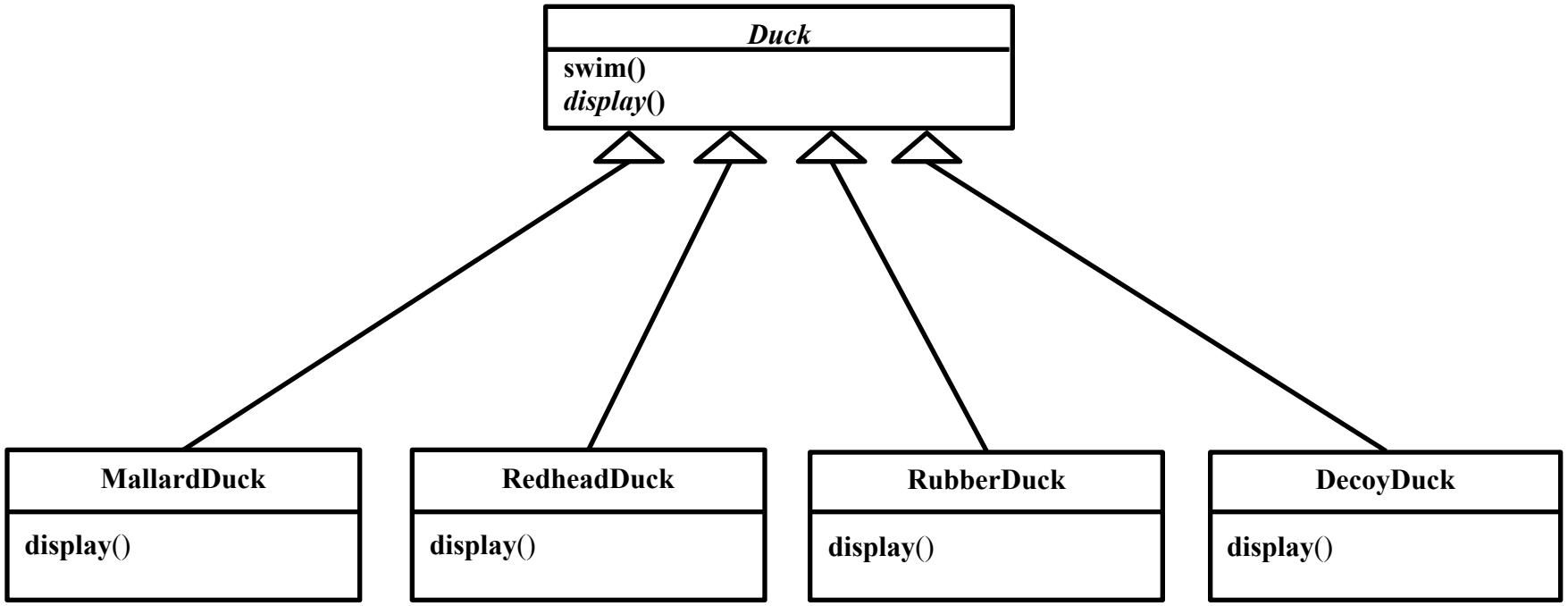
Requirements Statement$_1$

- There are four types of ducks in the game: MallardDuck, RedheadDuck, RubberDuck, and DecoyDuck.

