Abstract Classes, Polymorphism, and Inheritance

Learning Goals

- See examples of object-oriented analysis and design
- Demonstrate the need for abstract classes and abstract methods
- Give a detailed example of polymorphism
- Show how to use inheritance to pull out a common parent class
- Give an example of using a protected method
- Explain the purpose of an interface and show one is used

What makes Java object-oriented?

- Has encapsulation
  - Objects and classes
- Has inheritance
  - Object fields and methods are inherited from a parent class
- Has polymorphism
  - The method that is called at run-time depends on the run-time type of the object

Object-Oriented Analysis

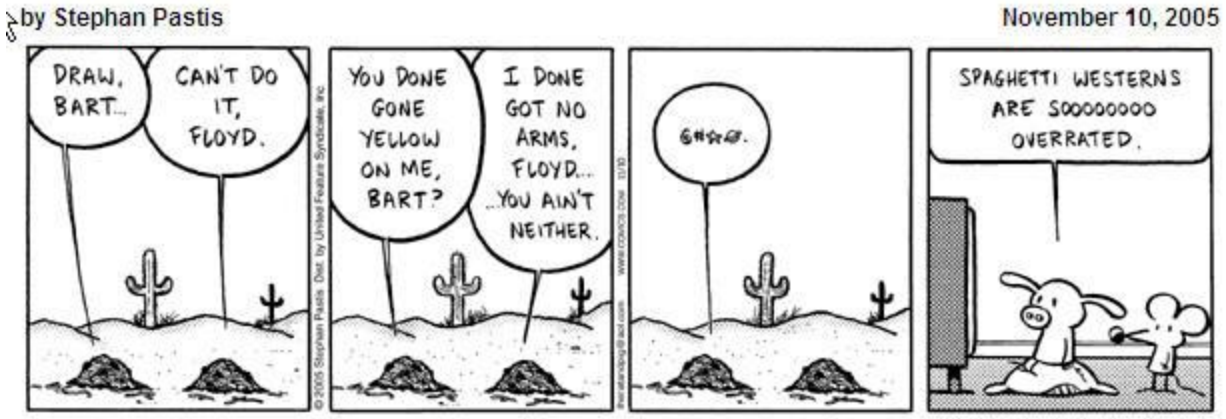
- Software engineers do analysis to determine
  - What objects are needed?
    - What data (fields) and what behaviors (methods) do the objects need?
  - How are the objects classified (the classes)?
  - What are the relationships between the classes?

Example

- If you were creating software to handle
  - Student registration
    - You would need to understand
      - Students, Courses, Teachers,
      - Classrooms, Class Periods (or Sections)
  - The diagnosis of a stroke
    - You would need to understand
      - Signs and symptoms,
      - Patients,
      - Doctors,
      - Tests,
      - Findings,

What do we mean by a comic strip?

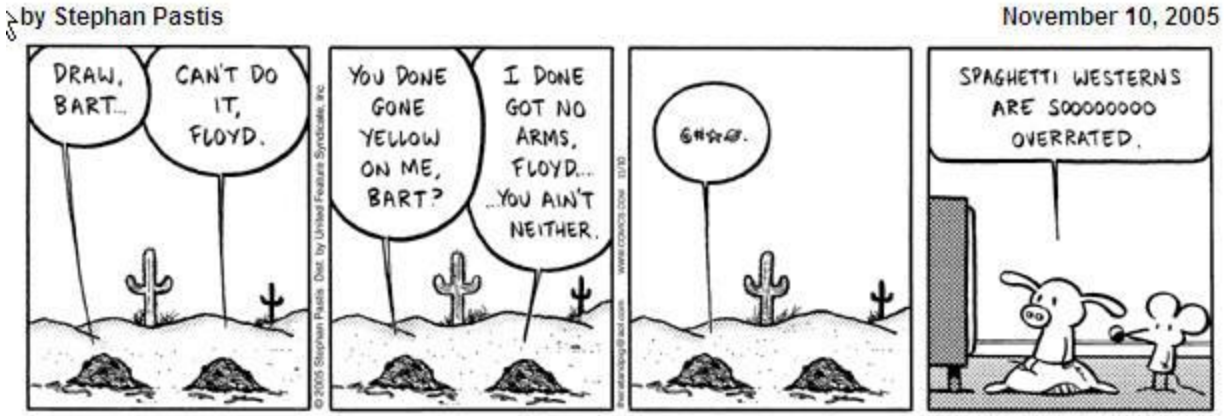
- What parts does it have?
- What things should you be able to find out about one?



A Comic Strip

- Has a name and an author
- Typically has 4 panels, but can have more or less

The Comic Strip: Pearls Before Swine

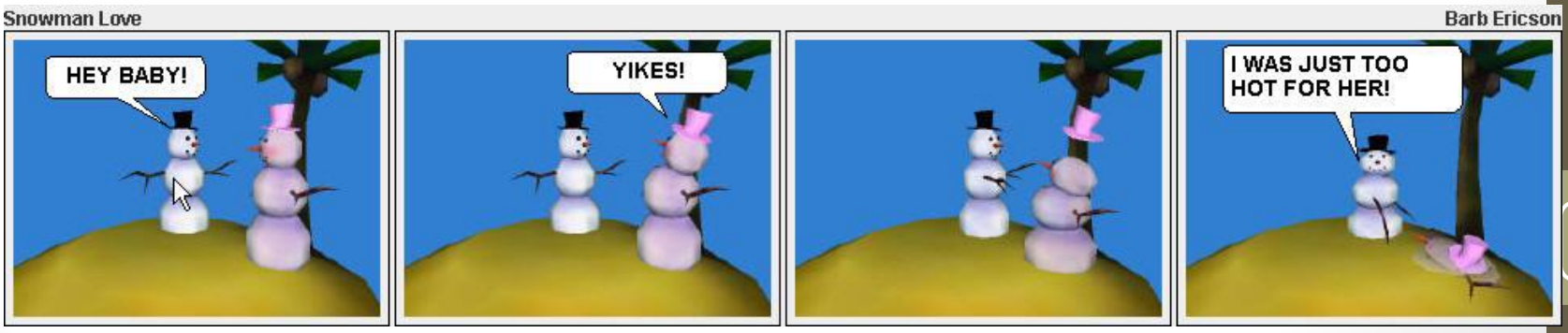


A Comic Panel

- Has a picture
- And one or more speech balloons

ComicStrip Class

- Will display a comic strip with one or more ComicPanel's in it.
- Modify the main method to create your own comic strip
  - You can use picture methods to make a black and white comic or one that looks like someone drew it in pencil

