Programming in Java: lecture 7

- Inheritance
- Polymorphism
- Abstract Classes
- this and super
- Interfaces
- Nested Classes and other details
- Example

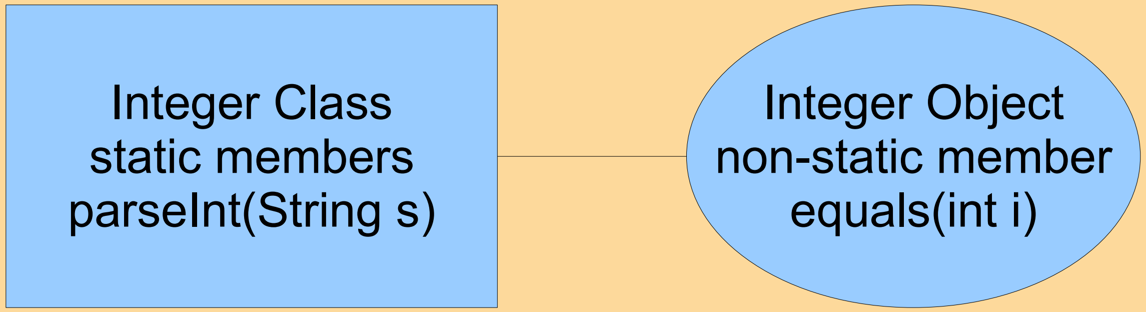
Slides made for use with "Introduction to Programming Using Java, Version 5.0" by David J. Eck
Some figures are taken from "Introduction to Programming Using Java, Version 5.0" by David J. Eck
Lecture 7 covers Section 5.5 to 5.7

Last time

- Objects, Classes and Instances
- Getters and setters
- Constructors and object initialization
- Wrapper Classes and Autoboxing
- Garbage collection and the heap

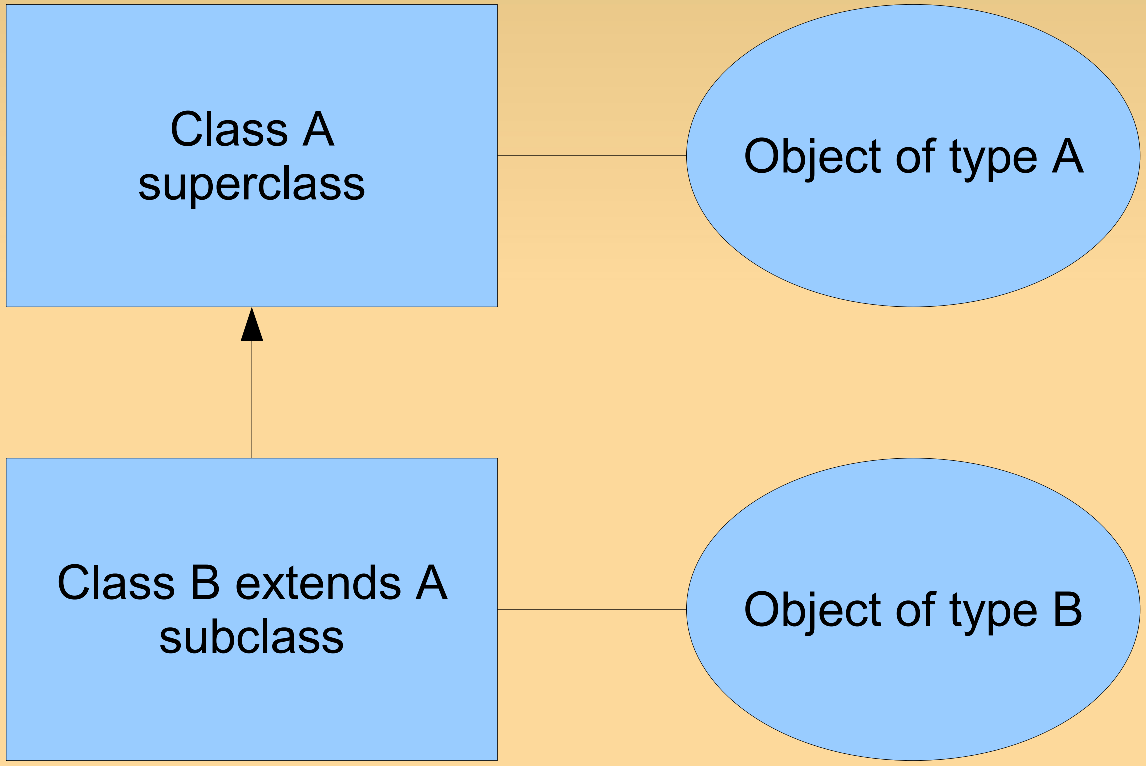
Classes and Objects

- A Class is a template
- Objects are objects
- Objects are instances of a given class

