6.092 Introduction to Software Engineering in Java
January (IAP) 2009

6.092

Lecture 6

Object-Oriented Programming

Inheritance

Abstract Classes and Interfaces

Assignment 5 Review - Graphics!

- Given
  - SimpleDraw.java
  - BouncingBox.java
  - DrawGraphics.java
- Part I: add three shapes to the window
- Part II: animate the box using moveInDirection(); add three boxes using an ArrayList

Assignment 5 Review - Graphics!

- **Look at the comments we write for you on Stellar!**
- **Look at the solutions we post on Stellar.**

Finding your way in the Java forest

- In case of multiple java files, who has the main() method?

```
public class SimpleDraw extends JPanel implements Runnable {
    private DrawGraphics draw;
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        draw.draw(g2);
    }
    public static void main(String args[]) {
        // start a drawing thread
    }
}
```

The DrawGraphics Class

```
public class DrawGraphics {
    BouncingBox box; // box is called a field

    public DrawGraphics() { // constructor
        box = new BouncingBox(200, 50, Color.RED);
    }

    public void draw(Graphics2D surface) {
        surface.drawLine(50, 50, 250, 250);
        box.draw(surface);
    }
}
```

The BouncingBox Class

```
public class BouncingBox {
        public BouncingBox(int startX, int startY, Color startColor) {
          // constructor
         x = startX;
         y = startY;
         color = startColor;
    }
```

The BouncingBox Class

```
public class BouncingBox {
public void draw(Graphics2D surface) {
      // draw the box
      }
public void moveInDirection(int xIncrement, int yIncrement) {
        xDirection = xIncrement;
        yDirection = yIncrement;
    }
}
```

moveInDirection() does not move the box. It specifies in which direction the box should be moving. Default value is (0,0) (i.e. not move).

Part I: add three shapes

```
public class DrawGraphics {


    public void draw(Graphics2D surface) {
        surface.drawLine(50, 50, 250, 250);
        box.draw(surface);
      surface.fillRect (150, 100, 25,  40);
      surface.fillOval (40, 40, 25, 10);
      surface,setColor (Color.YELLOW);
      surface.drawString ("Mr. And Mrs. Smith", 200, 10);
    }
}
```

Part II: animate the box

- “To get the box to move, call the moveInDirection method in the DrawGraphics constructor, with an x and y offset.”

Part II: animate the box

- "To get the box to move, call the moveInDirection method in the DrawGraphics constructor, with an x and y offset."

```
public class DrawGraphics {

    BouncingBox box;


    public DrawGraphics() { // constructor
        box = new BouncingBox(200, 50, Color.RED);
        box.moveInDirection (10,5);
    }
}
```

Reminder on constructors

- Constructors are for initialization.
- They are called **once and only once** every time an object of the class is created with **new**.
- They **must** have the same name as the class.

Part II: use an ArrayList

```
ArrayList<String> strings = new ArrayList<String>();
String c = “Phil”;
strings.add(c);
String d = strings.get(0);
```

```
ArrayList<BouncingBox> boxes = new ArrayList<BouncingBox>();
BouncingBox b = new BouncingBox (200, 50, Color.RED);
boxes.add(b);
BouncingBox d = boxes.get(0);
```

Part II: use an ArrayList

```
public class DrawGraphics {
    BouncingBox box;
    public DrawGraphics() { // constructor
      box = new BouncingBox(200, 50, Color.RED);
    }
  }
```

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;
    public DrawGraphics() { // constructor
        boxes = new ArrayList<BouncingBox>();
    }
}
```

What's wrong with this?

```
public class DrawGraphics {
    BouncingBox box;

    public DrawGraphics() { // constructor
        BouncingBox box;
        box = new BouncingBox(200, 50, Color.RED);
    }

    public draw (Graphics2D surface) {
        box.draw (surface);
    }
}
```

What's wrong with this?