MallardDuck	RedheadDuck	RubberDuck	DecoyDuck

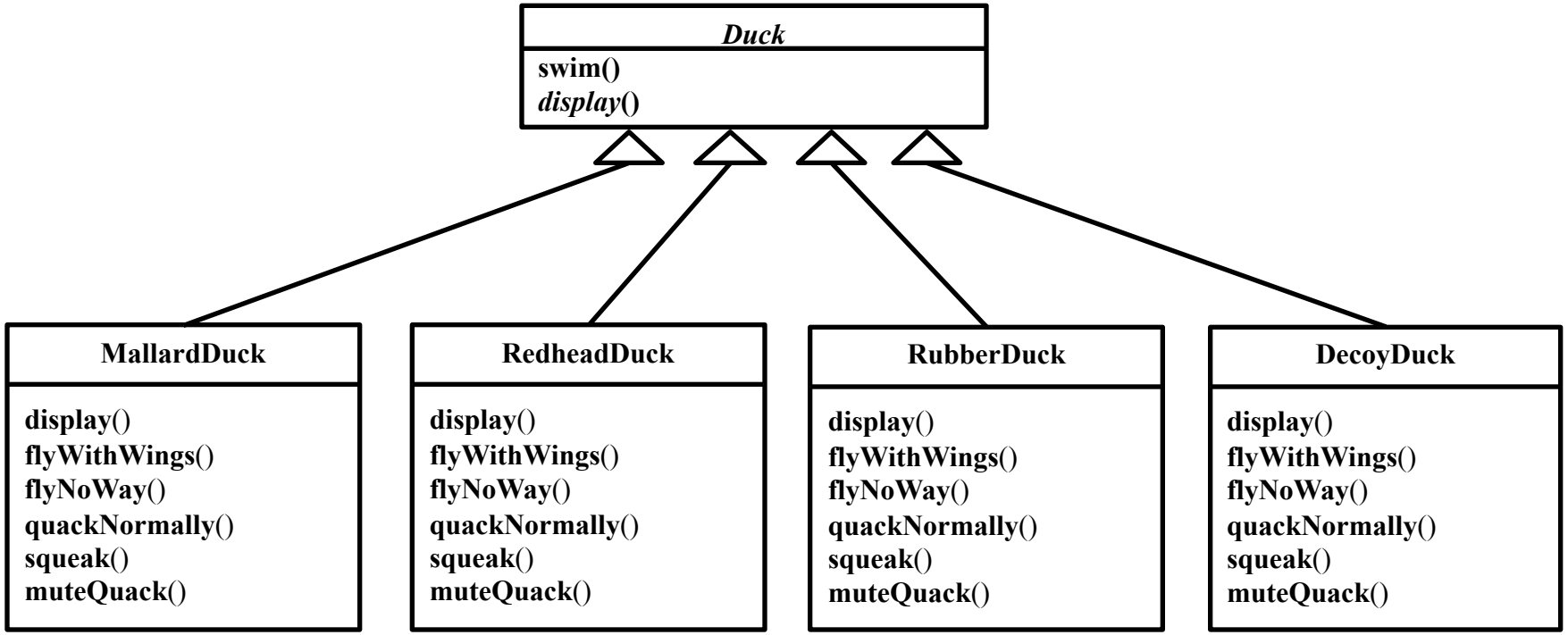
Requirements Statement$_2$

- All types of the ducks have the same swim behavior but are with different displays.



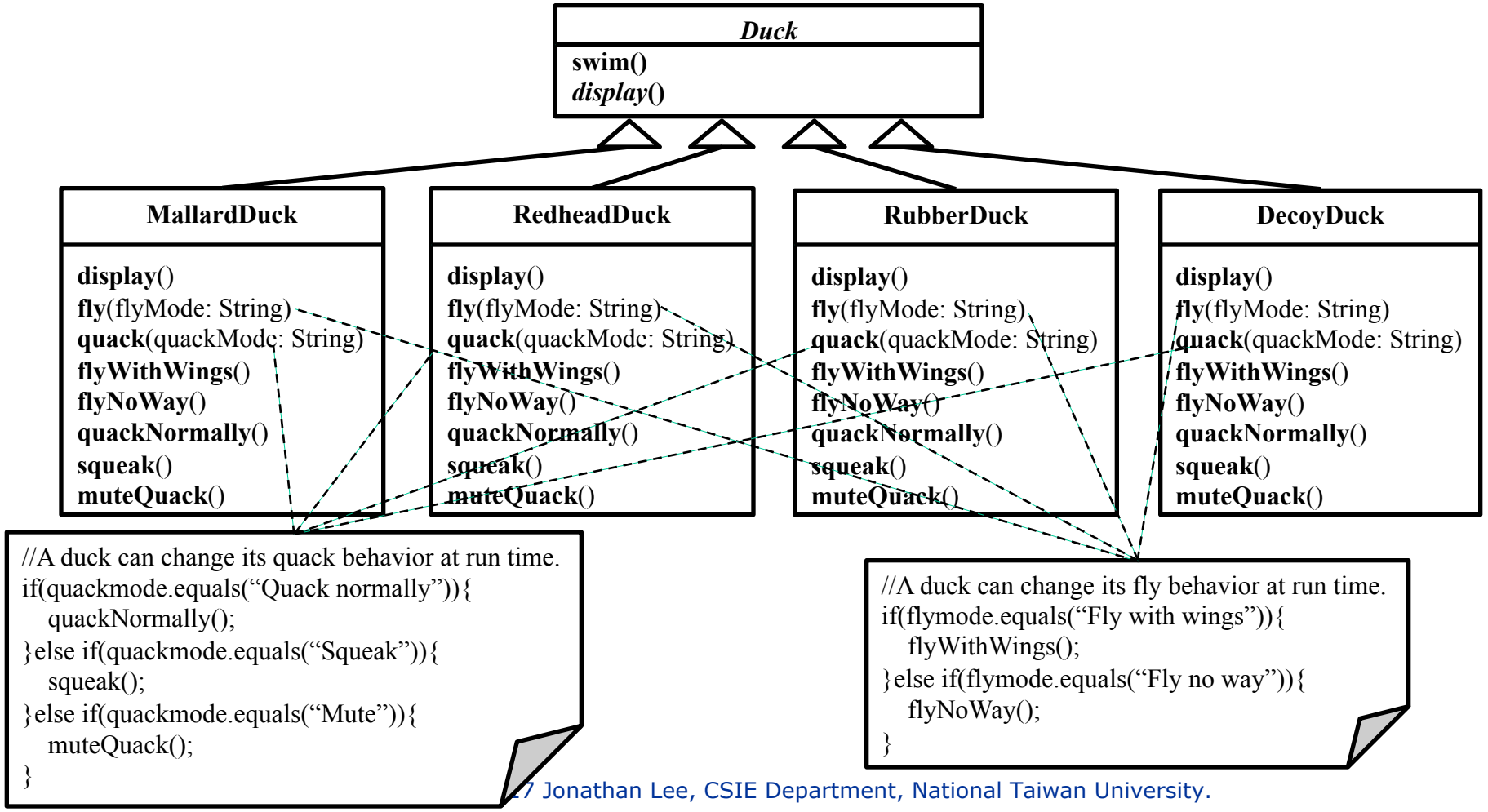
Requirements Statement$_3$

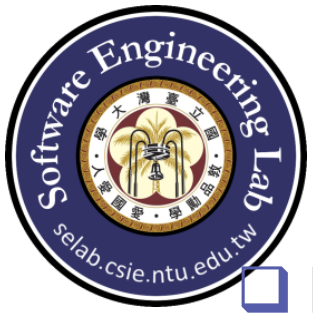
- Some ducks can fly with wings, but some cannot fly.
- A duck can quack, squeak, or be silent.



Requirements Statement$_4$

- A duck can change its fly or quack behavior at run time.