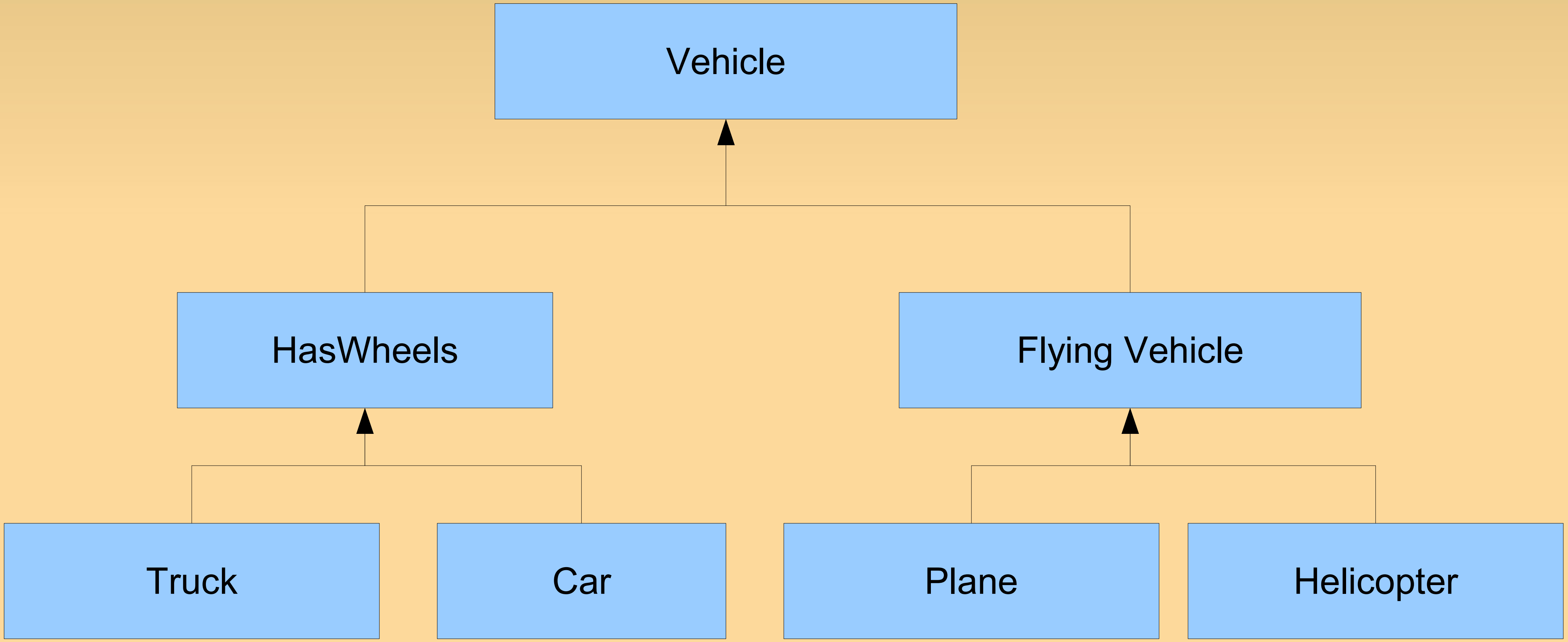


Inheritance

- Objects are instances of a given class

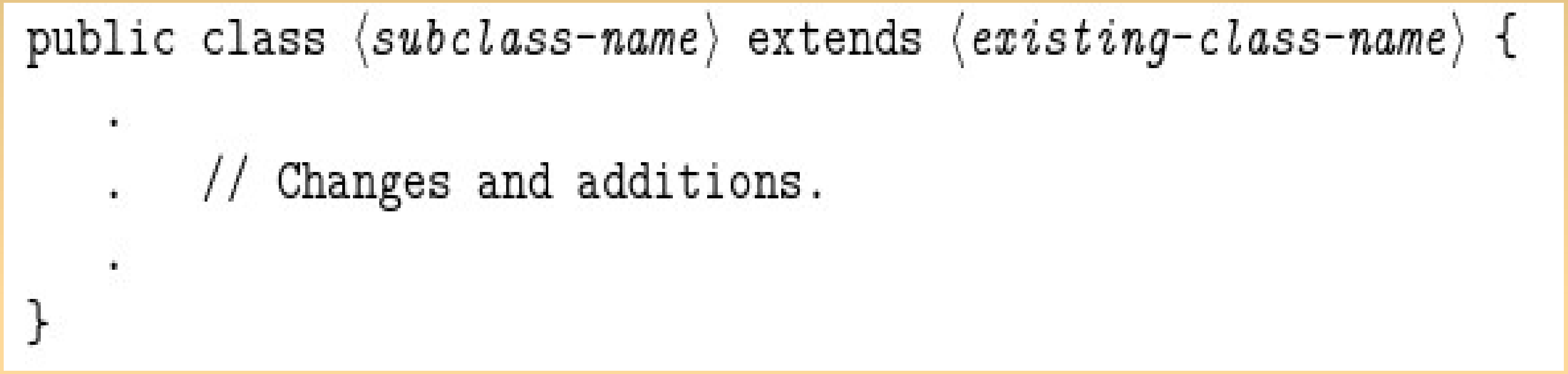


Inheritance



Inheritance

- Syntax

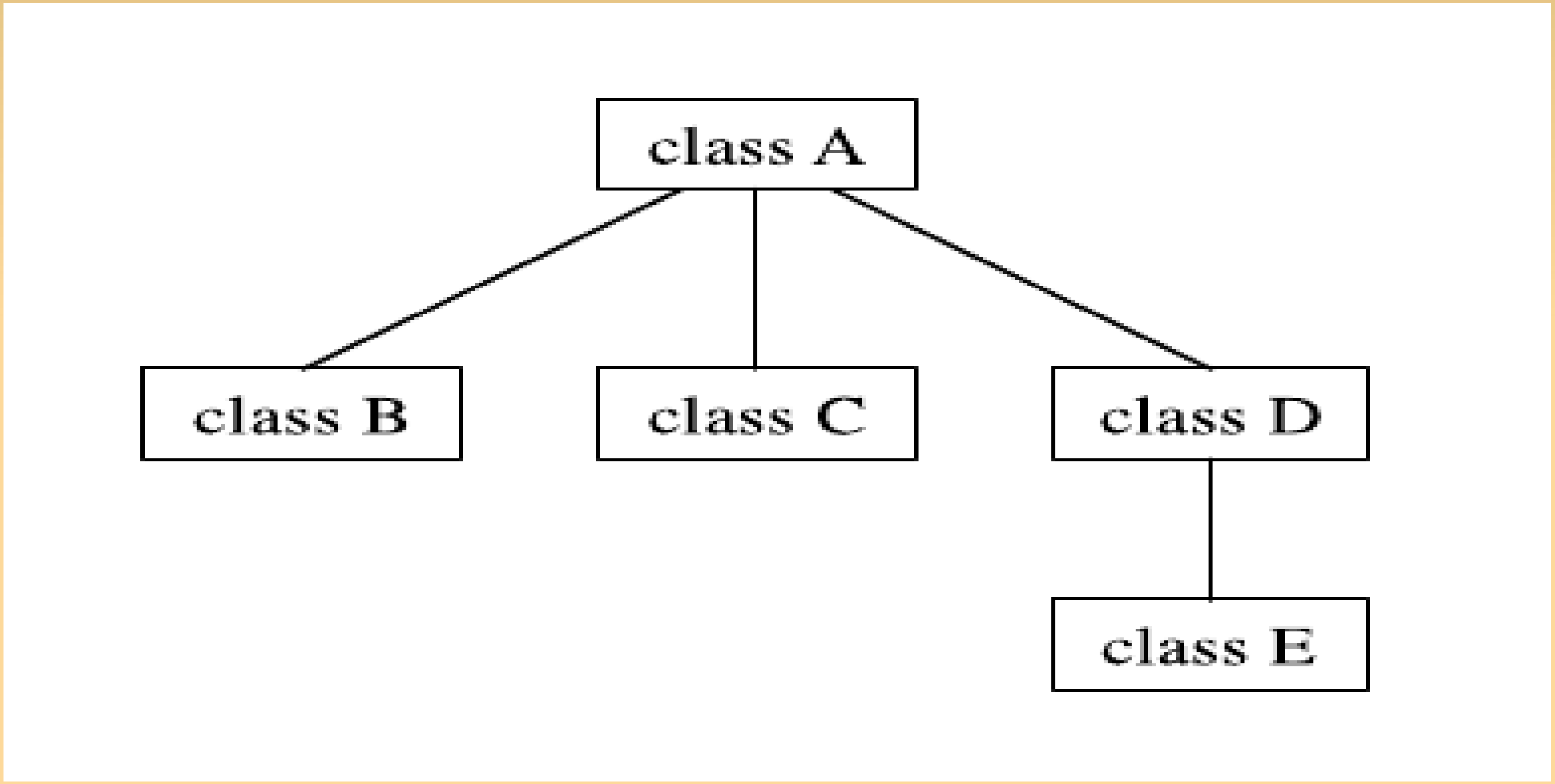


```
public class ⟨subclass-name⟩ extends ⟨existing-class-name⟩ {
    .
    .   // Changes and additions.
    .
}
```

