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

Generalization

Generalization

- Look for commonalities:
  - Common attributes
    - E.g., all vehicles have…
  - Common methods (behaviour)
    - E.g., all vehicles can…
- Generalize:
  - Find what is common and factor it out into a more general “base” abstraction

Generalization

- Implementation inheritance:
  - Generalize about method signatures, method implementations, and/or attributes
    - I.e., classes having these in common

Implementation Inheritance

- General part:
  - A base class (or "superclass") defines the attributes and methods to be shared
- Specific part:
  - A derived class (or "subclass") is endowed with the attributes and methods of its base class
  - A subclass may "extend" a superclass by adding attributes and methods, or overriding an existing method

Java Implementation Inheritance

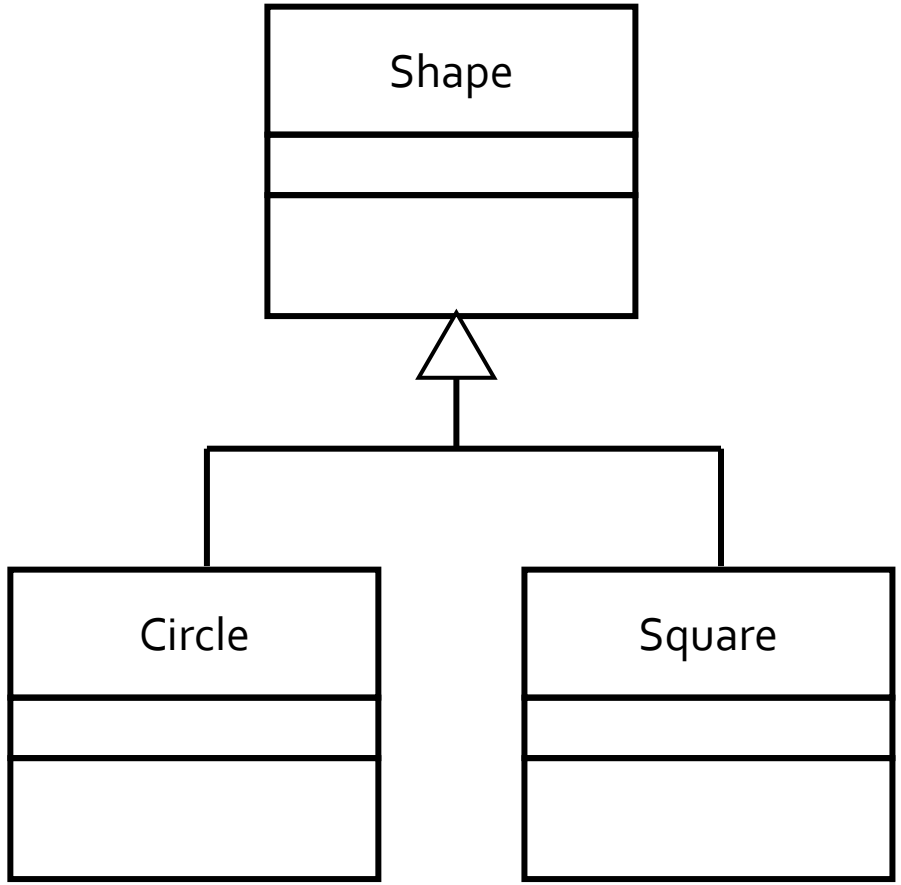
- ```
  public class Shape { // superclass
      protected Location myLocation;
      public Shape() { … }
      public void setLocation( Location p ) { … }
      public Location getLocation() { … }
  }
  ```

- ```
  public class Circle extends Shape { // subclass
      private int diameter;
      public Circle() { … }
      public void setDiameter( int d ) { … }
      …
  }
  ```

- ```
  public class Square extends Shape { // subclass
      private int side;
      public Square() { … }
      public void setSide( int s ) { … }
  }
  ```
  ```

UML Inheritance

- Implementation inheritance relationship:
  - "Is-a" relationship between classes
  - I.e., subclass "is-a" kind of superclass
  - I.e., subclass "extends" superclass
- E.g., Circle "is-a" kind of Shape



Generalization Principles

- Inappropriate inheritance:
  - Subclass inherits from superclass but "is-a" (is a kind of) relationship *does not* exist
  - If "is-a" test fails:
    - Likely not appropriate
  - If "is-a" test succeeds:
    - *May* or *may not* be appropriate

Generalization Principles

- Liskov substitution principle:
  - An instance of the subclass should be substitutable anywhere a reference to a superclass object is used
  - ```
    Shape s;
    s = new Circle(); // instance of subclass
    …
    Location l = s.getLocation(); // superclass method
    ```
    ```