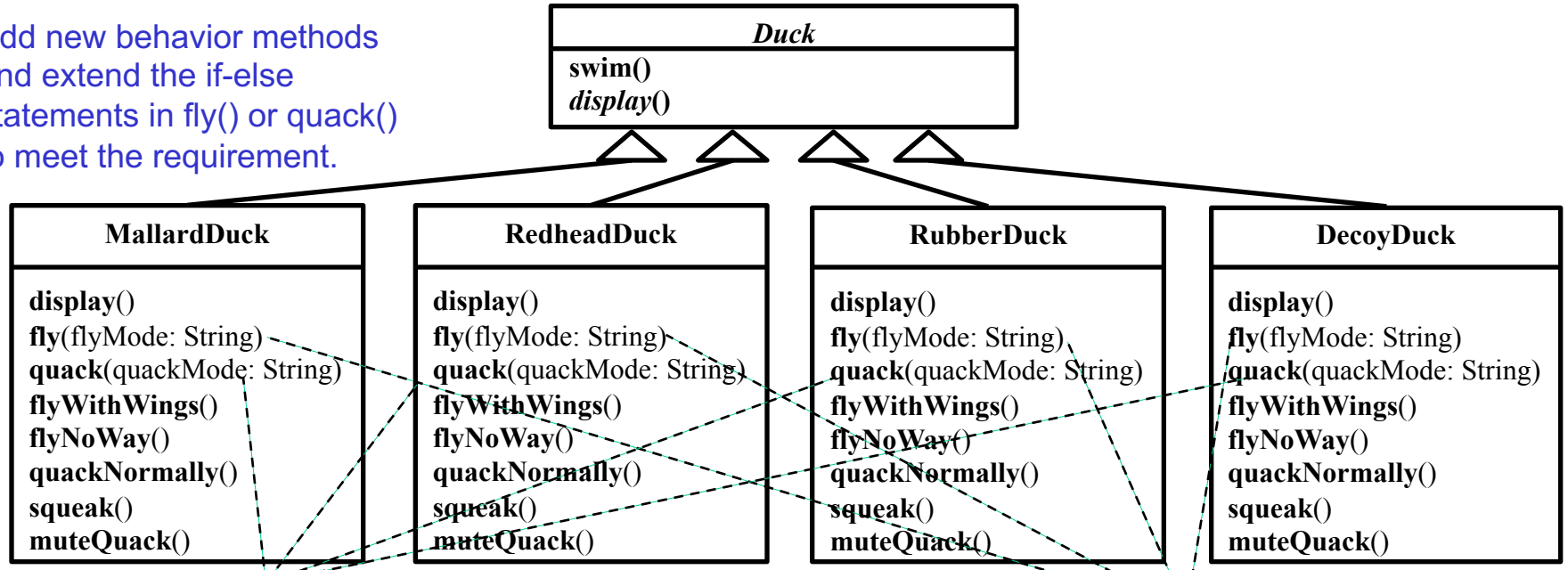




Requirements Statement$_5$

- New fly or quack behaviors can be added, and the existing behaviors can be modified at compile time.

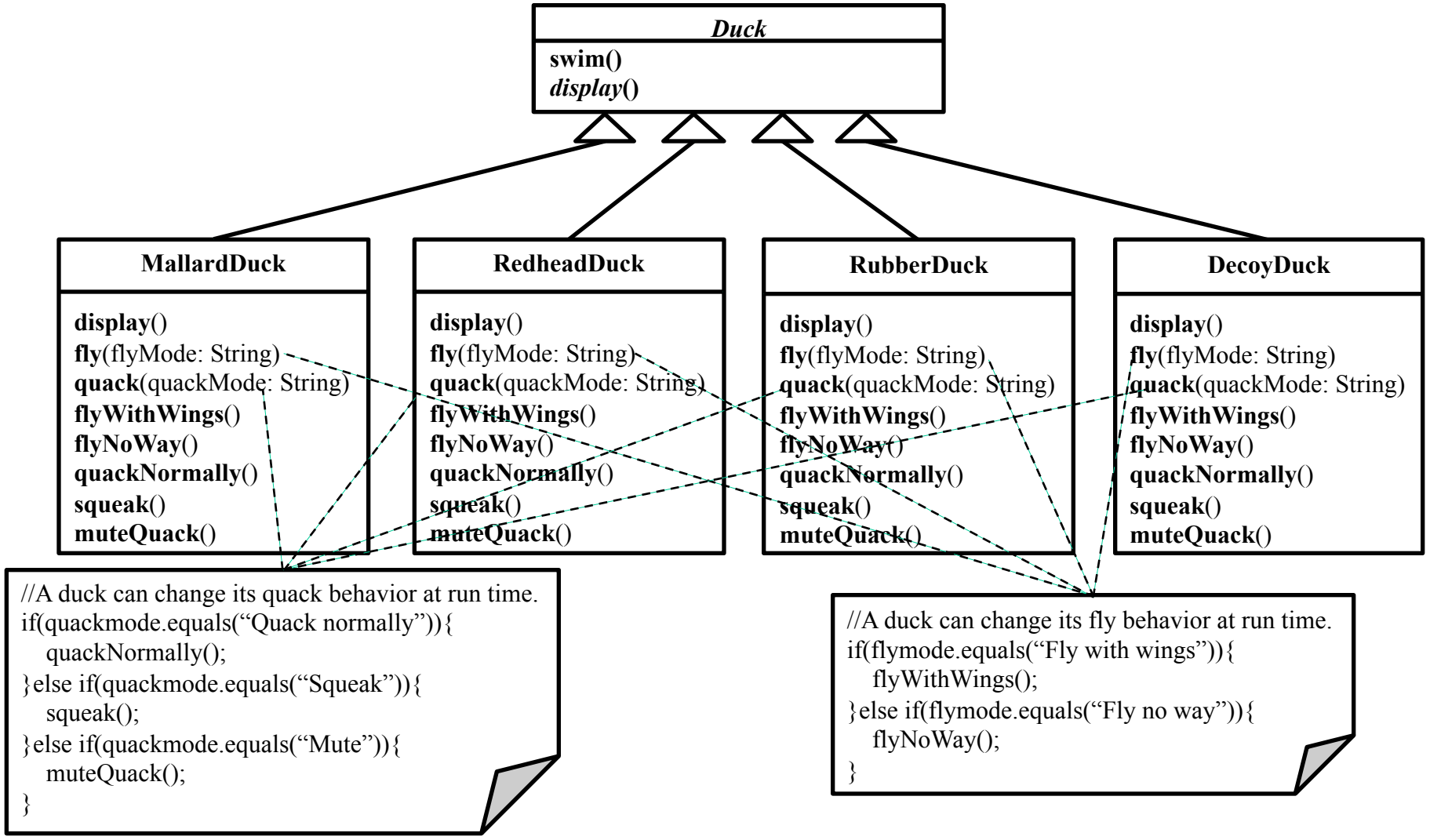
Add new behavior methods and extend the if-else statements in fly() or quack() to meet the requirement.