- Extending existing classes
  - new methods
  - override methods
  - new instance variables

Class hierarchy

- Everything extends Object

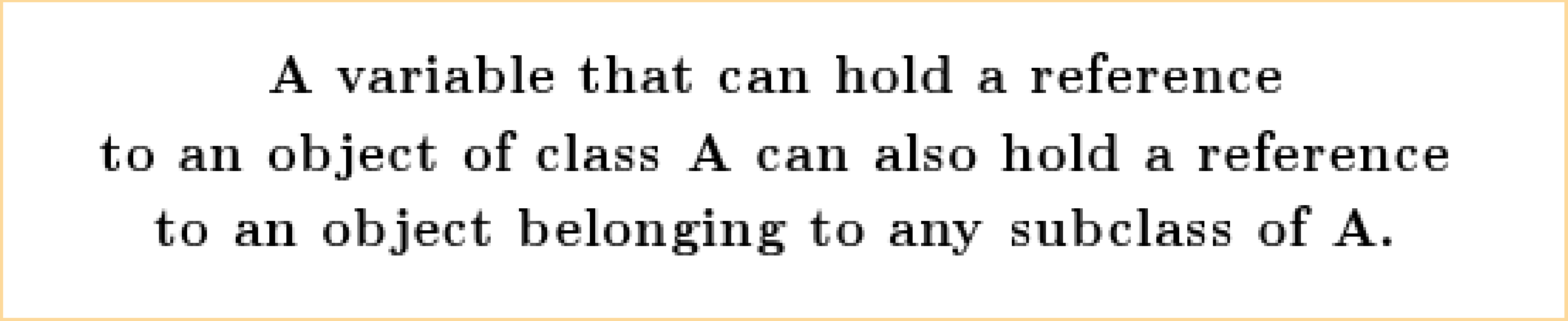


Access modifiers

- Private
  - Only in the class itself
- Protected
  - Same package and subclasses in other packages
- Default
  - Same package
- Public
  - Everybody