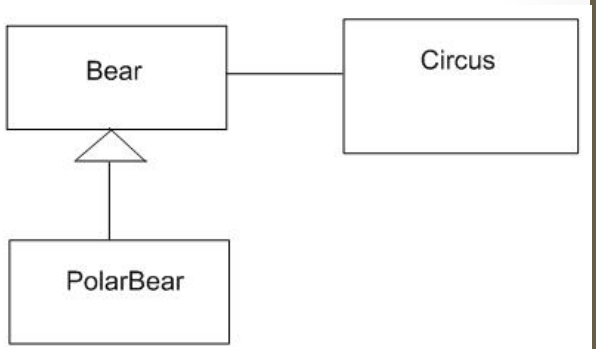


When to Use Inheritance?

- Should a comic panel have a picture as a field or inherit from the Picture class?
- It depends if they are both kind of pictures
- Does it make sense to use a comic panel anytime you would use a picture?
- A child inherits all the fields and methods of the parent class
  - So don't use inheritance if you just want access to some methods and/or fields
  - If you aren't sure use a field instead of inheritance

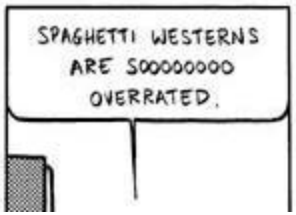
Examples of Class Relationships

- Is a Polar Bear a kind of Bear?
  - Yes, you can substitute one for the other
  - This is a "is-a" or "is-a-type-of" relationship (inheritance)
    - Shown with a triangle
- Is a circus a type of bear?
  - No, you can't substitute a bear for a circus
  - This is a "has-a" relationship (association)
    - Shown with a straight line



A Speech Balloon

- Is a rounded rectangle or ellipse with text (words) in it
- Has a tail that indicates who is speaking
  - The tail is shown as a triangle from the bottom of the rounded rectangle to the speaker

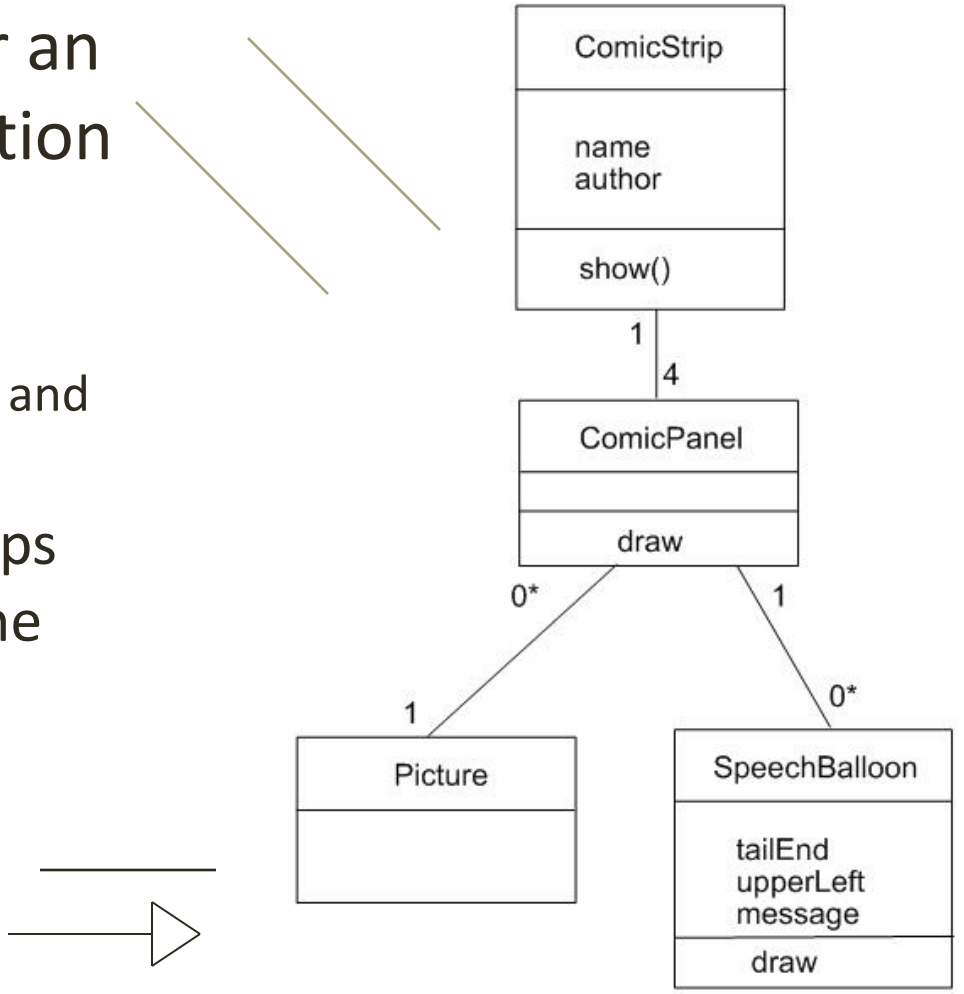


What else do we need?