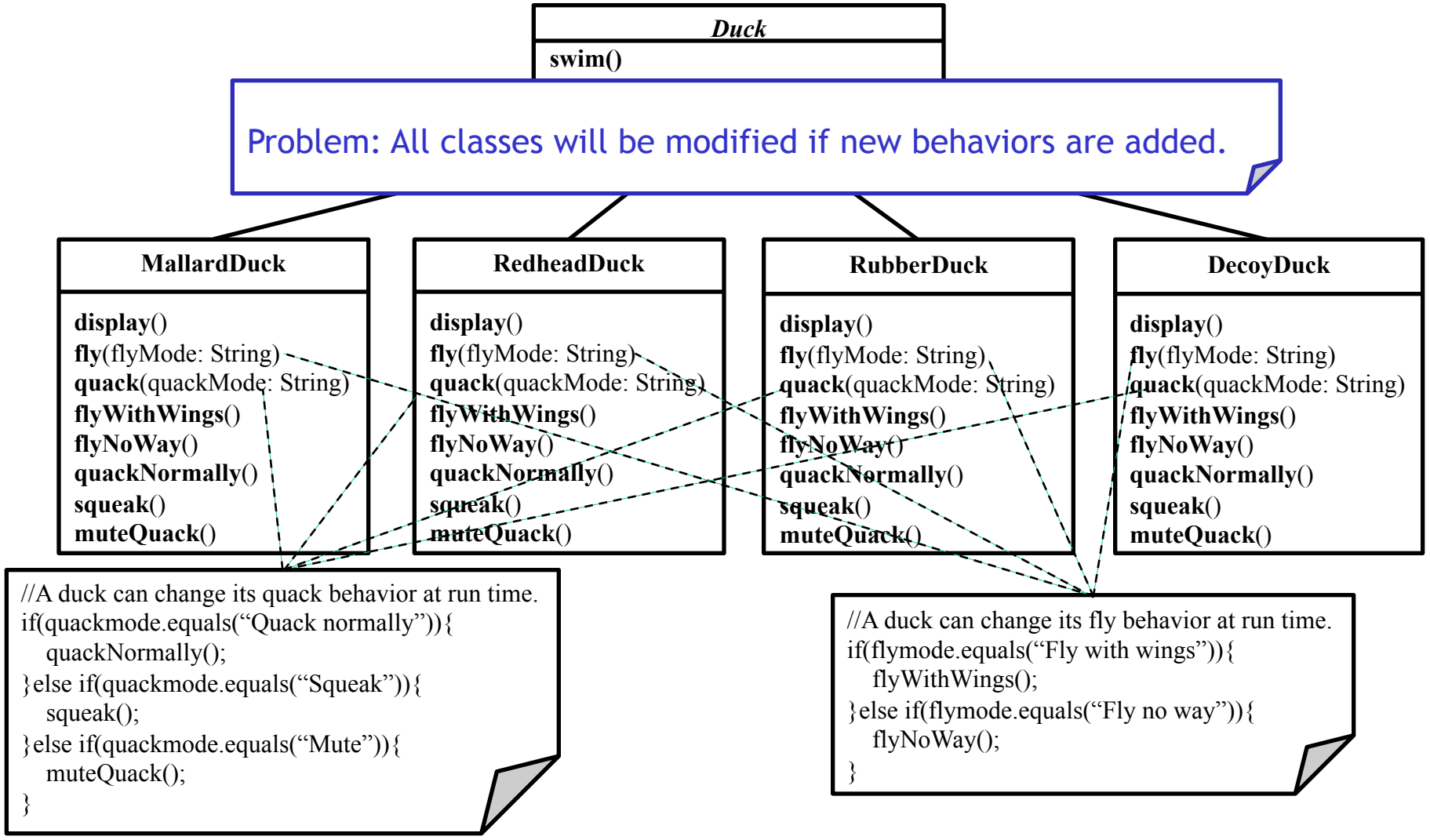

```
//A duck can change its quack behavior at run time.
if(quackmode.equals("Quack normally")){
   quackNormally();
}else if(quackmode.equals("Squeak")){
   squeak();
}else if(quackmode.equals("Mute")){
   muteQuack();
}
```

```
//A duck can change its fly behavior at run time.
if(flymode.equals("Fly with wings")){
   flyWithWings();
}else if(flymode.equals("Fly no way")){
   flyNoWay();
}
```