Polymorphism

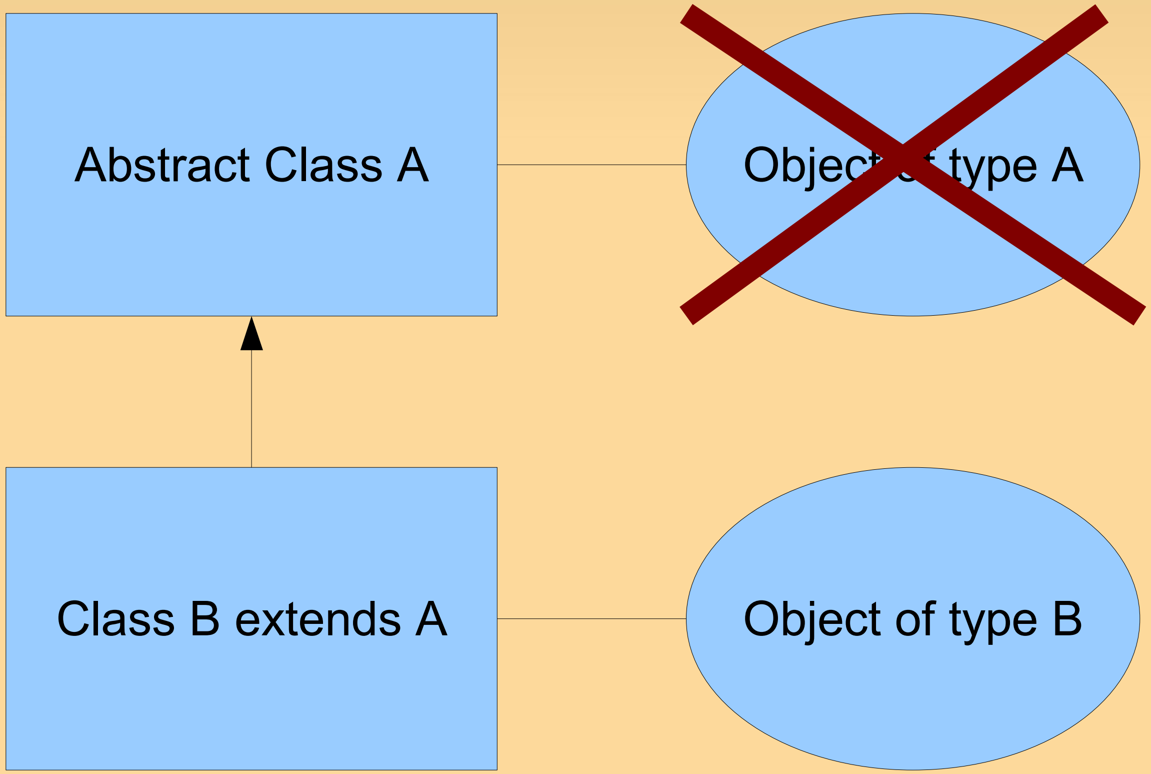
- Two concepts
- We can write code that can handle all future subclasses
- We can have variables without knowing the exact type of the object that it refers to



A variable that can hold a reference
to an object of class A can also hold a reference
to an object belonging to any subclass of A.

Abstract class

- Cannot make objects from abstract classes
- Can make variables from abstract classes