```
public class DrawGraphics {
    BouncingBox box;

    public DrawGraphics() { // constructor
        BouncingBox box;
        box = new BouncingBox(200, 50, Color.RED);
    }

    public draw (Graphics2D surface) {
        box.draw (surface); // box does not exist here!
    }
}
```

Part II: use an ArrayList

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;

    public DrawGraphics() { // constructor
        boxes = new ArrayList<BouncingBox>();
    }
}
```

Part II: use an ArrayList

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;

    public DrawGraphics() { // constructor
        boxes = new ArrayList<BouncingBox>();
        BouncingBox b = new BouncingBox (100, 50, Color.RED);
        boxes.add (b);
        boxes.add (new BouncingBox (10, 50, Color.RED));
        boxes.add (new BouncingBox (50, 80, Color.YELLOW));
    }
}
```

Part II: use an ArrayList

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;
    public DrawGraphics() { // constructor
        boxes = new ArrayList<BouncingBox>();
        // fill in boxes here
        BouncingBox c = boxes.get(0);
        c.moveInDirection (10, 5);
        boxes.get(1).moveInDirection (20,-5);
        boxes.get(2).moveInDirection (-3, 18);
    }
}
```

Even better...

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;
    public DrawGraphics() { // constructor
        boxes = new ArrayList<BouncingBox>();
        // fill in boxes here
        for (BouncingBox b : boxes) {
            b.moveInDirection (10, 15);
        }
    }
}
```

Part II: use an ArrayList

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;

    public void draw(Graphics2D surface) {
        surface.drawLine(50, 50, 250, 250);
        boxes..get(0).draw(surface);
        boxes..get(1).draw(surface);
        boxes..get(2).draw(surface);
    }
}
```

Even better...

```
public class DrawGraphics {
    ArrayList<BouncingBox> boxes;

    public void draw(Graphics2D surface) {
        surface.drawLine(50, 50, 250, 250);
        for (BouncingBox b : boxes) {
            b.draw (surface);
        }
    }
}
```

What you learned in Assignment 5

- Found your way in multiple Java files
- Used Graphics2D to draw shapes
- Animated a shape using moveInDirection()
- Used ArrayList to animate several boxes
-
- **Write clean and elegant code!!**

Menu du jour

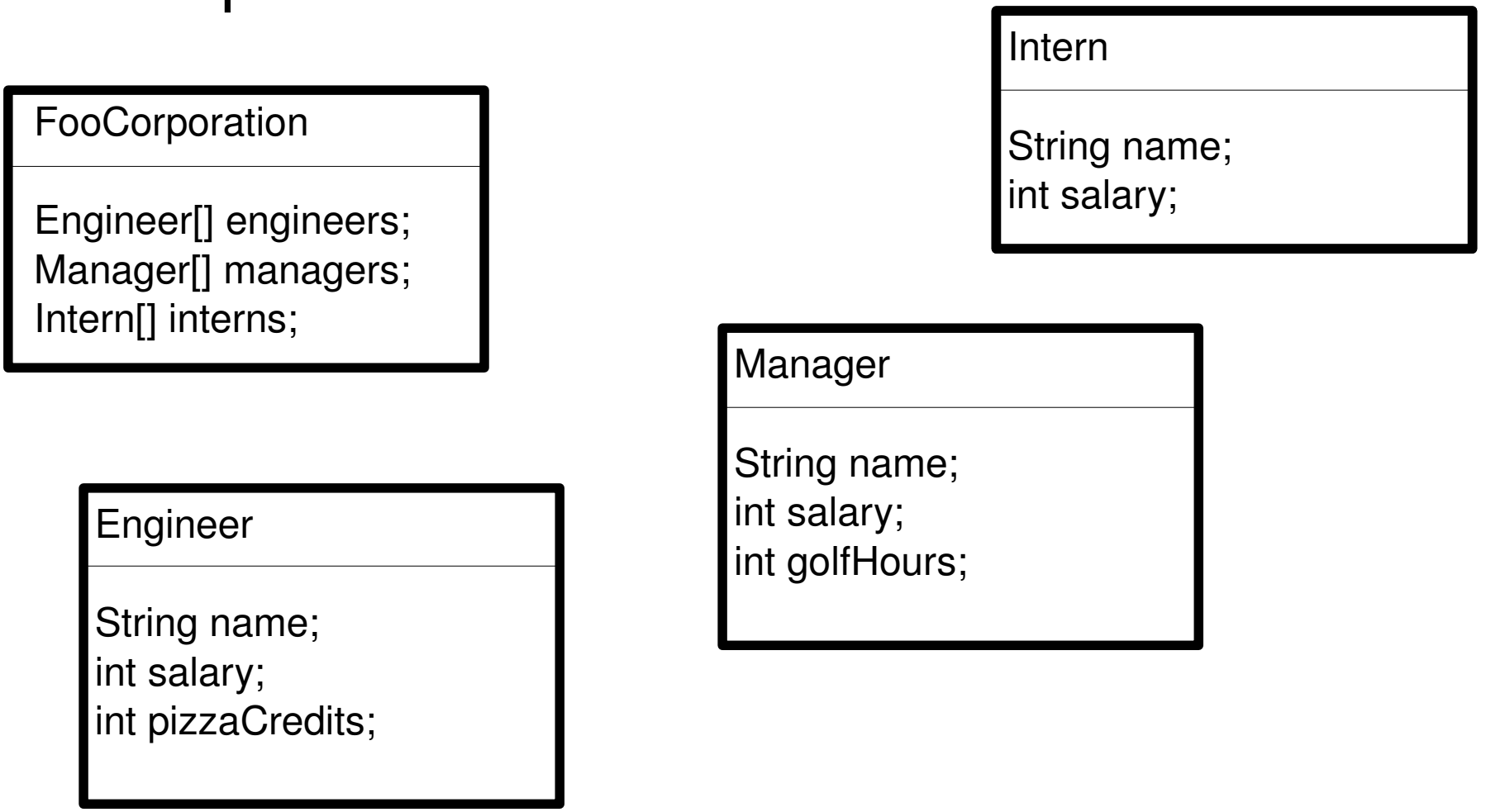
- Object-oriented programming (OOP)
- Inheritance
- Abstract classes and interfaces