- A comic strip should be able to show itself
- A comic panel needs
  - The picture
  - The speech balloons
- A speech balloon needs
  - The upper left corner for the ellipse
  - The message to draw
  - The end point for the triangle that points to the speaker

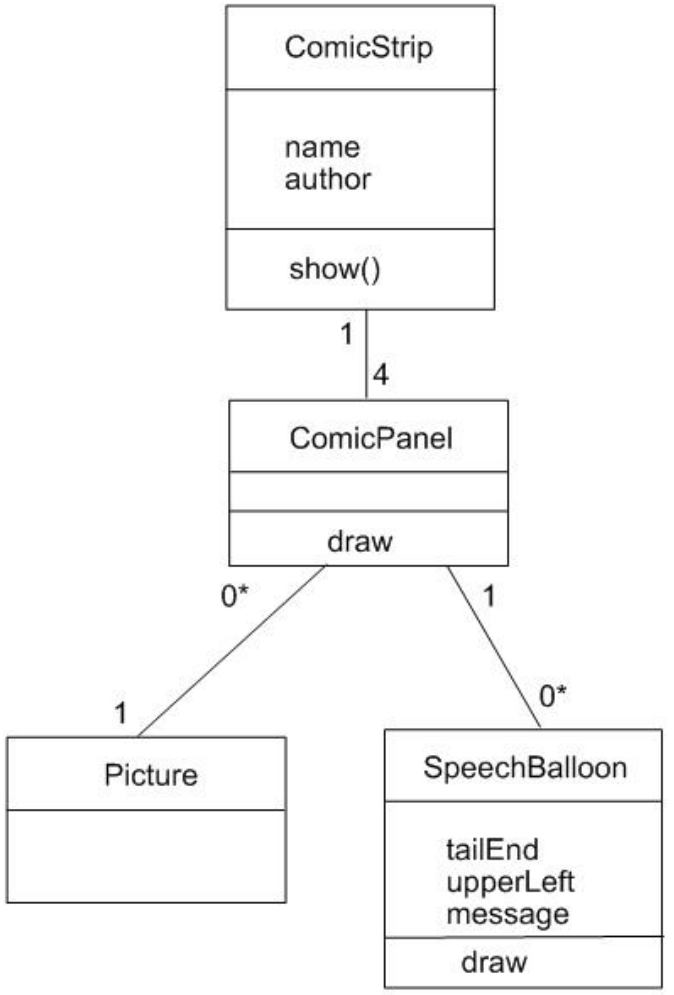
A UML Class Diagram

- Shows the design for an object-oriented solution
  - Shows each class as a rectangular box
    - May include the fields and methods
  - Shows the relationships between objects of the classes
    - Has a (association)
    - Is a (inheritance)



Translating from UML to Java

- Associations may become fields
  - ComicPanel
- The numbers on associations mean the number of objects of one class associated with an object of the other class
  - A ComicStrip has 4 Comic Panels
- The 0* means zero to many
  - A ComicPanel can have 0 to many SpeechBalloons




```
public class ComicStrip extends JPanel
{
 ////////////////// fields ///////////////////////
 private String name;
 private String author;
 private ComicPanel[] panelArray = null;
}
```

Thought Balloons

- Are like speech balloons
  - But they show what a character is thinking, not saying
  - Use circles to connect balloon to thinker

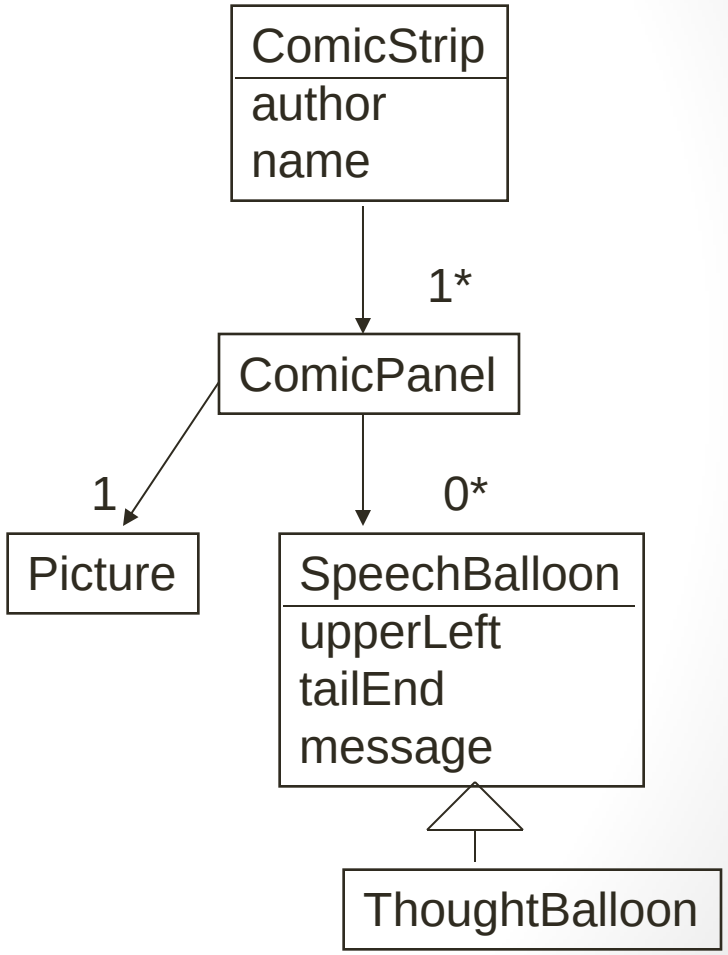
The Comic Strip: Ben

How to Handle Thought Balloons?

- We could copy the class for speech balloon and save it out with a new name
  - And change everywhere that has Speech to Thought
  - But what if we later want to add something to both classes?
    - We would have to add it in both classes
  - Also, how would we handle the ComicPanel?
    - Would it have 0 or more SpeechBalloons **and** 0 or more ThoughtBalloons?

How to Handle Thought Balloons?

- We could make a new class ThoughtBalloon
  - As a subclass of SpeechBalloon
    - Since they are very similar
    - But, is this right?
      - Is a ThoughtBalloon a kind of SpeechBalloon?

