Introduction to Object-Orientation

Baris Aktemur
CS 534 | Ozyegin University

What's an Object

- An object packages both data and the procedures that operate on that data.
- An object performs an operation when it receives a request (or message) from a client.

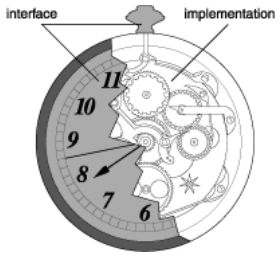
[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

Interface

- Every operation declared by an object specifies the operation's name, the objects it takes as parameters, and the operation's return value.
- This is known as the operation's **signature**.
- The set of all signatures defined by an object's operations is called the **interface** to the object.
- An object's interface says nothing about its implementation—different objects are free to implement requests differently.

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

Interface



[developer.apple.com]

Encapsulation

- Requests are the only way to get an object to execute an operation.
- Operations are the only way to change an object's internal data.
- Because of these restrictions, the object's internal state is said to be **encapsulated**
  - it cannot be accessed directly
  - its representation is invisible from outside the object.
- Objects are known only through their interfaces.

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

5

Dynamic Binding

- When a request is sent to an object, the particular operation that's performed depends on both the request and the receiving object.
- The run-time association of a request to an object and one of its operations is known as **dynamic binding**. (or dynamic dispatch)

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

11

Polymorphism

- Issuing a request doesn't commit you to a particular implementation until runtime.
- **Polymorphism**: being able to substitute objects that have identical interfaces for each other at run-time.
- Lets objects vary their relationships to each other at run-time.

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

12

Program to an Interface, not an Implementation

- Two benefits
  - Clients remain unaware of the specific types of objects they use, as long as the objects adhere to the interface that clients expect.
  - Clients remain unaware of the classes that implement these objects.
- Greatly reduce implementation dependence

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

14

Reuse Mechanisms

- (Class) Inheritance
  - White-box reuse
- Composition
  - Black-box reuse
  - Must have well-defined interfaces

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

16

Class inheritance

+ Straightforward
+ Easy to modify the implementation being reused
– Can't change the implementation dynamically
– "breaks encapsulation": superclass is exposed to subclass; changing superclass likely to force a change in subclasses

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

17

Composition

+ Strong encapsulation
+ Easily change the implementation at runtime
+ Fewer implementation dependencies
+ Keep class hierarchies manageably small
– Design interfaces very carefully
– Many more objects and relations

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

18

Favor Composition over Inheritance

19

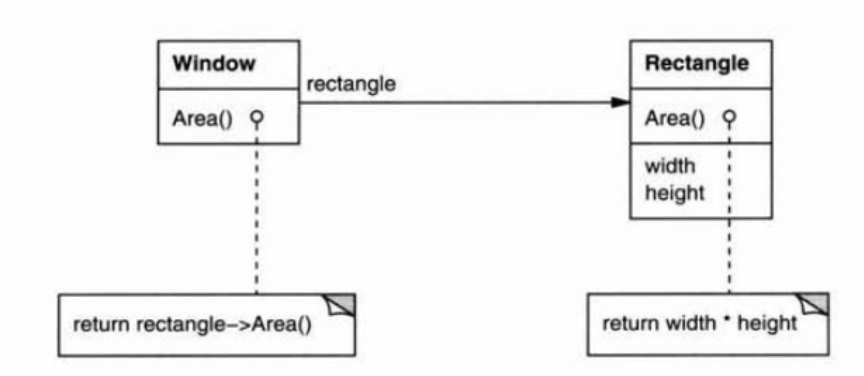
Delegation

- Achieving inheritance reuse via composition
- A receiving object delegates operations to its **delegate.**
- Receiver passes itself to the delegate
  - similar to referring to the receiver object using **this** in inheritance
- Delegation is a good design choice only when it simplifies more than it complicates.

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

20

Delegation



Can change rect to circle at runtime to make the window circular.

[Gamma, Helm, Johnson, and Vlissides. Design Patterns. Addison-Wesley, 1994*]*

21

Law of Demeter

- Governs the communication structure within an object-oriented design
  - restricts message-sending statements in method implementations
  - **only talk to your immediate friends**
- Message target can only be one of the following objects:
  - the method's object itself (C++: *this*)
  - an object that is an argument in the method's signature
  - an object referred to by the object's attribute
  - an object created by the method
  - an object referred to by a global variable

http://c2.com/cgi/wiki?LawOfDemeter

31