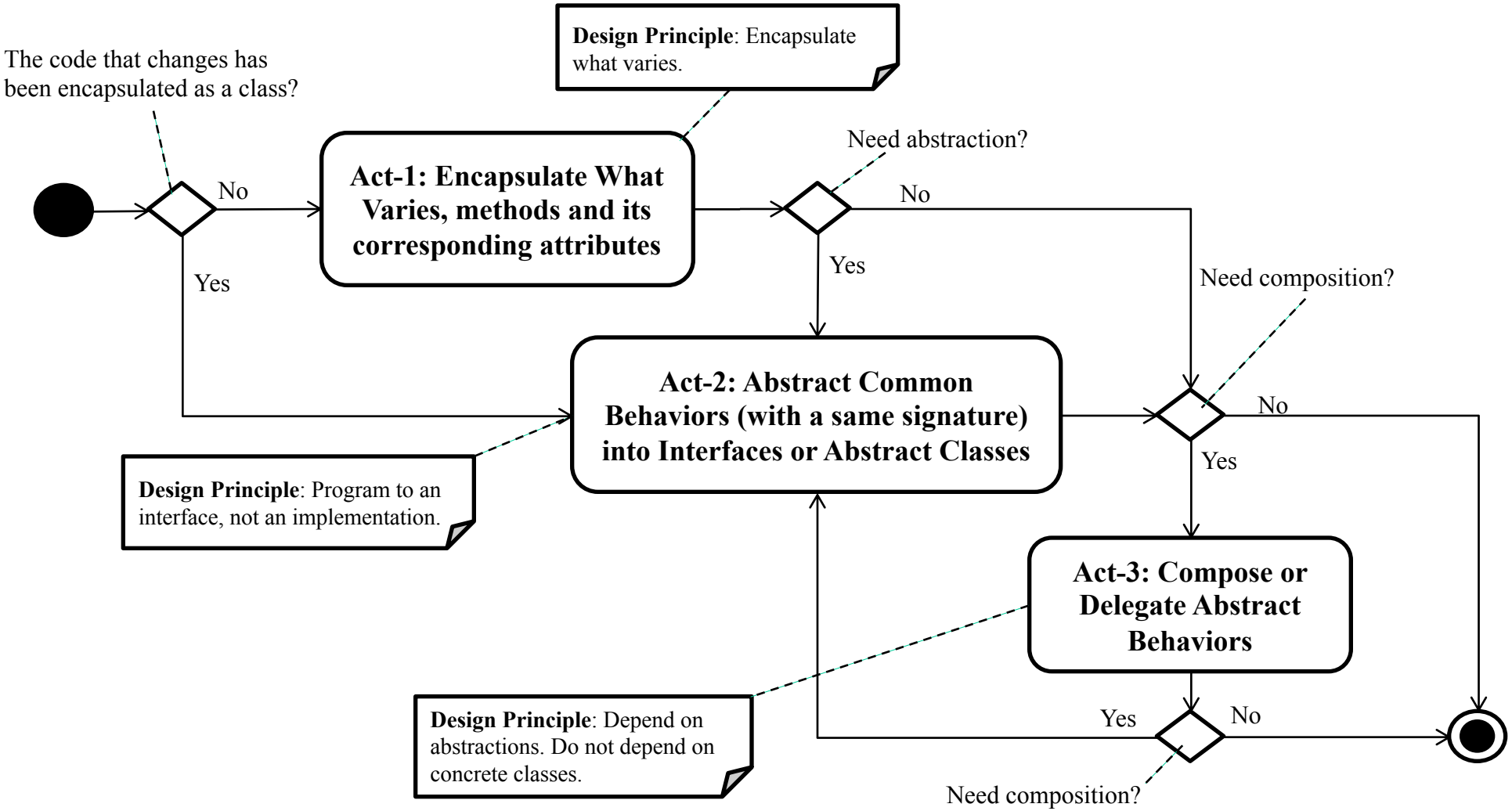
Initial Design - Class Diagram



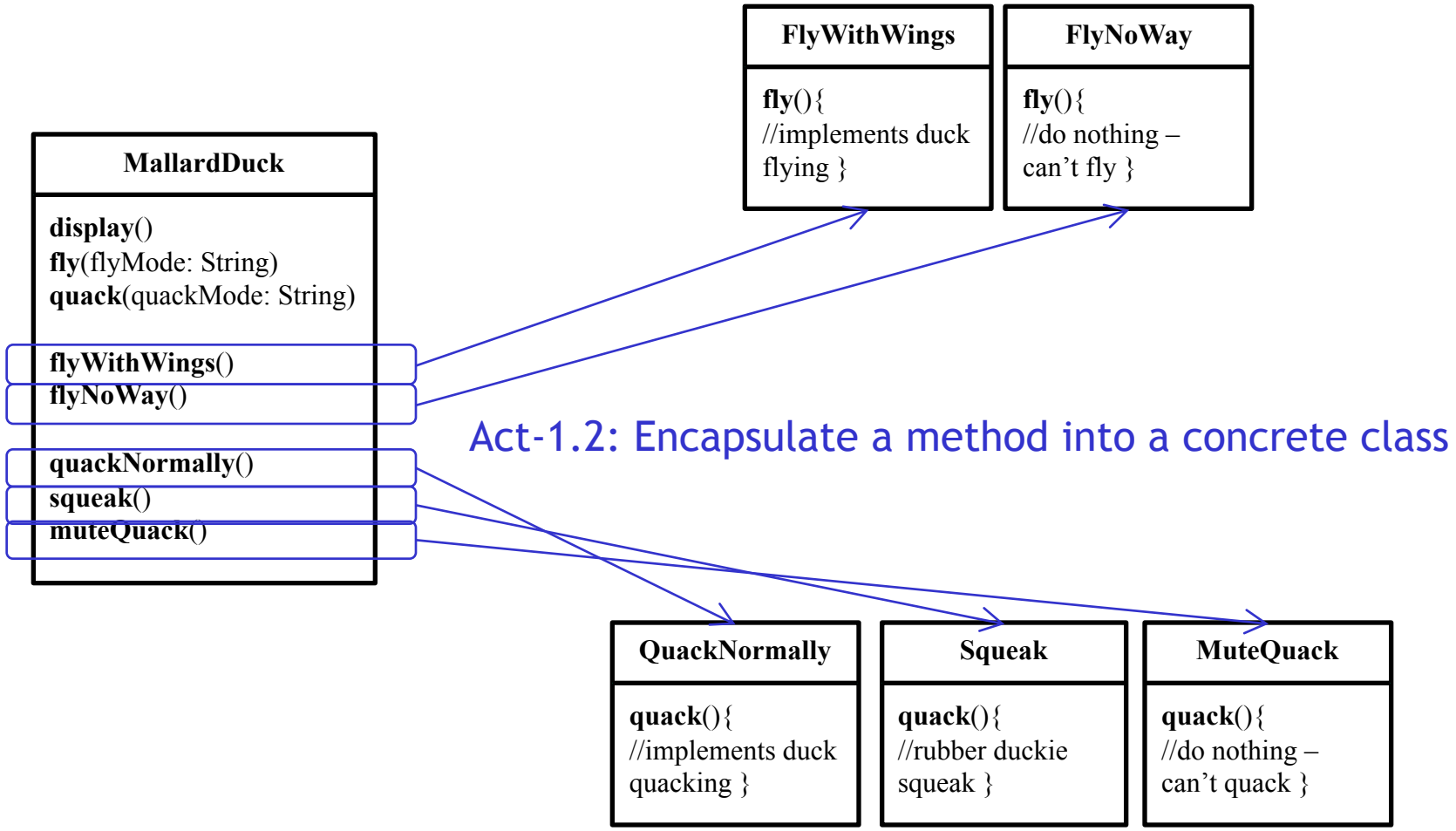