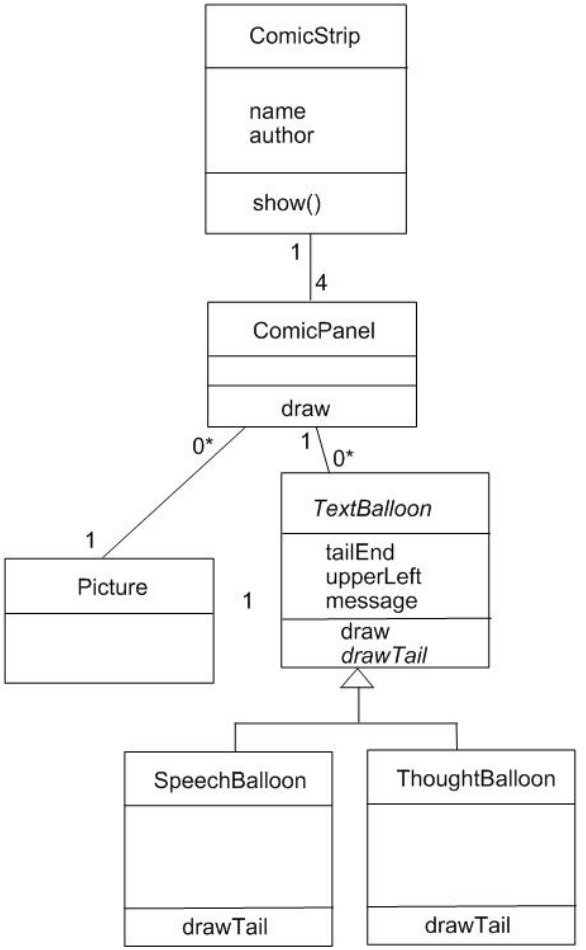


Be Careful with Inheritance!

- Only use it when the child is really of the same type of thing as the parent
  - Not when they just have some similar fields or methods
- Instead pull out a common parent class
  - And have both SpeechBalloon and ThoughtBalloon inherit from it
    - Pull out all common fields and methods
      - And put them in the common parent class
- This is also called generalization

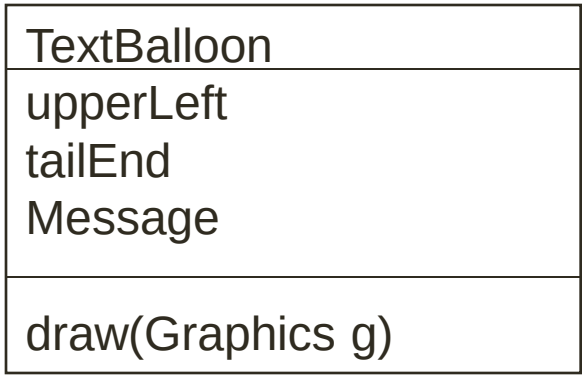
Generalization

- Move common fields and methods into a parent class
  - And let the children classes inherit the fields and methods
  - Easier to maintain than copying fields and methods
  - Better than using inheritance when the child isn't really a kind of parent
  - Makes it easier to add new children classes



TextBalloons

- What should a text balloon know how to do?
  - It should know how to draw itself
    - It will need a graphics context
      - Used to do the drawing
- Add the draw method to the UML class diagram



Abstract Class

- Can we draw just a text balloon?
  - What would we draw?
    - We could draw the balloon and text
    - What kind of tail would it have?
- We need to know what type of text balloon it is in order to draw the tail
  - Which we will only know at run-time
    - Can't predict what will be needed
- So TextBalloon is an abstract class
  - One that you can **not** create an object from
  - You can only inherit from it

public abstract class TextBalloon

Try it

- Try to create a TextBalloon object using the interactions pane in DrJava
  - Use the constructor that takes a upper left point (java.awt.Point), a width, the point of the tail (java.awt.Point) and a string.

```
import java.awt.Point;
TextBalloon balloon = new TextBalloon(new Point (23,33),100,new
    Point(50,40),”Hi”);
```

- What error do you get?

Abstract Methods

- Abstract classes often have one or more abstract methods
  - They don't have to have any
- To make a method abstract
  - Add the abstract keyword after the visibility and before the return type
  - The declaration ends in a semicolon
    - Abstract methods can't have a method body

visibility **abstract** *returnType methodName(parameterList)*;

public abstract void drawTail(paramList);

What visibility should drawTail use?

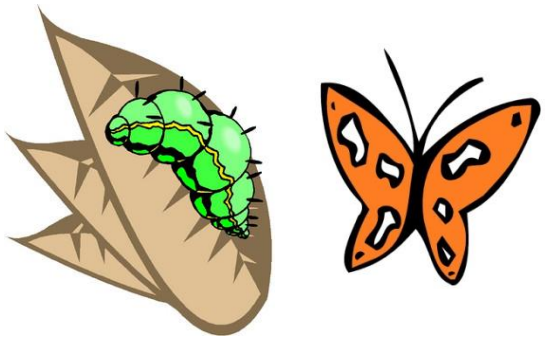
- The abstract method drawTail needs to be overriden by subclasses
  - We can use public visibility
    - But, this means any class has access
  - We can't use private visibility
    - You can't override private methods
  - We could use protected visibility
    - Means the subclasses and classes in the same package have access
      - But, this means you should really put your classes in packages to restrict access