Object-oriented programming

- The world is more than a pile of int, double and arrays...
- Classes model the real world
  - e.g. Bicycle, FooCorporation, etc.
- An object is an instance of a class
  - Person me = new Person (“Joshua”);
  - The ***object*** referred to by the ***variable*** with the name “me” is an ***instance*** of the Person ***class***

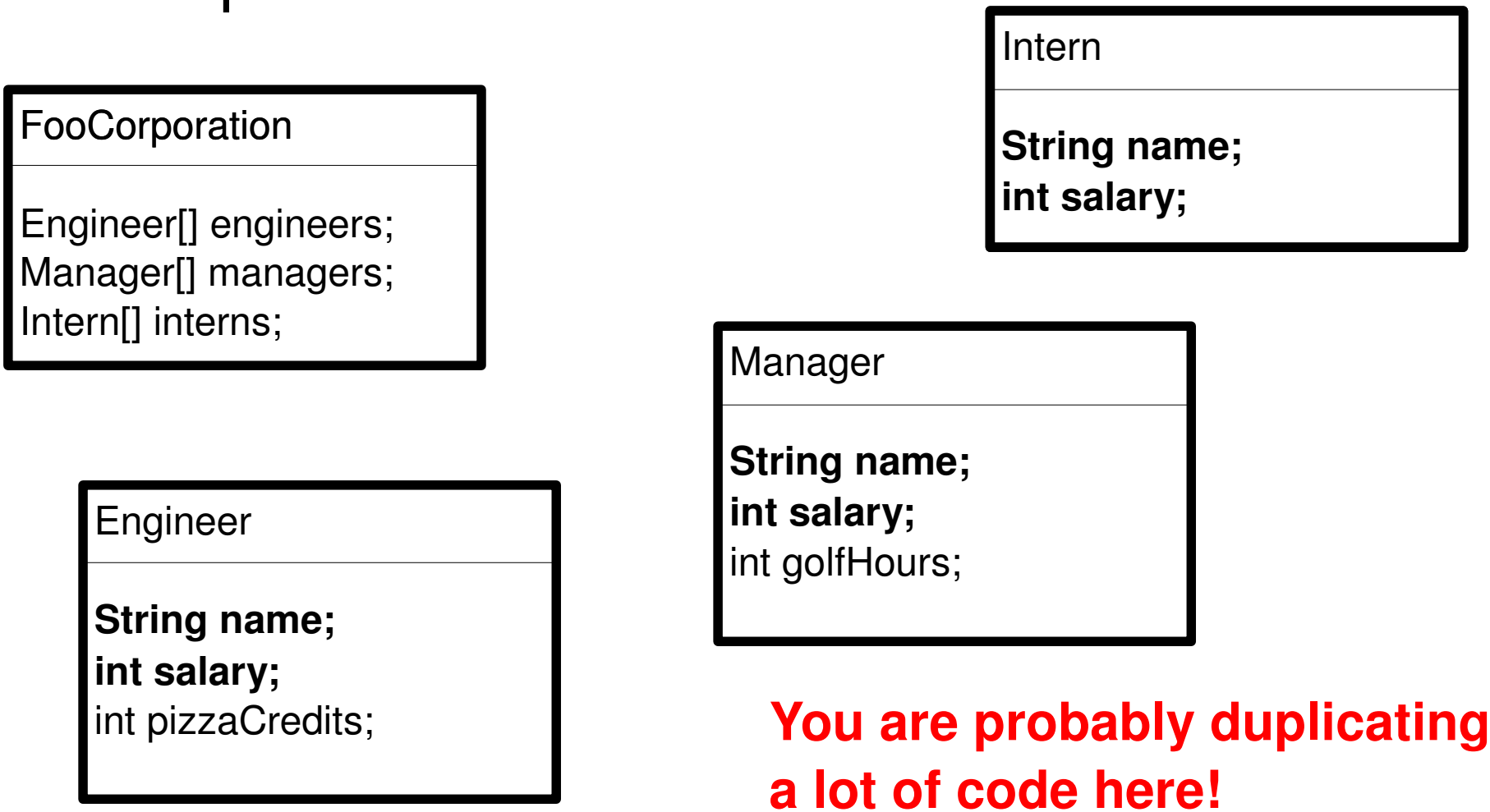
Object-oriented programming

- OOP helps you model the world on your computer



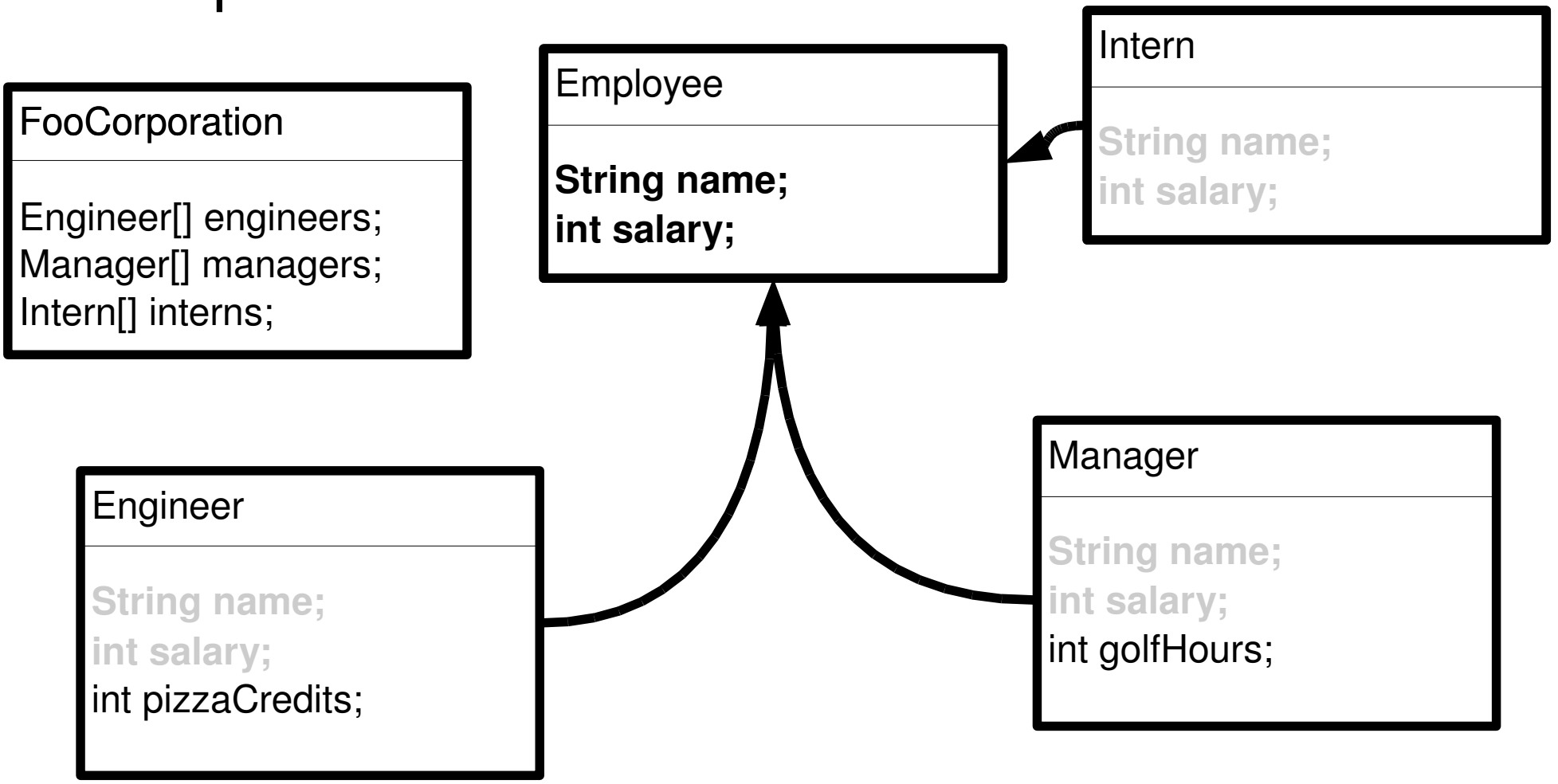
Object-oriented programming

- OOP helps you model the world on your computer



Object-oriented programming

- OOP helps you model the world on your computer



Inheritance

- The class Engineer and the class Manager **inherit** properties from a super-class (e.g. Employee)
- Write all the generic stuff in the Employee class
- Write manager-specific code in the Manager class
- Write engineer-specific code in the Engineer class

Inheritance

```
public class Employee {
    String name;
    int salary;
}

public class Manager extends Employee {
}
public class Engineer extends Employee {
}
```

Inheritance

- Classes inherits fields and methods from their parents

```
public class Employee {
    String name;