Generalization Principles

- Liskov substitution principle:
  - An instance of the subclass should be substitutable anywhere a reference to a superclass object is used
  - ```
    Shape s;
    s = new Circle(); // instance of subclass
    …
    Location l = s.getLocation(); // superclass method
    ```

PollEv.com/blesseddawn004
    ```

Inheritance Example

- Suppose:
  - class Dog
    - Provides bark(), fetch()
  - class Cat extends Dog
    - “Hides” bark(), “hides” fetch(), and adds purr()
- Question:
  - Cat “is a” Dog?

Inheritance Example

- Suppose:
  - class Window
    - Provides show(), move(), resize()
  - class FixedSizeWindow extends Window
    - “Hides” resize()
- Question:
  - FixedSizeWindow “is a” Window?

Inheritance Issue

- Problem:
  - Superclass method is inherited, but it is not appropriate
  - What to do?

Inheritance Issue

- ```
  public class Rectangle {
      …
      public Rectangle( Size s ) { … }
      public void setLocation( Location p ) { … }
      public void setSize( Size s ) { … }
      public void draw() { … }
      public void clear() { … }
      public void rotate() { … }
      …
  }
  ```

- ```
  public class Square extends Rectangle {

      // inherits setSize(), but want to "hide" it
  }
  // Square 'is a' Rectangle?
  // Square specializes Rectangle?
  ```

Override the Method Approach

- ```
  public class Square extends Rectangle {
      @Override
      public void setSize( Size s ) {
          // should not implement
      }

  }
  ```
  ```

Aggregation Approach

```
public class Square {
    private Rectangle rect;
    // Square 'has a' Rectangle,
    // not 'is a' Rectangle

    public Square( int side ) {
        rect = new Rectangle(
            new Size( side, side ) );
    }
    …
    public void setSide( int newSide ) {
        rect.setSize(
            new Size( newSide, newSide ) );
    }

    public void draw() {
        rect.draw();
    }
    …
}
```

Restructuring Approach

- ```
  public class Quadrilateral {
      …
      public Quadrilateral() { … }
      public void setLocation( Location p ) { … }
      public void draw() { … }
      public void clear() { … }
      public void rotate() { … }
  }
  ```

- ```
  public class Rectangle extends Quadrilateral {
      …
      public Rectangle( Size s ) { … }
      public void setSize( Size s ) { … }
  }
  ```

- ```
  public class Square extends Quadrilateral {
      …
      public Square( int side ) { … }
      public void setSide( int side ) { … }
  }
  ```
  ```

Inheritance

- Java abstract class:
  - Declares one or more abstract methods
  - Cannot be instantiated; must be subclassed and have abstract methods overridden

- ```
  public abstract class Shape {
      public abstract double area();
      public abstract double perimeter();
      // there may be other instance data and methods
  }
  ```
- ```
  public class Circle extends Shape {
      public Circle() { … }
      public double area() { … }
      public double perimeter() { … }
  }
  ```

Interface Inheritance

- Java interface:
  - Declares method signatures
  - Classes implement the interface by providing all the method bodies

```
public interface Bordered {
    public double area();
    public double perimeter();
}
```

```
public class Circle implements Bordered {
    public Circle() { … }
    public double area() { … }
    public double perimeter() { … }
}
```

Interface Inheritance

- Java interface:
  - A "contract", specifying a *capability* that an implementing classes must provide
  - Gives method signatures, but typically no implementation
  - Cannot be instantiated
  - May extend other (sub)interfaces

- ```
  public interface Transformable extends Scalable, Translatable, Rotatable {
      …
  }
  ```
  ```

Java Interface

- ```
  public interface Cloneable {
      public Cloneable clone();
  }
  ```

- ```
  public class Color implements Cloneable {
      private int red;
      private int green;
      private int blue;

      public Color( int r, int g, int b ) { … }

      public Cloneable clone() {
          return new Color( red, green, blue );
      }
  }
  ```

- ```
  Color red = new Color( 255, 0, 0 );
  Cloneable redClone = red.clone();
  Cloneable redClone2 = redClone.clone();
  ```
  ```

Java Interface

- ```
  public interface Cloneable {
      public Cloneable clone();
  }
  ```
  "clone() returns some object whose type implements Cloneable"

- ```
  public class Color implements Cloneable {
      private int red;
      private int green;
      private int blue;

      public Color( int r, int g, int b ) { … }

      public Cloneable clone() {
          return new Color( red, green, blue );
      }
  }
  ```
  Returning it as Cloneable is upcasting
  // Color → Cloneable (upcast)

- ```
  Color red = new Color( 255, 0, 0 );
  Cloneable redClone = red.clone();
  Cloneable redClone2 = redClone.clone();
  ```
  Color object created
  return new Color(red, green, blue);
  ```