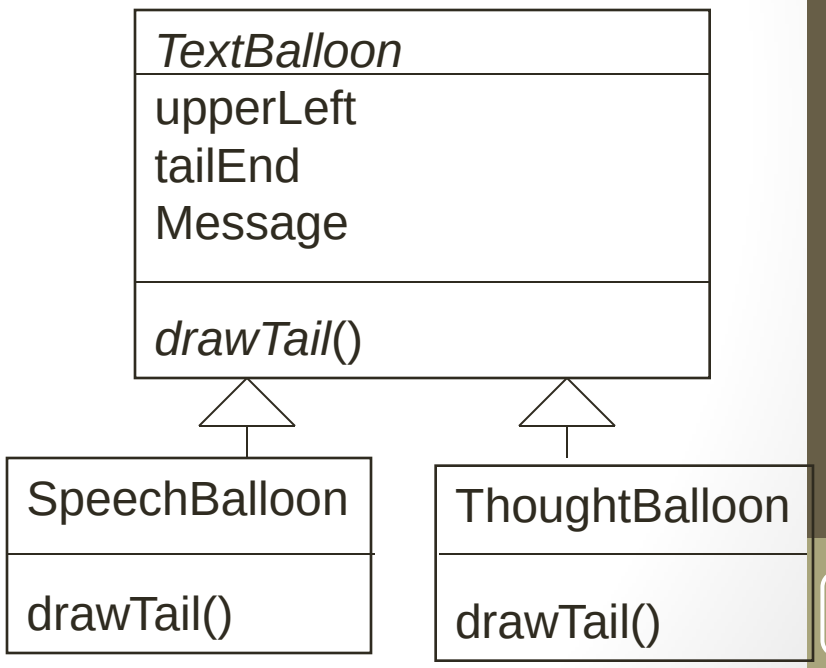
Exercise

- Run the main method in ComicPanel to see an example with both a speech balloon and a thought balloon
- Modify the main method to create your own comic panel with a speech balloon and a thought balloon

Polymorphism

- Means many forms
- Means that the method that is executed depends on the type of the object
  - Often the method is determined by the type of the object at run-time
  - This is called dynamic or run-time binding



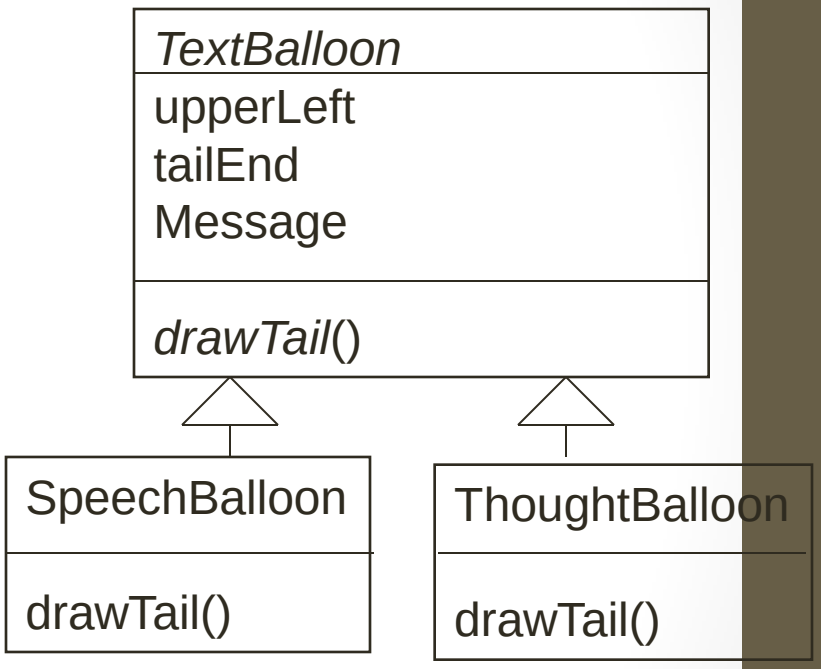


Adding a TextBalloon to a ComicPanel

- Notice that ComicPanel has

  add(TextBalloon textBalloon)

  - Adds it to a list of text balloons
- The method will be called with both SpeechBalloon and ThoughtBalloon
  - They are both types of TextBalloon so there is no problem with this
    - Called Upcasting



Run-time Binding of Methods

- In getFinalPicture()
  - Each element of the textBalloonList is told to draw
    - And in draw each is told to drawTail()
  - What method is called for each of the two types of TextBalloons?
    - The SpeechBalloon
    - The ThoughtBalloon