Problem with Initial Design



Design Process for Change

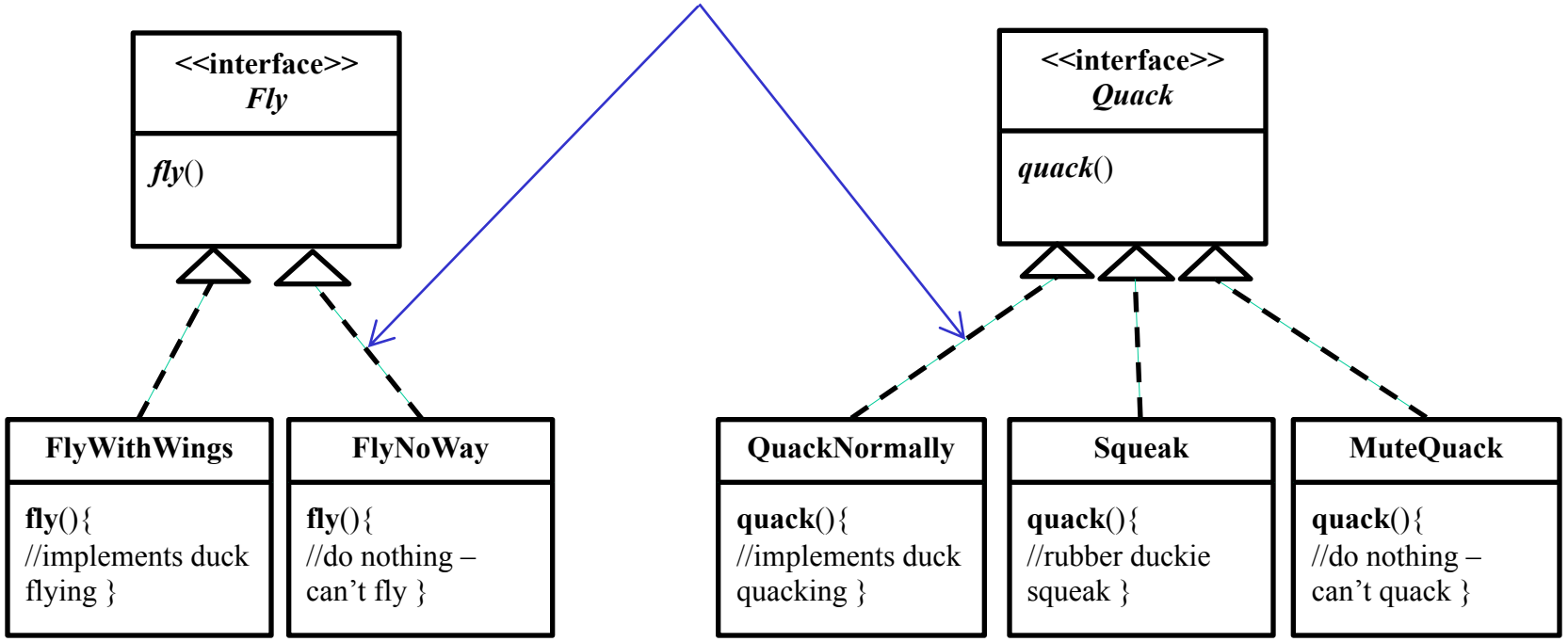