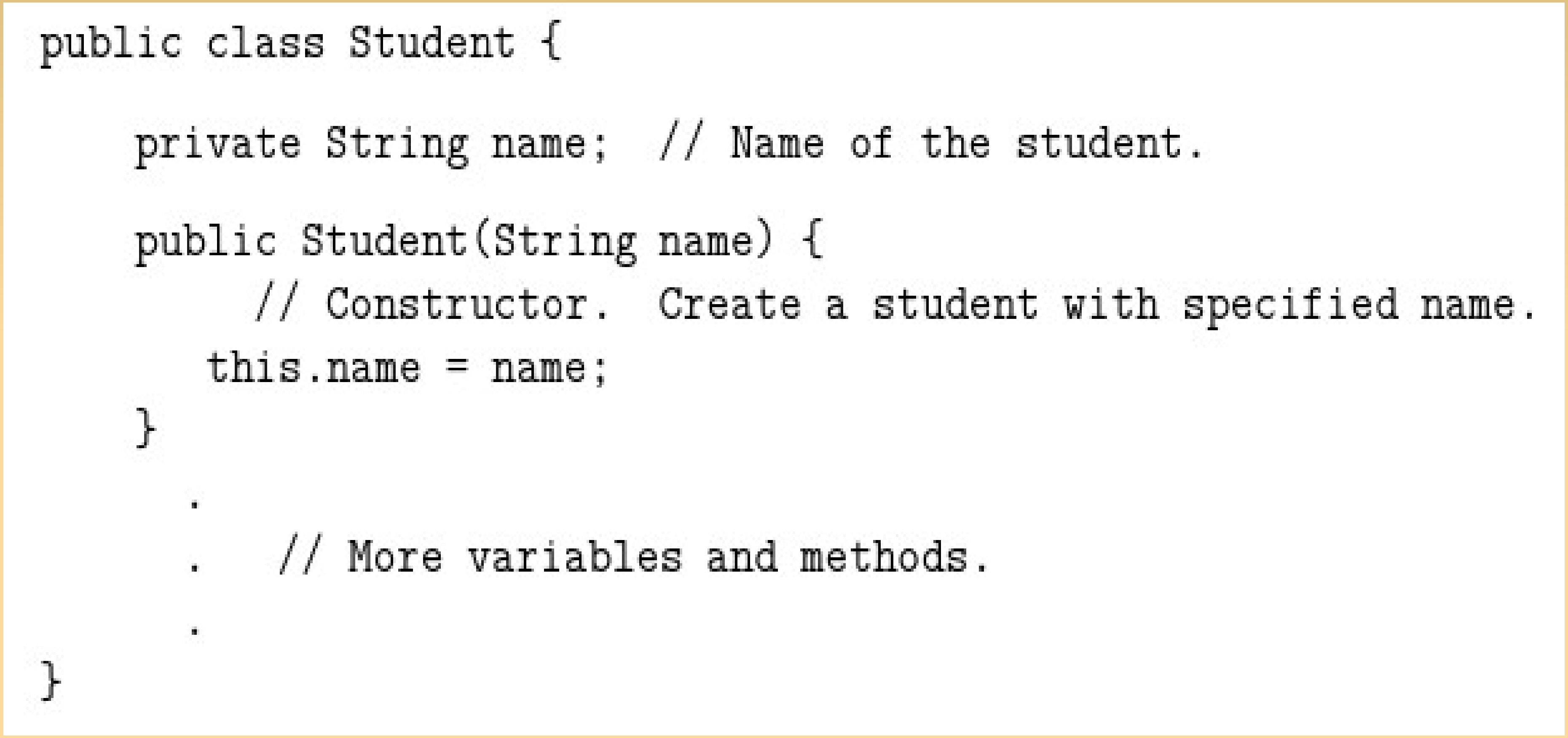
Abstract example

```
public abstract class Shape {

    Color color;   // color of shape.

    void setColor(Color newColor) {
          // method to change the color of the shape
       color = newColor; // change value of instance variable
       redraw(); // redraw shape, which will appear in new color
    }

    abstract void redraw();
          // abstract method---must be defined in
          // concrete subclasses

    . . .           // more instance variables and methods

} // end of class Shape
```

this and super

- special variables
- cannot be assigned to
- this – the object we are currently in
- super – used to call methods of the super class
  - forgets the exact type of the object
- special use in constructors
  - Used as a method name
  - Calls other constructors

this – example



```
public class Student {

    private String name;  // Name of the student.

    public Student(String name) {
         // Constructor.  Create a student with specified name.
       this.name = name;
    }
      .
      .    // More variables and methods.
      .
}
```

super – example

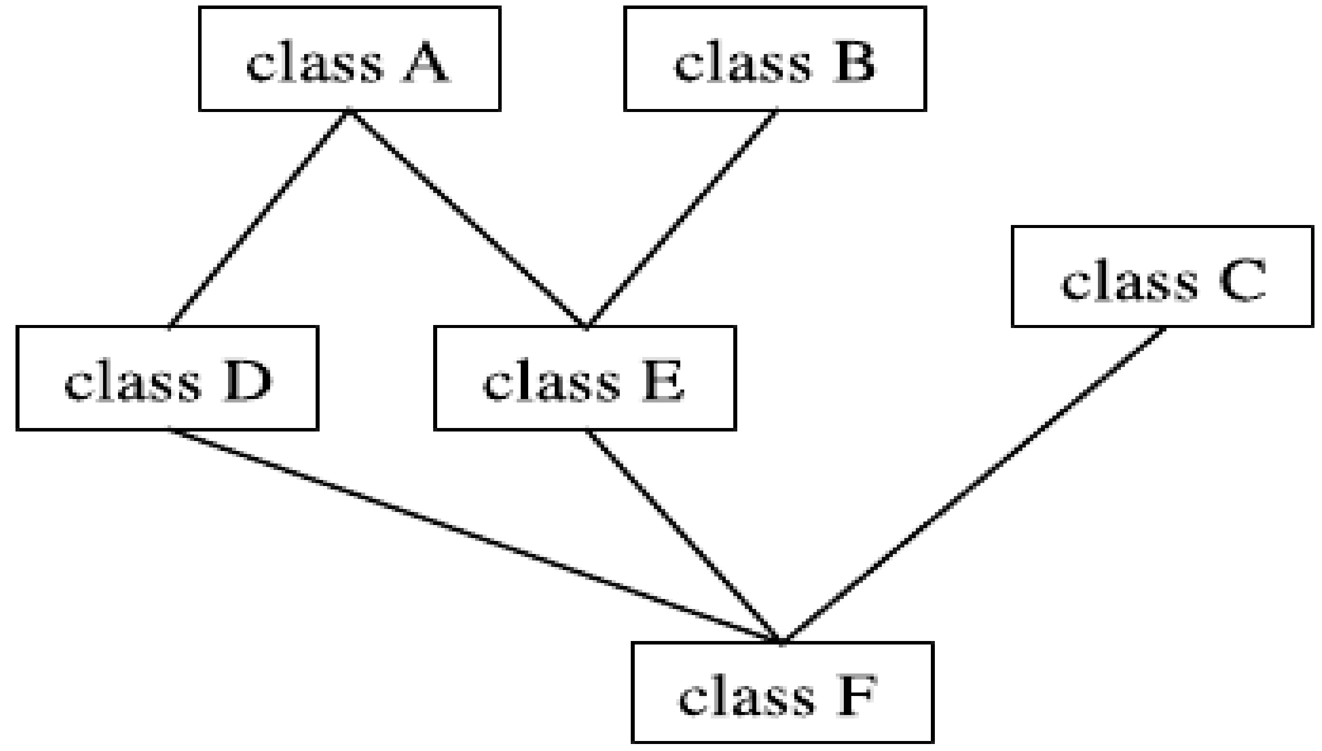
```
public class SymmetricBrighten extends RandomBrighten {

   void brighten(int row, int col) {
         // Brighten the specified square and its horizontal
         // and vertical reflections.  This overrides the brighten
         // method from the RandomBrighten class, which just
         // brightens one square.
      super.brighten(row, col);
      super.brighten(ROWS - 1 - row, col);
      super.brighten(row, COLUMNS - 1 - col);
      super.brighten(ROWS - 1 - row, COLUMNS - 1 - col);
   }

} // end class SymmetricBrighten
```