    int salary;

    public static void printSalary () {
        System.out.println ("Salary of " + name + " is " + salary);
    }
}
```

Inheritance

- Classes inherits fields and methods from their parents

You can now call printSalary() on a object of the Manager class!

```
public class World {
        public static void main (String[] args) {
            Manager m = new Manager (“Joshua”, 4000);
            m.printSalary ();
        }
}
```

this

- In Java, the keyword **this** refers to the current object.

```
public class Bicycle {
    int gear;

    public Bicycle (int gear) {
        this.gear = gear;
    }
}
```

Inheritance

- Sub-classes inherit the **default constructor** automatically (i.e. the constructor with no arguments)

```
public class Employee {
    String name;
    int salary;
    public Employee () {
        this.name = “Joe”;
        this.salary = 10000;
    }
}
```

```
public class Manager extends Employee {
 // no need for a constructor here
}
```

Inheritance

- Sub-classes do **not** inherit **non-default constructors** automatically (i.e. the constructor with arguments)

```
public class Employee {
    String name;
    int salary;
    public Employee (String name) {
        this.name = “Joe”;
        this.salary = 10000;
    }
}
```

```
public class Manager extends Employee {
 // need a constructor here!!
}
```

Inheritance

- If you define a constructor in a class, you **must** define it in all its subclasses.

```
public class Employee {
    String name;
    int salary;
    public Employee (String name, int salary) { // constructor
        this.name = name;
        this.salary = salary;
        System.out.println ("Created new employee " + name);
    }
}
```

Inheritance

- You can reuse the super-class constructor using **super**.

```
public class Manager extends Employee {
    String name;
    int salary;
    public Manager (String name, int salary) { // constructor
        super (name, salary);
        salary += 1000; // managers get bonus when hired
    }
}
```

Abstract classes

- Sometimes, the super-class should never be instantiated (i.e. no object of that class should exist)
- e.g. in FooCorporation, you are either an Engineer or a Manager, but not just an Employee
- An **abstract** class is a class than can never be implemented
- It may have **abstract methods** that have no body but also regular methods.
- An abstract method **must** be implemented in the subclasses.