Act-1: Encapsulate What Varies



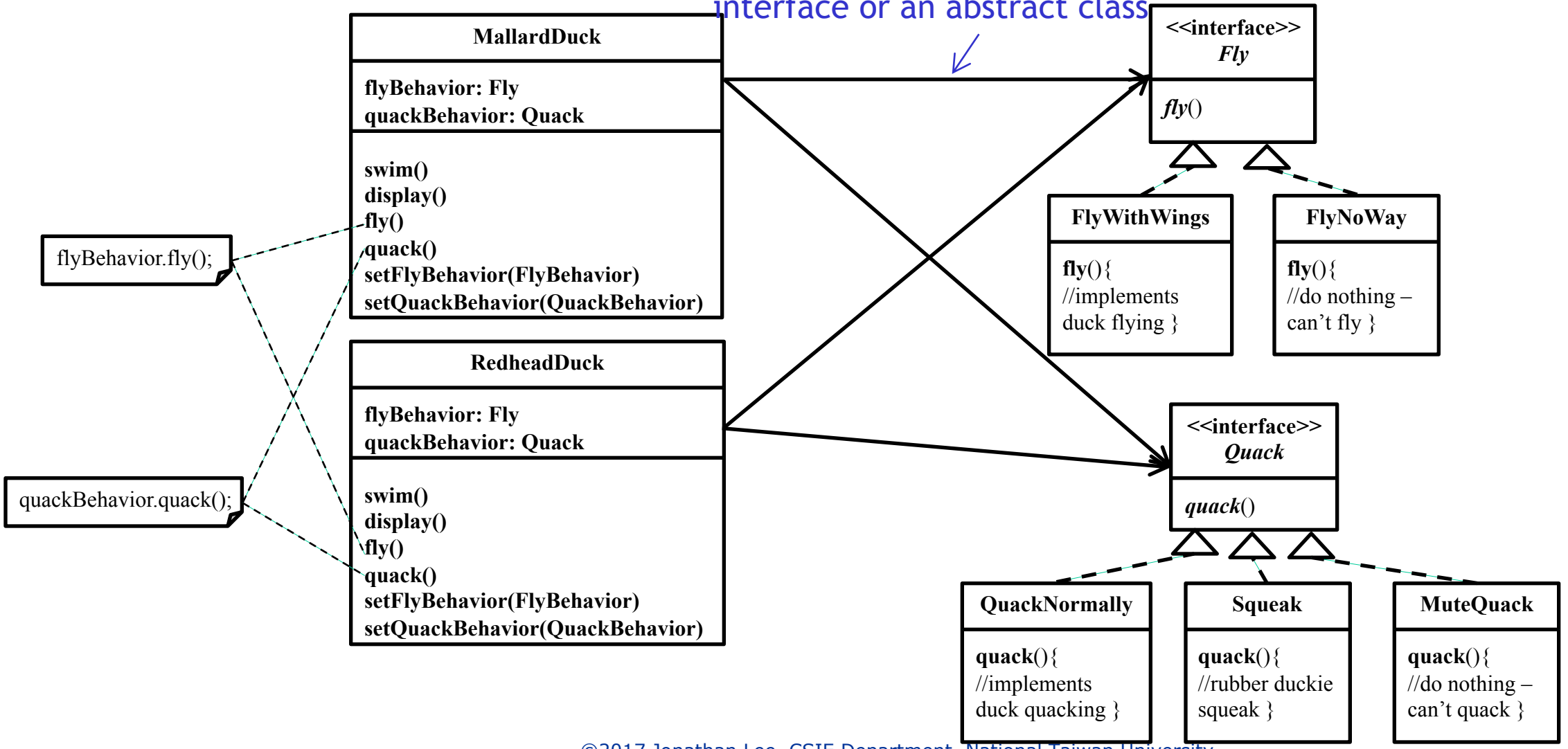
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