Constructor example

```
public class GraphicalDice extends PairOfDice {

    public GraphicalDice() {  // Constructor for this class.

        super(3,4);  // Call the constructor from the
                     //   PairOfDice class, with parameters 3, 4.

        initializeGraphics();  // Do some initialization specific
                               //   to the GraphicalDice class.
    }

       .
       .  // More constructors, methods, variables...
       .
}
```

Multiple inheritance

- Not allowed in Java



Multiple inheritance (NOT allowed in Java)

Interfaces

- Describes an aspect
- Completely abstract class
  - nothing can be implemented

```
public interface Drawable {
   public void draw(Graphics g);
}
```

```
public class Line implements Drawable {
    public void draw(Graphics g) {
        . . . // do something---presumably, draw a line
    }
    . . . // other methods and variables
}
```

Interfaces

- Implementing multiple interfaces (serializable)

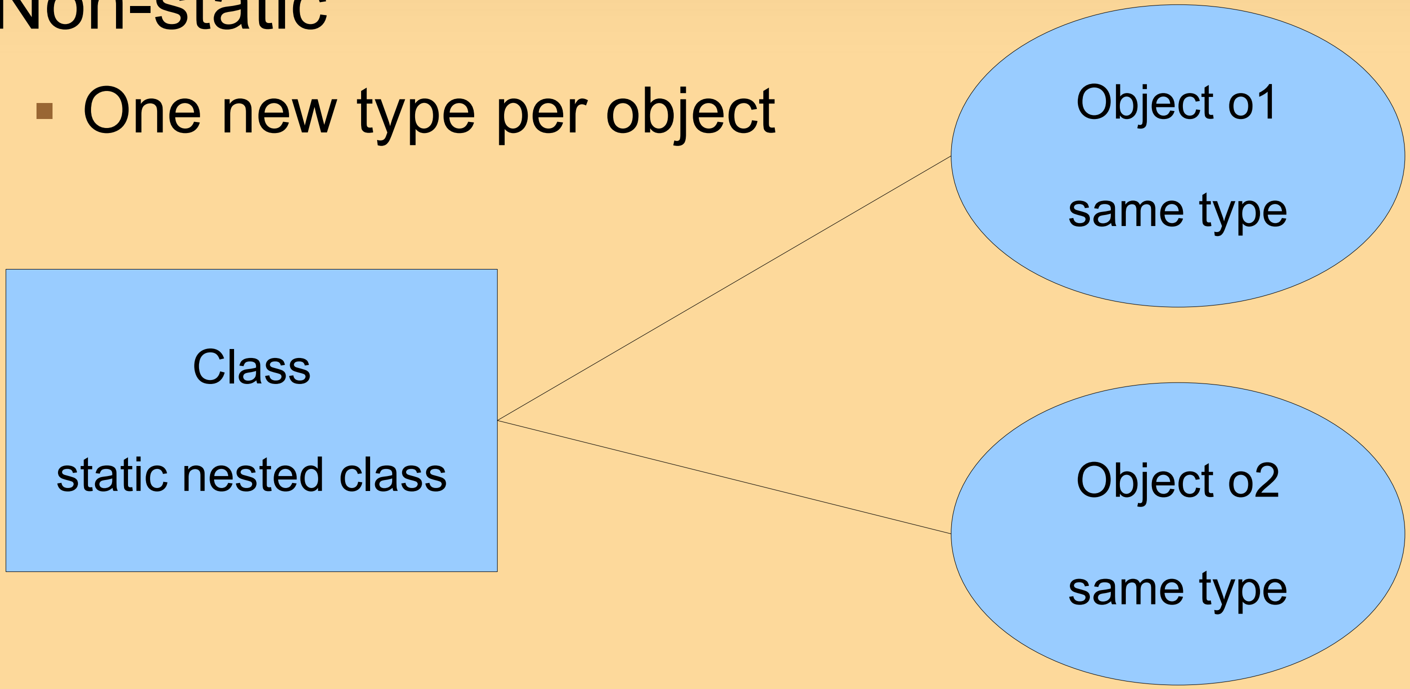
```
class FilledCircle extends Circle
                        implements Drawable, Fillable {
   . . .
}
```

