Objects, UML, and Java

Suleman Shahid
suleman.shahid@lums.edu.pk

Abdul Ali Bangash
abdulali@lums.edu.pk

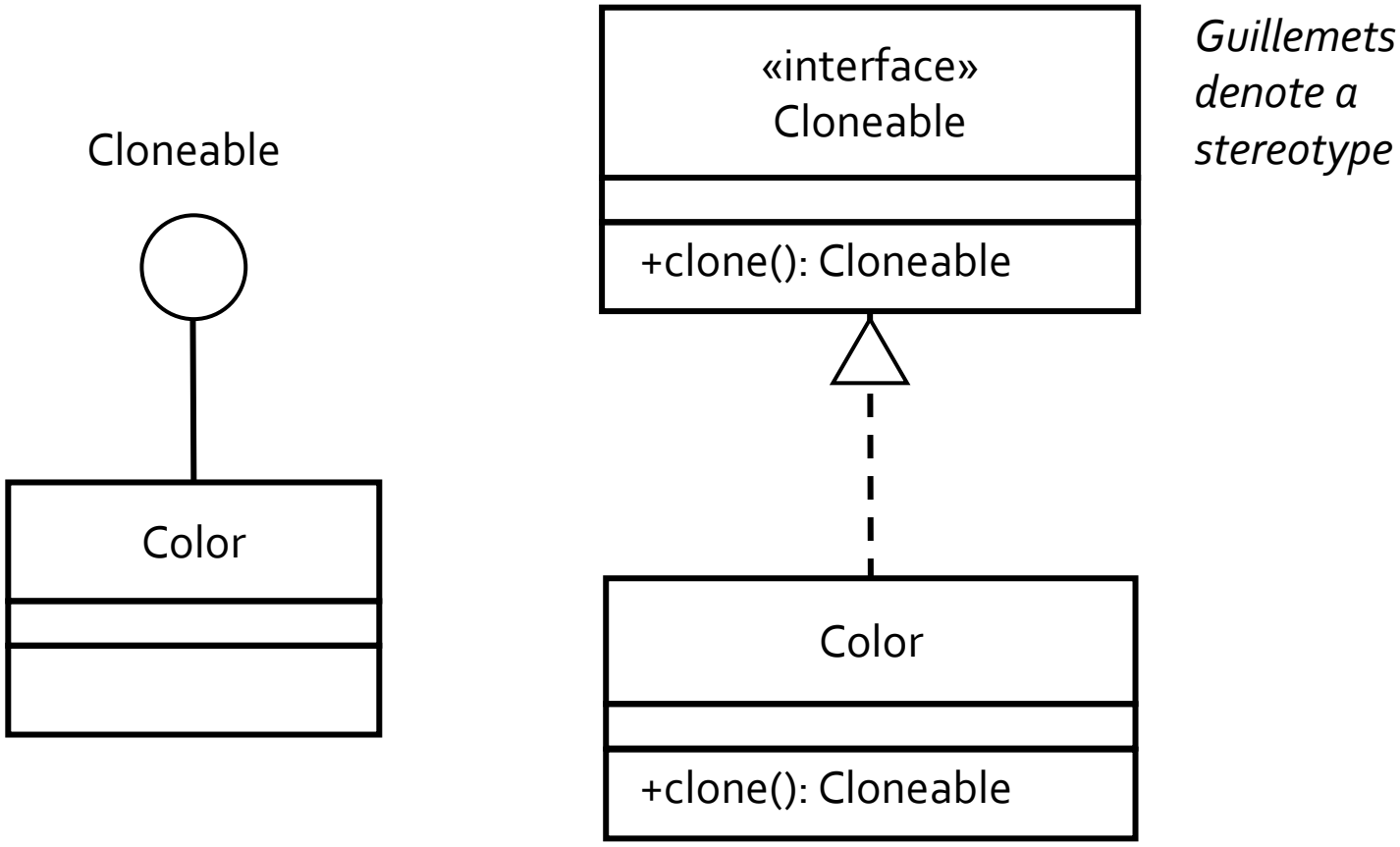
Department of Computing Science

LUMS University

CMPUT 360 – Introduction to Software Engineering

Slides adapted from Henry Tang, Dr. Abram Hindle

UML Interface

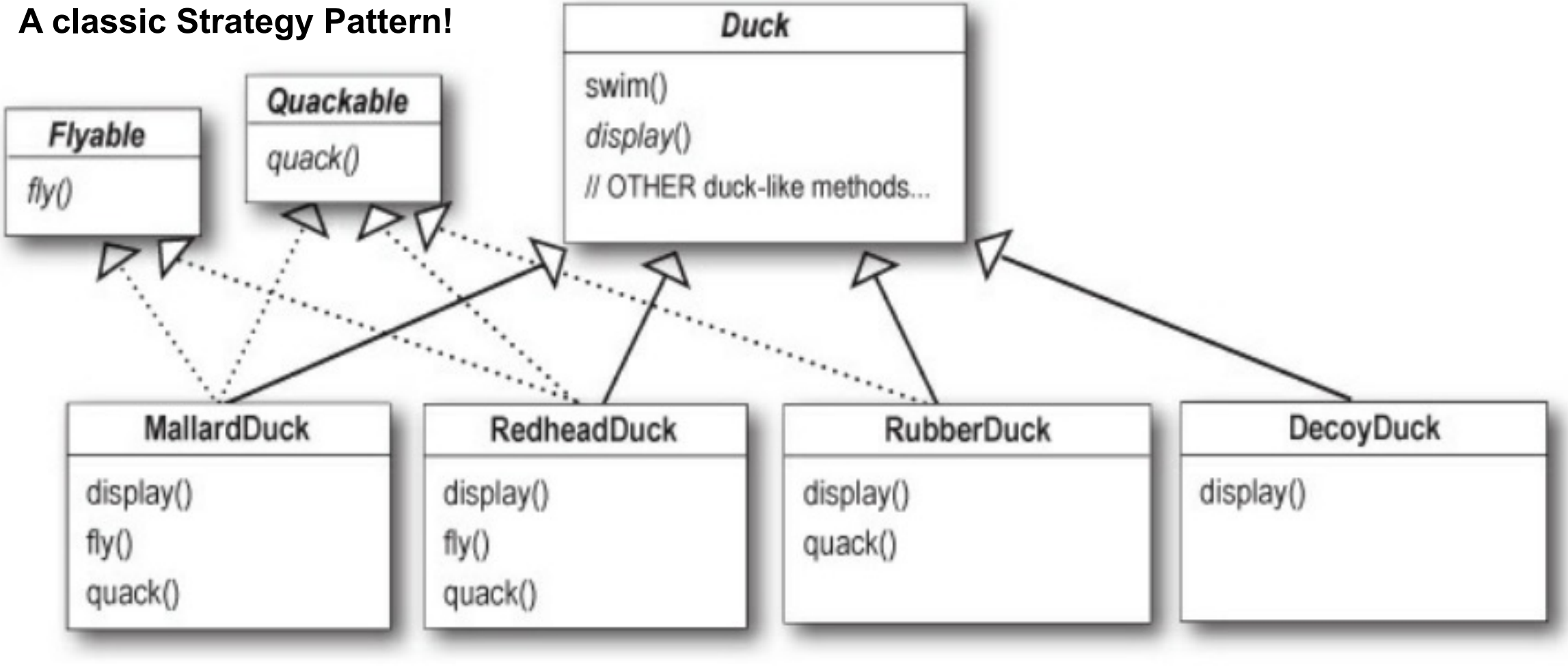


Guillemets denote a stereotype

Abstract Class versus Interface

- Differences:
  - An abstract class may provide a partial implementation
  - A class may implement any number of interfaces, but only extend one superclass
  - Adding a method to an interface will “break” any class that previously implemented it

A classic Strategy Pattern!

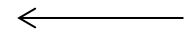


Java Subtleties

Java Call-by-Value

- ```
  public class Sender {
      …
      public void send() {
          Receiver r = new Receiver();
          Info argRef = new Info();

          r.receive( argRef );
          argRef.doSomeMore();
      }
  }
  ```

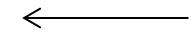


- ```
  public class Receiver {
      …
      public void receive( Info infoRef ) {
          infoRef.doSomething();
          infoRef = null;
      }
  }
  ```

Java Call-by-Value

- ```
  public class Sender {
      …
      public void send() {
          Receiver r = new Receiver();
          Info argRef = new Info();
          r.receive( argRef );
          argRef.doSomeMore();
      }
  }
  ```
  Info object created

  Two references. Same object.

- ```
  public class Receiver {
      …
      public void receive( Info infoRef ) {
          infoRef.doSomething();
          infoRef = null;
      }
  }
  ```
  Info object (copy of the reference)

  This only changes the local copy of the reference.