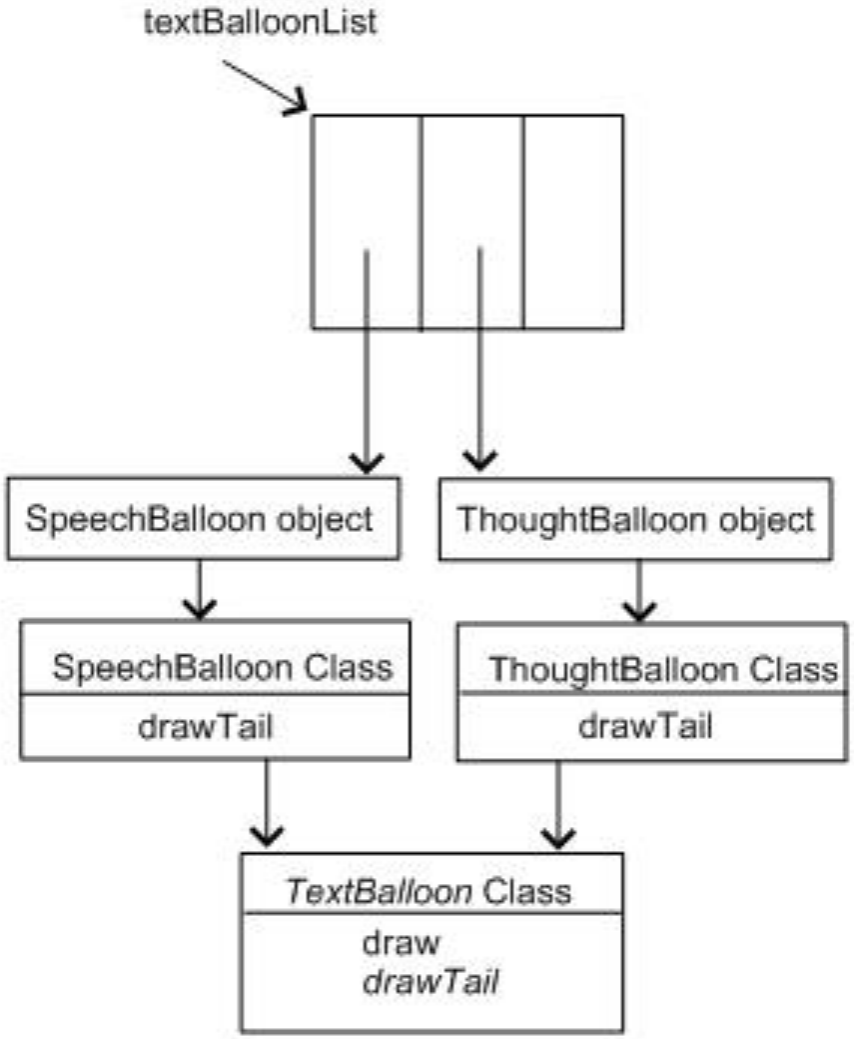
Finding the draw method

- The Java Virtual Machine will start looking for a draw method
  - In the object that defines the class that created the current object
    - Each object keeps a reference to the class that created it
      - Can get this with getClass()
  - If the method isn't found it will try the parent of the class that created the object
  - And so on up the inheritance tree until the method is found
    - It will be found or the code wouldn't have compiled
      - draw will be found in TextBalloon

Finding the drawTail method

- The draw method calls drawTail
  - Again the Java virtual machine will look for the method starting with the class that created the object the method was invoked on
    - So for a SpeechBalloon object it will find it in SpeechBalloon
    - And for a ThoughtBalloon object it will find it in ThoughtBalloon

How it works

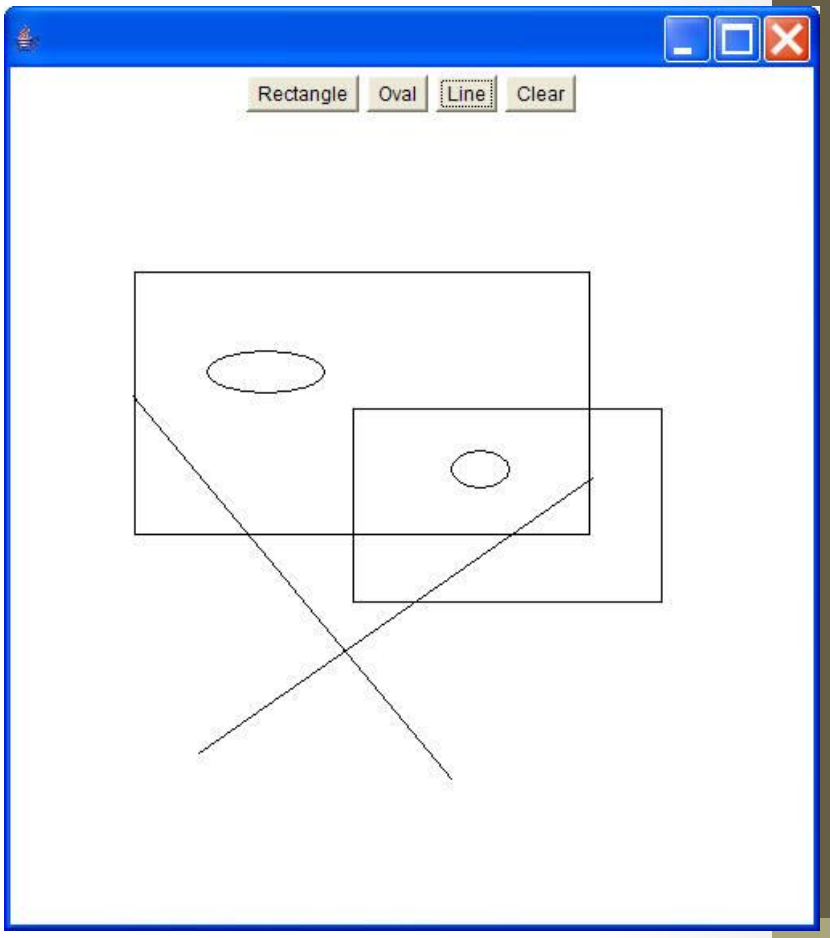

textBalloonList
SpeechBalloon object
ThoughtBalloon object
SpeechBalloon Class
drawTail
ThoughtBalloon Class
drawTail
TextBalloon Class
draw
drawTail

Exercise

- There are other kinds of text balloons
  - Some speech balloons only have a line to the speaker, not a triangle.
  - Some speech balloons don’t draw the outline of the balloon in black.
  - Some thought balloons use a cloud to enclose the text.
  - A scream balloon has bold text and a spiny outline with a flash like tail. It indicates the character is screaming.
  - Some speech balloons have a black background and white text
- Pick at least one of these and implement it.
  - Present what you did and why

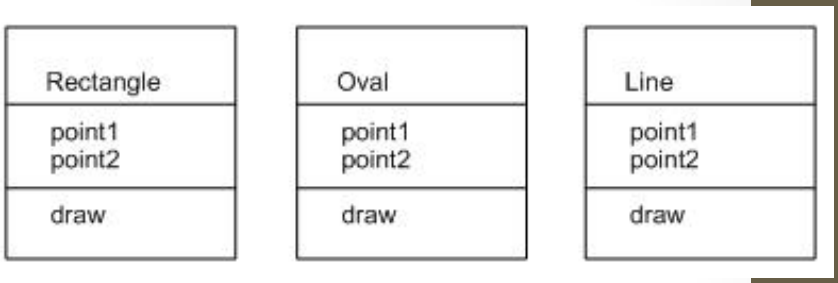
An Application for Drawing Shapes

- What if you want to create an application that allows you to draw shapes
  - That can be defined by two points?
    - Rectangle
    - Oval
    - Line
  - What classes do you need?