Abstract classes

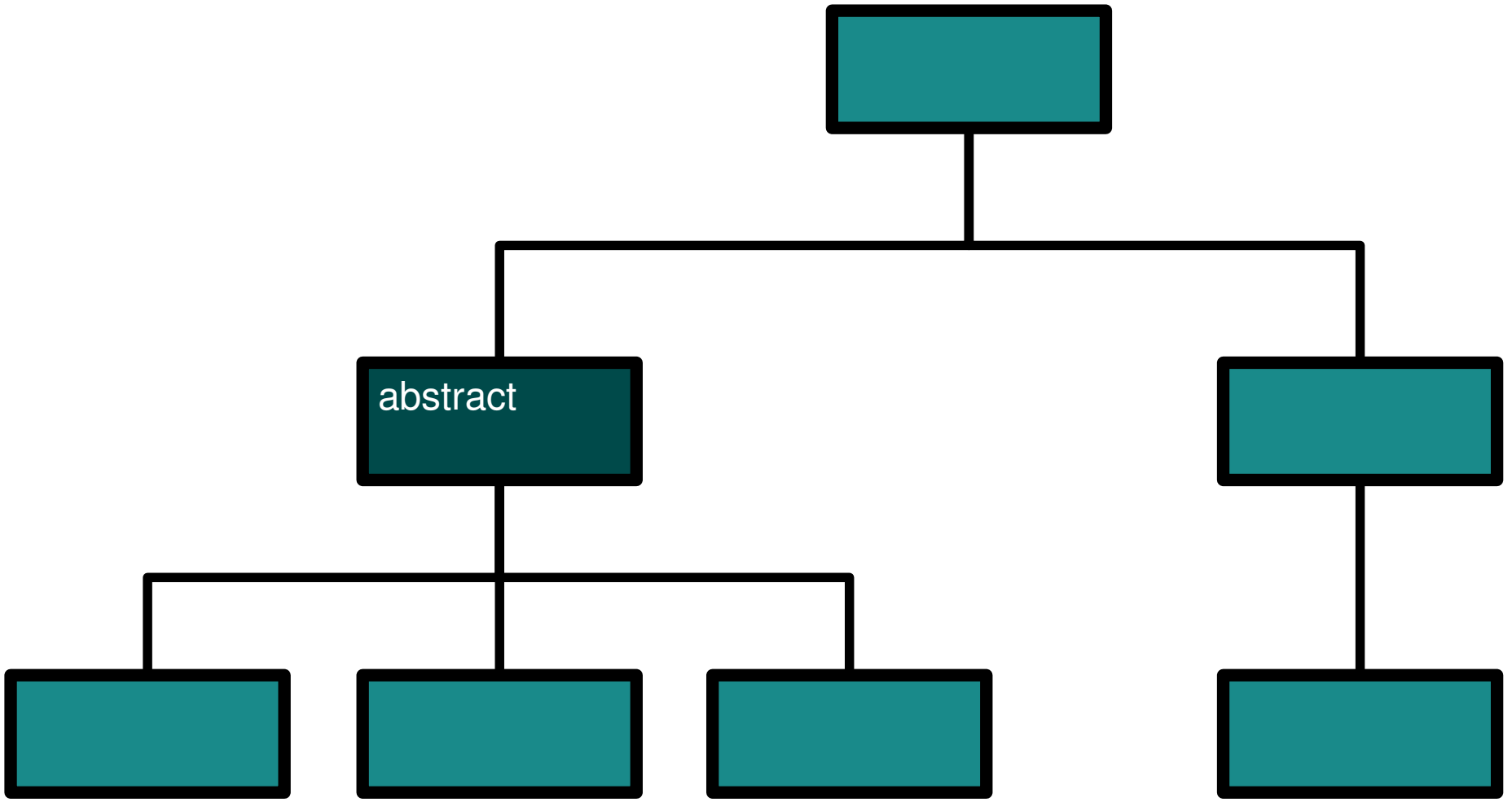
```
public abstract class Employee {
    String name;
    int salary;

    public void printSalary () { // regular method
        System.out.println ("My salary is " + salary);
    }

    public abstract void printPizzaCredits (); // abstract method
}
```