Java Constructors

- ```
  public class Base {
      protected int value;
      public Base() {

          value = -1;
      }
  }
  ```

- ```
  public class Derived extends Base {
      public Derived() {

      }
  }
  ```

- ```
  Derived d = new Derived();
  ```
  ```

Java Constructors

- ```
  public class Base {
      protected int value;
      public Base() {
          // implicitly inserted call to super()
          value = -1;
      }
  }
  ```

- ```
  public class Derived extends Base {
      public Derived() {
          // implicitly inserted call to super()
      }
  }
  ```

- ```
  Derived d = new Derived();
  ```
  ```

Java Constructors

- ```
  public class Base {
      protected int value;
      public Base( int initValue ) {
          // implicitly inserted call to super()
          value = initValue;
      }
  }
  ```

- ```
  public class Derived extends Base {
      public Derived( int initValue ) {
          super( initValue );
          // explicit call to super( … );
          // super( … ) if used, must come first
      }
  }
  ```

- ```
  Derived d = new Derived( -1 );
  ```
  ```

Java Constructors

- ```
  public class Base {
      protected int value;
      public Base( int initValue ) {
          // implicitly inserted call to super()
          value = initValue;
      }
      public Base() {
          this( -1 );
          // this( … ) if used, must come first
      }
  }
  ```
- ```
  public class Derived extends Base {
      public Derived( int initValue ) {
          super( initValue );
      }
      public Derived() {
          // implicitly inserted call to super()
      }
  }
  ```
- ```
  Derived d = new Derived();
  ```
  ```

Java Shadowing Data

- ```
  public class Base {
      protected int value; // 2, 3
  }
  ```

- ```
  public class Derived extends Base {
      private int value; // 0, 1

      public void test() {
          value = 0;
          this.value = 1;
          super.value = 2;
          ((Base)this).value = 3;
      }
  }
  ```

Java Dynamic Binding

```
public class Base {
    // default implementation
    public void op() { … }
}
```

```
public class Derived1 extends Base {
    // does not override op()
}
```

```
public class Derived2 extends Base {
    // override ...
    @Override
    public void op() { … }
}
```

Selection of method to be run is made at run time, depending on type of receiving object

```
Base base;
base = new Derived1();  // implicit upcast
base.op();              // calls op() in Base
base = new Derived2();  // implicit upcast
base.op();              // calls op() in Derived2
```

Receiving object does the "right thing", even if the calling code does not show its actual type