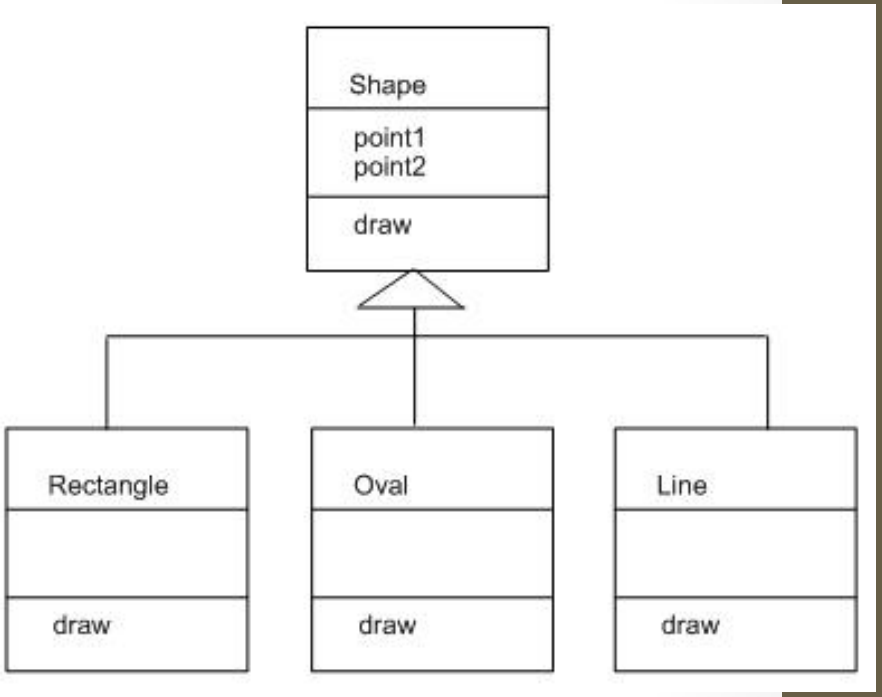
You might start with this

- Classes for Rectangle, Oval, and Line
  - Need 2 points
    - There is a Point class in Java that you can use to represent a 2-dimensional point
  - Need to be able to draw
- Is there a common parent class here?



Rectangle, Oval, and Line are Shapes

- We can pull out a common parent class
  - Shape
- Each shape has
  - Two points that define it
  - A draw method to draw the shape
- But, can we create just a shape object or draw it?
  - No, Shape needs to be an abstract class
    - With an abstract draw method



Code for Shape class

- Abstract class
- Has fields for the two points
- Can have constructors
- Can have other methods
- Has an abstract draw method

```
public abstract class Shape extends Object
{
  protected Color color = Color.black;  // color
  protected Point p1 = new Point(); // first point
  protected Point p2 = new Point(); // second point

  /* constructors */

  /* methods */


  public abstract void draw(Graphics g);
}
```

Code for Rectangle (draw method)

```
public class Rectangle extends Shape
{
  // … constructors

  /** Draw the shape
   * @param g   the graphics context on which to draw
   */
  public void draw(Graphics g)
  {
    // set the color
    g.setColor(color);

    // draw the rectangle given the top left point and width
    // and height
    g.drawRect(getMinX(),getMinY(),getWidth(),getHeight());
  }
}
```

Code for Oval Class (draw method)

```
public class Oval extends Shape
{
  // … constructors

  /** Draw the shape
   * @param g   the graphics context on which to draw
   */
  public void draw(Graphics g)
  {
    // set the color
    g.setColor(color);

    // draw the shape given the top left corner of the enclosing
    // rectangle and the width and height
    g.drawOval(getMinX(),getMinY(),getWidth(),getHeight());
  }
}
```

Why use polymorphism?

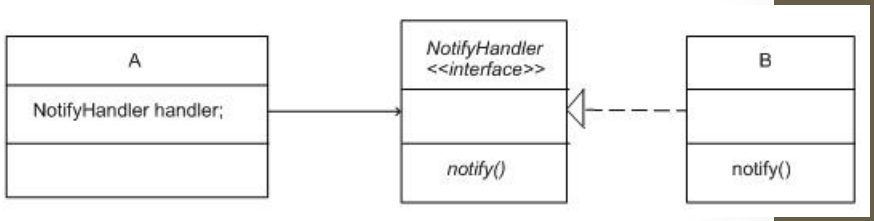
- Allows you to create one parent class
  - And many subclasses that override inherited methods
    - Often abstract methods
  - Easy to add new classes
    - They just need to inherit from the common parent
    - And provide the method or methods
  - You don't need to modify working code
    - Which can result in errors

If you didn't have polymorphism

- You could have one shape class
  - That had a field to say what type the object actually was
    - oval, rectangle, line
  - Then the draw method could use conditional statements to draw the right shape
    - Based on the field
  - Adding new types
    - Means modifying existing code
  - The code becomes harder to read and extend

Interfaces

- If class A has a field of type class B and wants to call the notify method we need to know it is of type B
  - And if we change to class C we need to change class A
- But, if we use an interface we can add new classes that implement the same interface
  - Without changing class A