- Use of objects

```
Drawable figure;  // Declare a variable of type Drawable.  It can
                  //    refer to any object that implements the
                  //    Drawable interface.

figure = new Line();  // figure now refers to an object of class Line
figure.draw(g);   // calls draw() method from class Line

figure = new FilledCircle();   // Now, figure refers to an object
                               //    of class FilledCircle.
figure.draw(g);   // calls draw() method from class FilledCircle
```

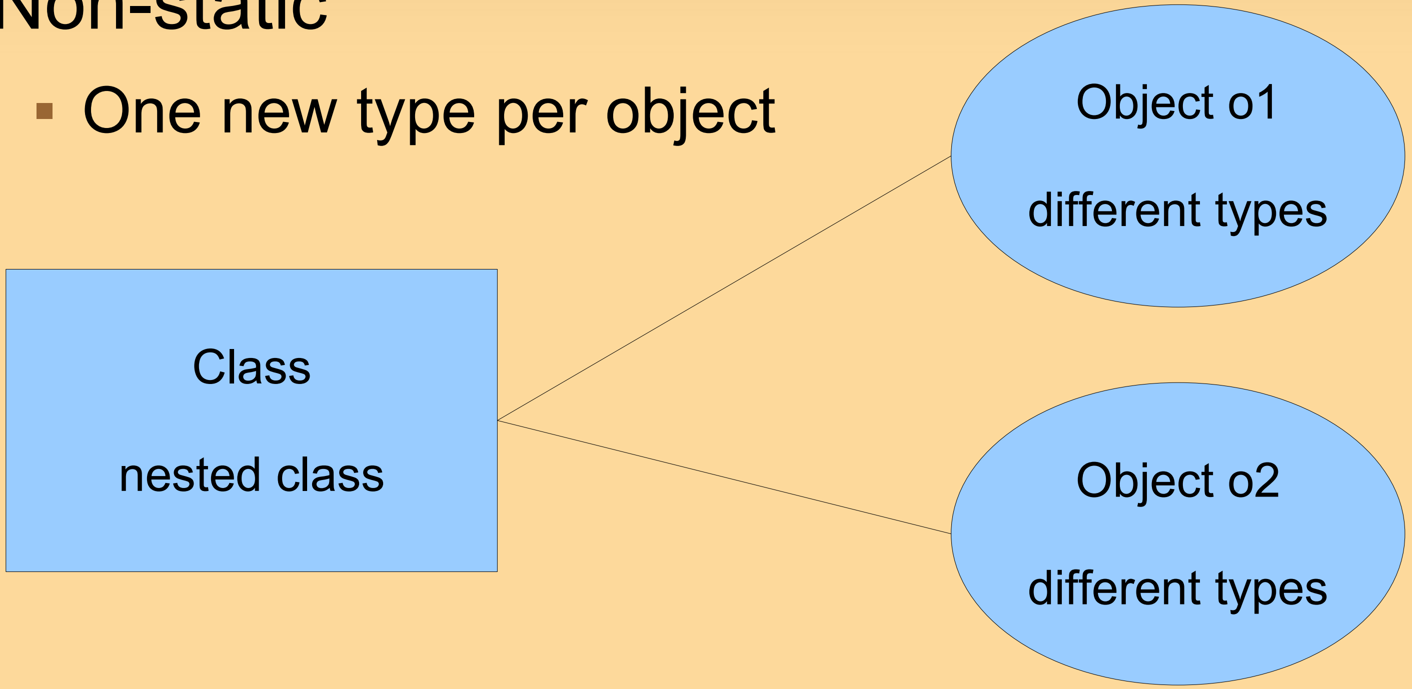
Nested classes

- Classes inside classes
  - Static
    - Only one new type
  - Non-static
    - One new type per object



Nested classes

- Classes inside classes
  - Static
    - Only one new type
  - Non-static
    - One new type per object