Overriding Is Not Shadowing

- ```
  public class Base {
      public int i = 1;
      public int f() { return i; }
  }
  ```
- ```
  public class Derived extends Base {
      public int i = 2;                 // shadowing
      public int f() { return -i; }     // overriding
  }
  ```
- ```
  public class Test {
      public static void main( String args[] ) {
          Derived d = new Derived();
          // d.i is 2
          // d.f() returns -2
          Base b = (Base)d;
          // b.i is 1
          // b.f() returns -2, 'dynamic binding'
      }
  }
  ```
  ```

Object-Oriented Analysis and Design

UML and OOA&D

- Analysis:
  - Requirements specification activity
    - Create UML use cases and class diagrams
- Design:
  - Architectural design activity
    - Refine UML class diagrams
  - Detailed design activity
    - Refine UML class diagrams
    - Create UML sequence and state diagrams

Object-Oriented Analysis

- Steps:
  - Discover objects from problem domain
    - Nouns may lead to classes and attributes
    - Verbs may lead to relationships and methods
  - Use CRC cards to note the analysis
  - Evaluate

Problem Description

- The library has books and magazines. Books may be borrowed by any patron for four weeks while magazines may only be borrowed for two days. Up to six items at a time may be borrowed. The system tracks when books and magazines are borrowed.

Nouns

- The library has books and magazines. Books may be borrowed by any patron for four weeks while magazines may only be borrowed for two days. Up to six items at a time may be borrowed. The system tracks when books and magazines are borrowed.

Verbs

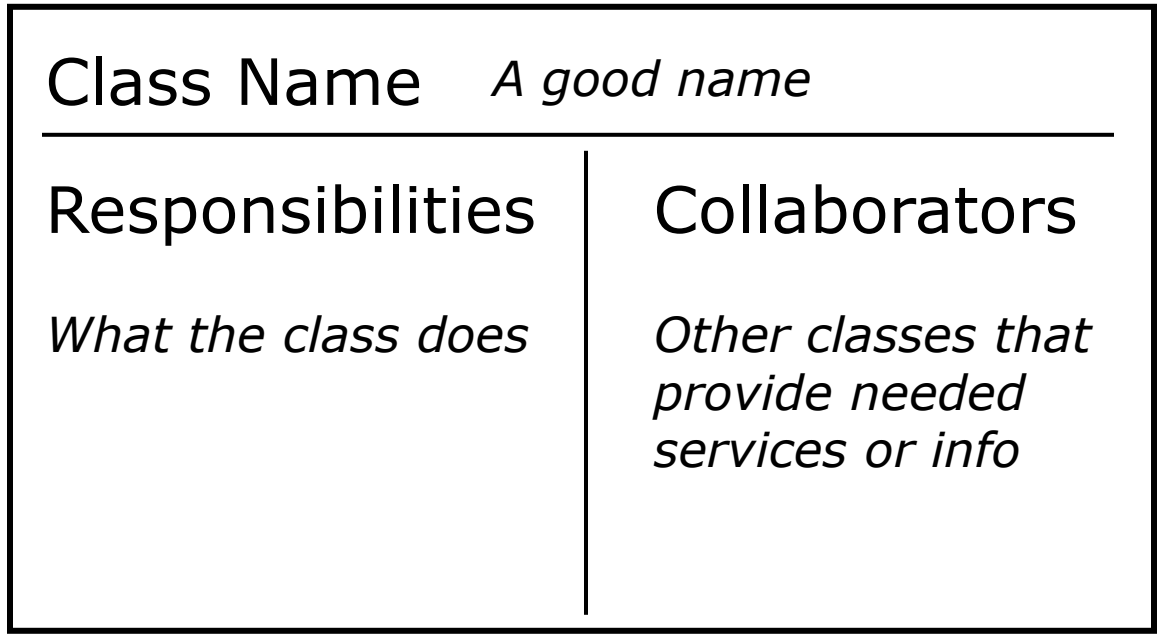
- The library has books and magazines. Books may be borrowed by any patron for four weeks while magazines may only be borrowed for two days. Up to six items at a time may be borrowed. The system tracks when books and magazines are borrowed.

Discover Objects

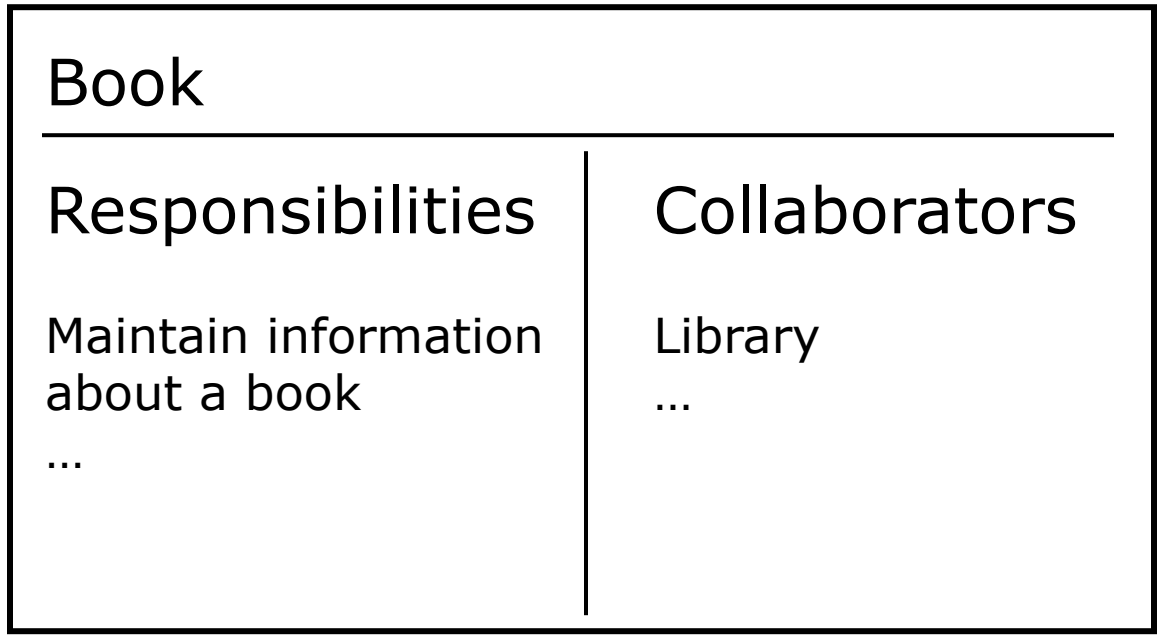
- Entity objects:
  - Things that model the problem domain
- Control objects:
  - Things that respond to events and coordinate services
- Boundary objects:
  - Things that interact with the system
    - E.g., other applications, devices, sensors, actors, roles, windows, forms

Use CRC Cards

- Class-Responsibility-Collaborator
  - Explore classes, their responsibilities, and their interactions
  - Organize index cards on a table



Use CRC Cards



Use CRC Cards

- Role playing:
  - Refine the cards by acting out a particular scenario with the candidate objects
  - "Become" the object
    - What do I do?
    - What do I need to remember?
    - With whom do I need to interact?
    - How do I respond?

Evaluate

- Principles:
  - During analysis, objects should initially be technology independent
  - If an object has only one attribute, perhaps it should not be a separate object at all
  - If an object has several highly related attributes, perhaps these attributes should form a separate object

Guidelines

- Get the big picture:
  - Understand the problem
    - Talk to the customer, end users, domain experts
  - Understand the target environment
    - Know the implementation constraints