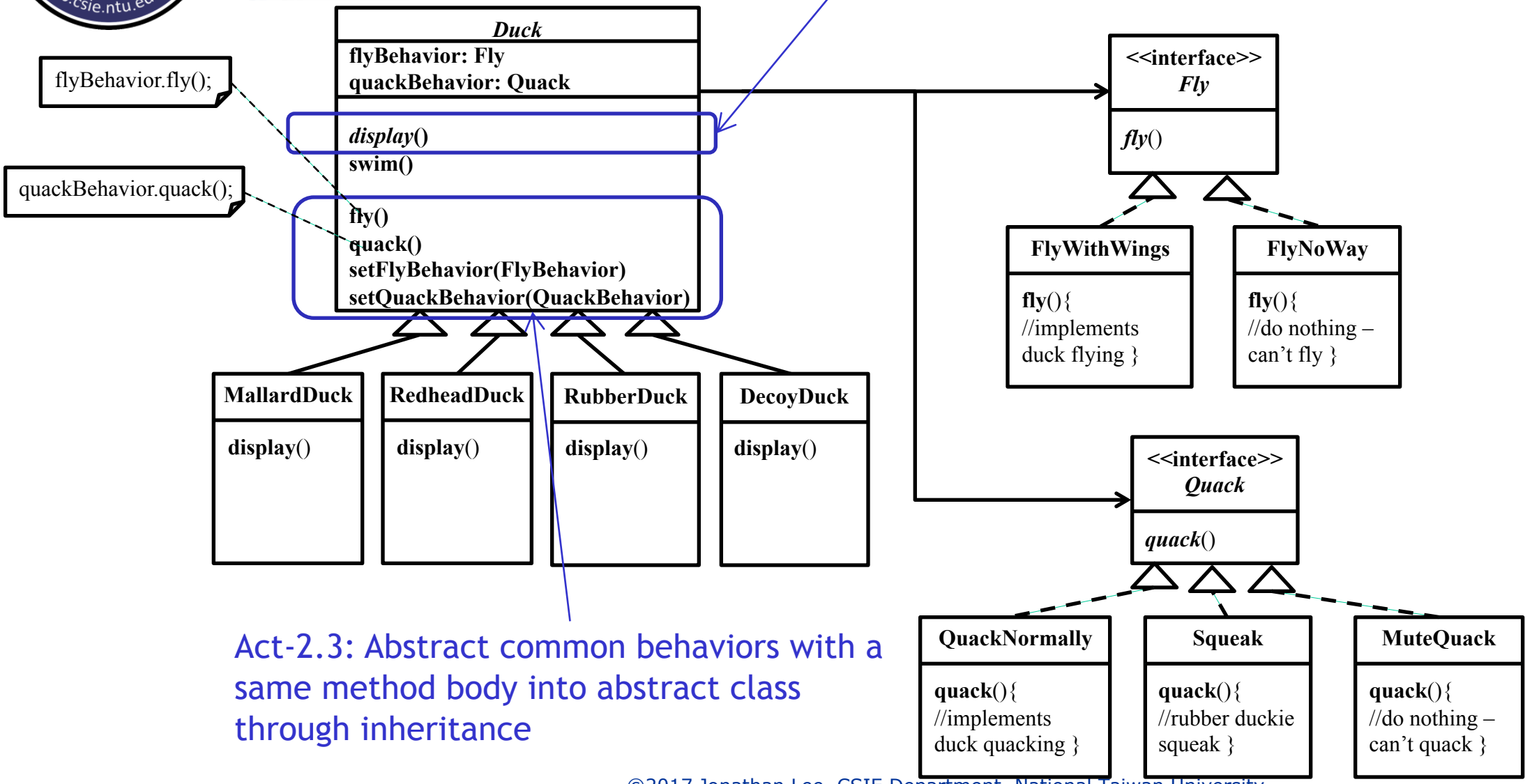
Act-3: Compose Abstract Behaviors

Act-3.1: Compose behaviors of an interface or an abstract class



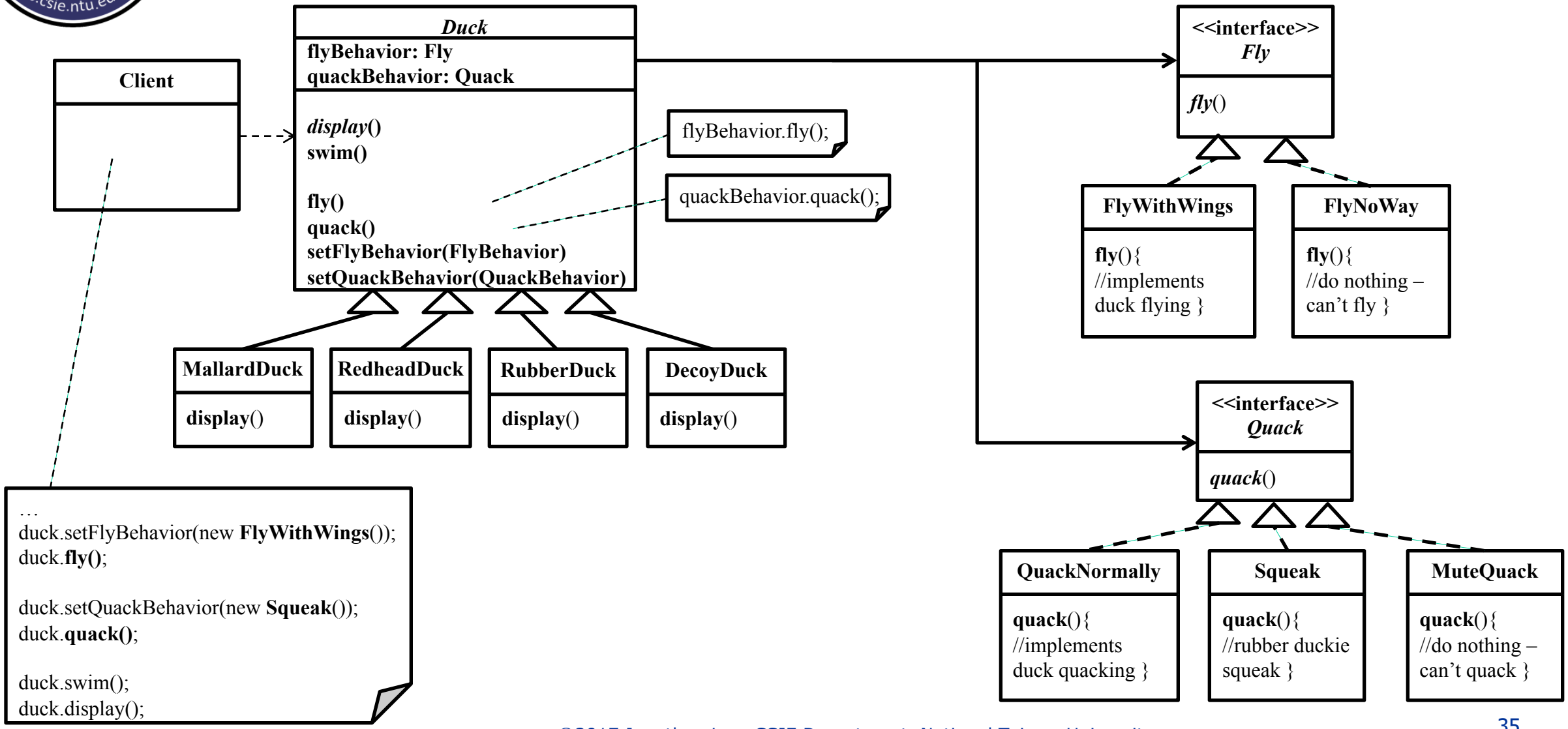
Act-2: Abstract Common Behaviors

Act-2.2: Abstract common behaviors with a same signature into abstract class through inheritance



Act-2.3: Abstract common behaviors with a same method body into abstract class through inheritance

Refactored Design after Design Process



Homework 1: Requirements Statements$_1$

- In a spreadsheet application,
  - A spreadsheet object, bar chart object, and pie chart object can depict information in the same application data object by using different presentations.
  - When the user changes the information in the spreadsheet, the bar chart reflects the changes immediately, and vice versa.

Homework 1: Requirements Statements$_2$

- Both a spreadsheet object and bar chart object can depict information in the same application data object by using different presentations.