Example – static

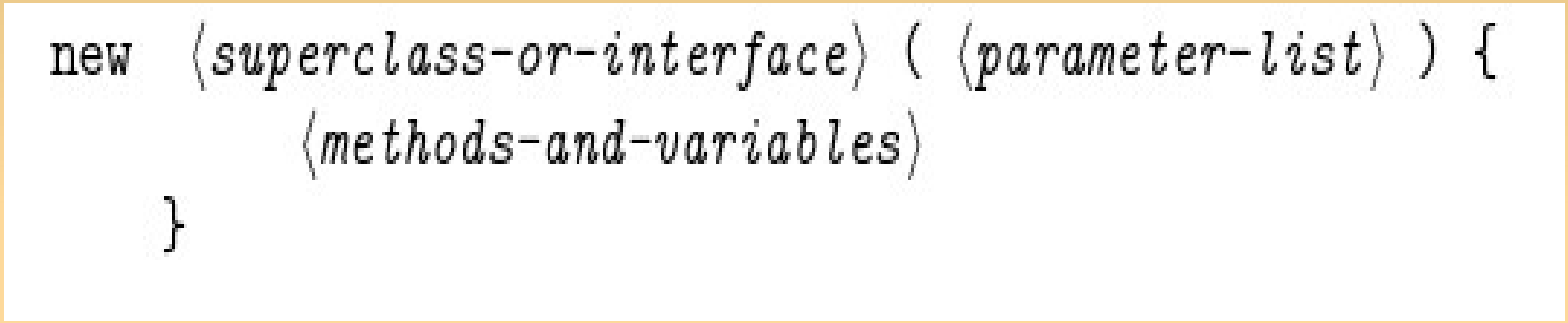
```
public class WireFrameModel {

   . . . // other members of the WireFrameModel class

   static public class Line {
         // Represents a line from the point (x1,y1,z1)
         // to the point (x2,y2,z2) in 3-dimensional space.
      double x1, y1, z1;
      double x2, y2, z2;
   } // end class Line

   . . . // other members of the WireFrameModel class

} // end WireFrameModel
```

Example – non static

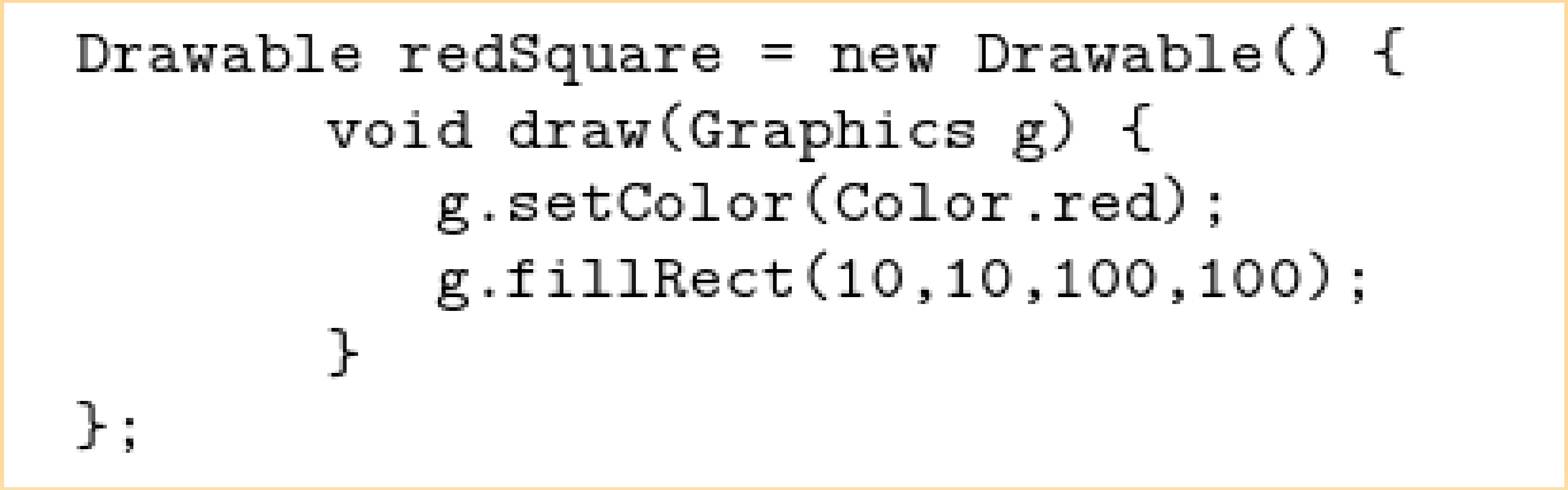
```
public class PokerGame {  // Represents a game of poker.

    private class Player {  // Represents one of the players in this game.
        .
        .
        .
    } // end class Player

    private Deck deck;       // A deck of cards for playing the game.
    private int pot;         // The amount of money that has been bet.

    .
    .
    .

} // end class PokerGame
```

Anonymous Inner Classes

- If you only need it in one place



```
new ⟨superclass-or-interface⟩ ( ⟨parameter-list⟩ ) {
        ⟨methods-and-variables⟩
    }
```



```
Drawable redSquare = new Drawable() {
        void draw(Graphics g) {
            g.setColor(Color.red);
            g.fillRect(10,10,100,100);
        }
};
```

Static import

```
import static ⟨package-name⟩.⟨class-name⟩.⟨static-member-name⟩;
```

```
import static ⟨package-name⟩.⟨class-name⟩.*;
```

```
import static java.lang.System.out;
```

```
import static java.lang.Math.*;
```

Enums

- Enums are classes
- each enumerated type is a public static final member

Example

- Team programming