  - Avoid reinventing the wheel
    - Reuse designs

Guidelines

- Modularity:
  - Increase cohesion
    - Class has a clear specific responsibility
  - Reduce coupling
    - Class is not connected to or knows too many others
  - Separate the layers
    - Identify entity, control, and boundary objects
    - Allow replacing layers

Guidelines

- Classes:
  - Use good names
    - Should be meaningful and explanatory
  - Avoid huge "blob" classes
    - A single class should not do everything
  - Use information hiding
    - Hide changeable details, reveal assumptions

Guidelines

- Generalization:
  - Find superclasses
    - Look for and factor commonalities among classes
  - Apply Liskov principle for proper inheritance
    - Or use “is-a” test
  - “Is-a” test is not always enough
    - Class names can mislead, look at specific behaviour

Guidelines

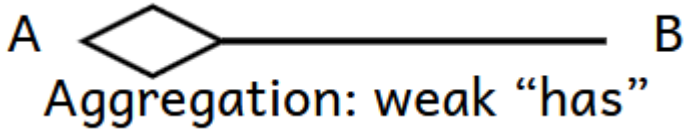
- Adaptation:
  - Hard to get it right the first time
    - Recognize problems and fix them
  - Your software won't go away
    - Make it easy to adapt to change
  - Simplicity (as simple as possible)
    - Does not always mean using the first thing that comes to mind
    - Elegant designs may need effort

Exercise (Take Home)

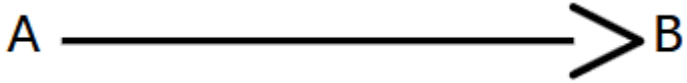
- Chess is a two-player, turn-based game where players move pieces on an 8x8 board. Each player begins a game with 16 initial pieces: 1 king, 1 queen, 2 rooks, 2 bishops, 2 knights, and 8 pawns. One player has the white pieces, and the other player has the black pieces. Pieces of different types move in specific ways or situations (e.g., navigating directionally, capturing an opponent's piece, castling, promoting, etc.).
- Consider a chess tutorial app, where a learner can tap on a piece and see its possible moves from which to choose. A learner can make a move upon the board. After making successive move(s), the learner can take back move(s).
- Using CRC cards, identify the key entity classes (and their generalizations as appropriate) in a well-designed, object-oriented model for this app. For each class, give it a good name, outline its main responsibilities, and list its collaborators.

Self-reading material beyond this slide.

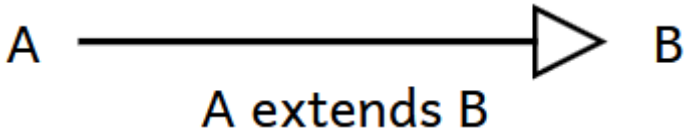
Association
Usually two-way navigability,
but sometimes just not specified

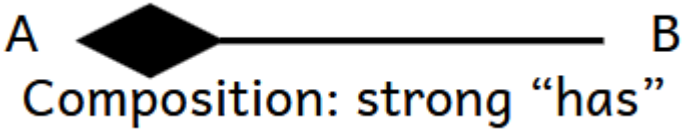


A has some Bs
The same Bs that A has can
be shared
Not exclusive!



One-way Navigability
We can only follow references from A to B





B is a part of A:
When instance of A is deleted,
all of its B are also deleted

More Information

- Books:
  - The Essence of Object-Oriented Programming with Java and UML
    - B. Wampler
    - Addison-Wesley, 2002
  - Java in a Nutshell
    - D. Flanagan
    - O'Reilly, 2005

More Information

- Books:
  - UML Distilled
    - M. Fowler
    - Addison-Wesley, 2003
  - The Elements of UML 2.0 Style
    - S. W. Ambler
    - Cambridge, 2005

More Information

- Link:
  - UML Quick Reference
    - http://www.holub.com/uml/
- Books:
  - Book: Developing applications with Java and UML
  - Book: Java Design: Objects, UML and Process
  - Book: Object-oriented design with UML and Java