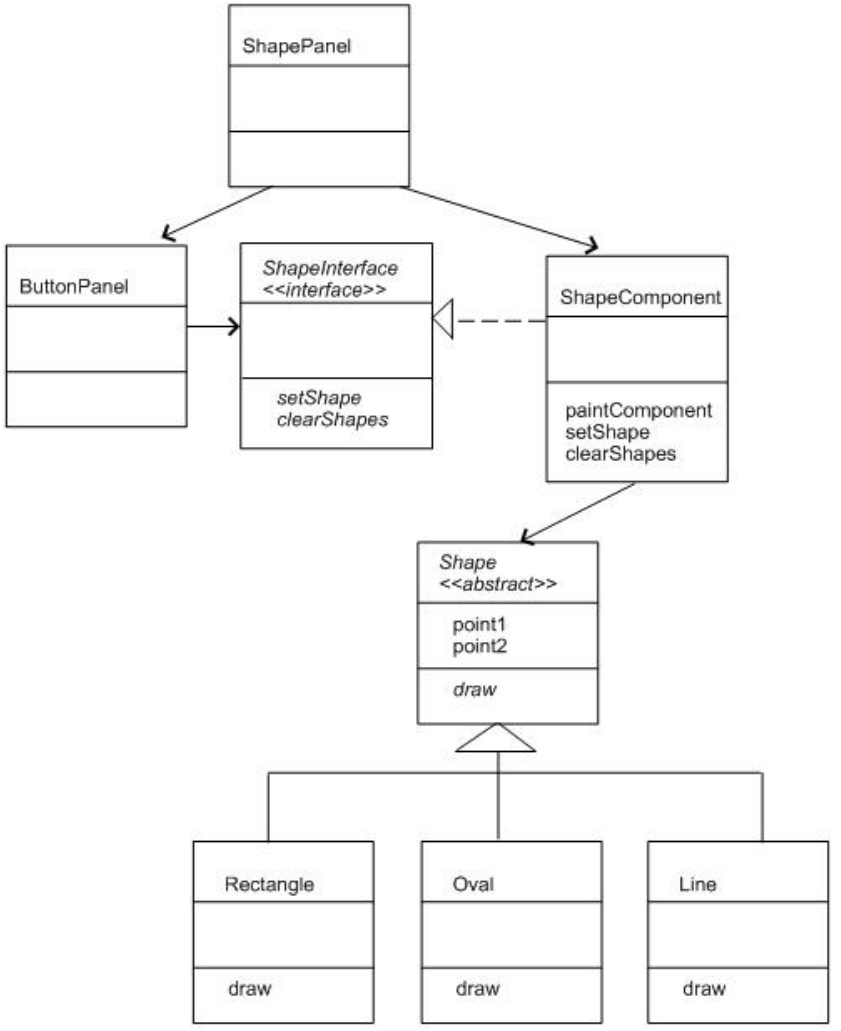


Examples of Interfaces

- Many computers have USB ports
  - You can plug in any device that uses the USB interface
    - Mouse, camera, external disk
- Electrical plugs in the US have one type of interface
  - You can plug in any device

Interface use in ShapeExample

- ShapeInterface is an Interface
- ButtonPanel has an object of type ShapeInterface
  - Calls setShape and clearShapes on it
- ShapeComponent implements the interface



Comparable Interface

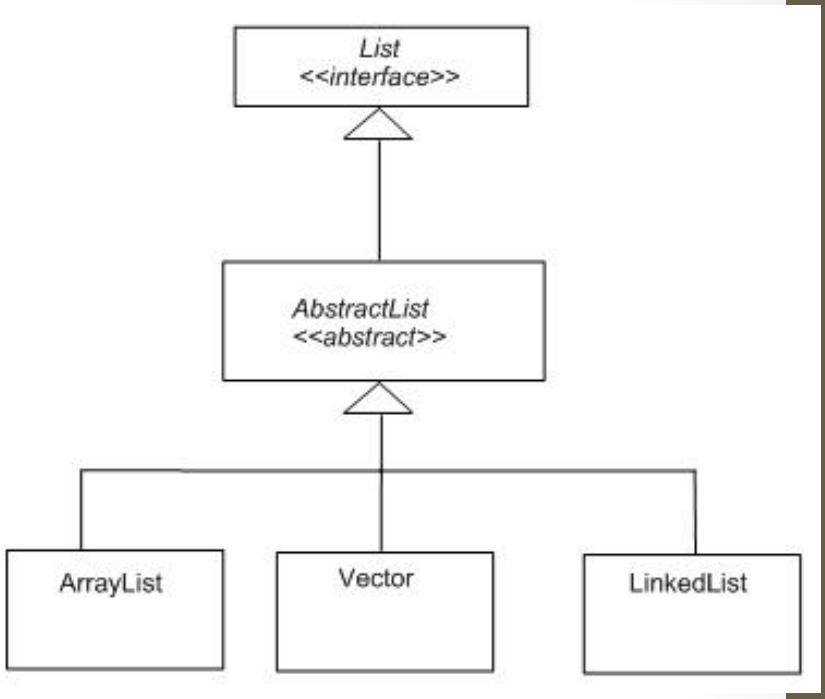
- The Comparable interface allows you to write search methods
  - that don't depend on the class of the objects
  - just on the Comparable interface
- The Comparable interface
  - Defines the compareTo method

List Interface

- What can you do with a list?
  - Add objects to it
  - Remove objects from it
  - Check to see if an object is in the list
  - Get an item at a specific position in the list
- Java defines a List interface
  - And several classes implement that interface
    - ArrayList, Vector, LinkedList

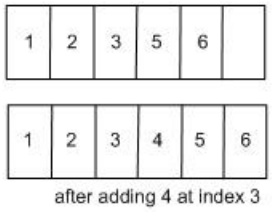
List Interfaces and Classes

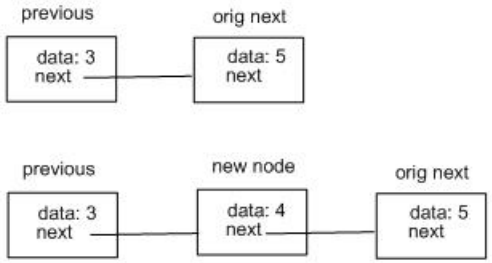
- The List interface defines the basic methods for all lists
- The AbstractList class implements the List interface and provides some methods
- The ArrayList, Vector, and LinkedList classes inherit from AbstractList



List Classes

- ArrayList
  - Uses an array to implement a list
    - Grows and shrinks as needed to hold the data
  - Isn't thread safe
- Vector
  - Uses an array to implement a list
  - Is thread safe
- LinkedList
  - A list of nodes where each node has a link to the next one





Thread safe

- Means that the data is protected so that two processes can't modify the data at the same time
- Computation can be broken into multiple processes
  - Like having more than one person help with cooking a meal
  - You need to be careful to not do the same step
    - Like adding the salt

Summary

- In analysis you try to understand the problem you are trying to solve
  - What are the objects that are involved? What data and behavior do the objects need?
- In design you describe classes and their relationships
  - In a UML class diagram
- Inheritance is used to
  - Pull out common things into a parent class - Generalization
  - Allow a child to differ from a parent - Specialization
- Abstract classes are used to allow for easy addition of new subclasses
  - With some abstract method overriden by the child class
- Polymorphism allows for the right method to be executed based on the run-time type of the object