Abstract classes

```
public class Manager extends Employee {

    public Manager(String name, int salary) {
        super (name, salary);
    }

    public void printPizzaCredits () { // implements abstract method
        System.out.println ("No pizza credit for managers!");
    }

}
```

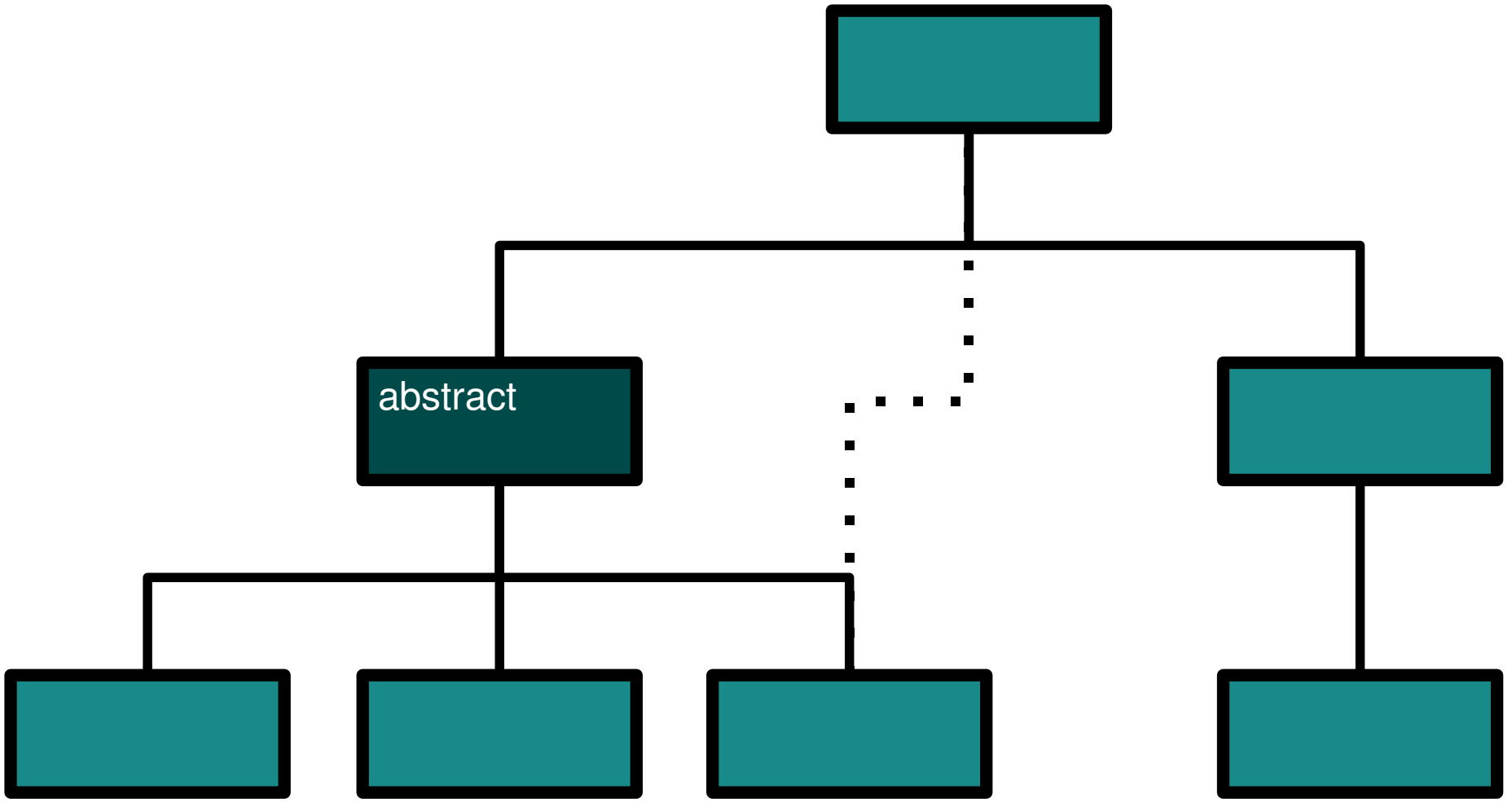
Inheritance and abstract classes

- Inheritance implements hierarchical structures.



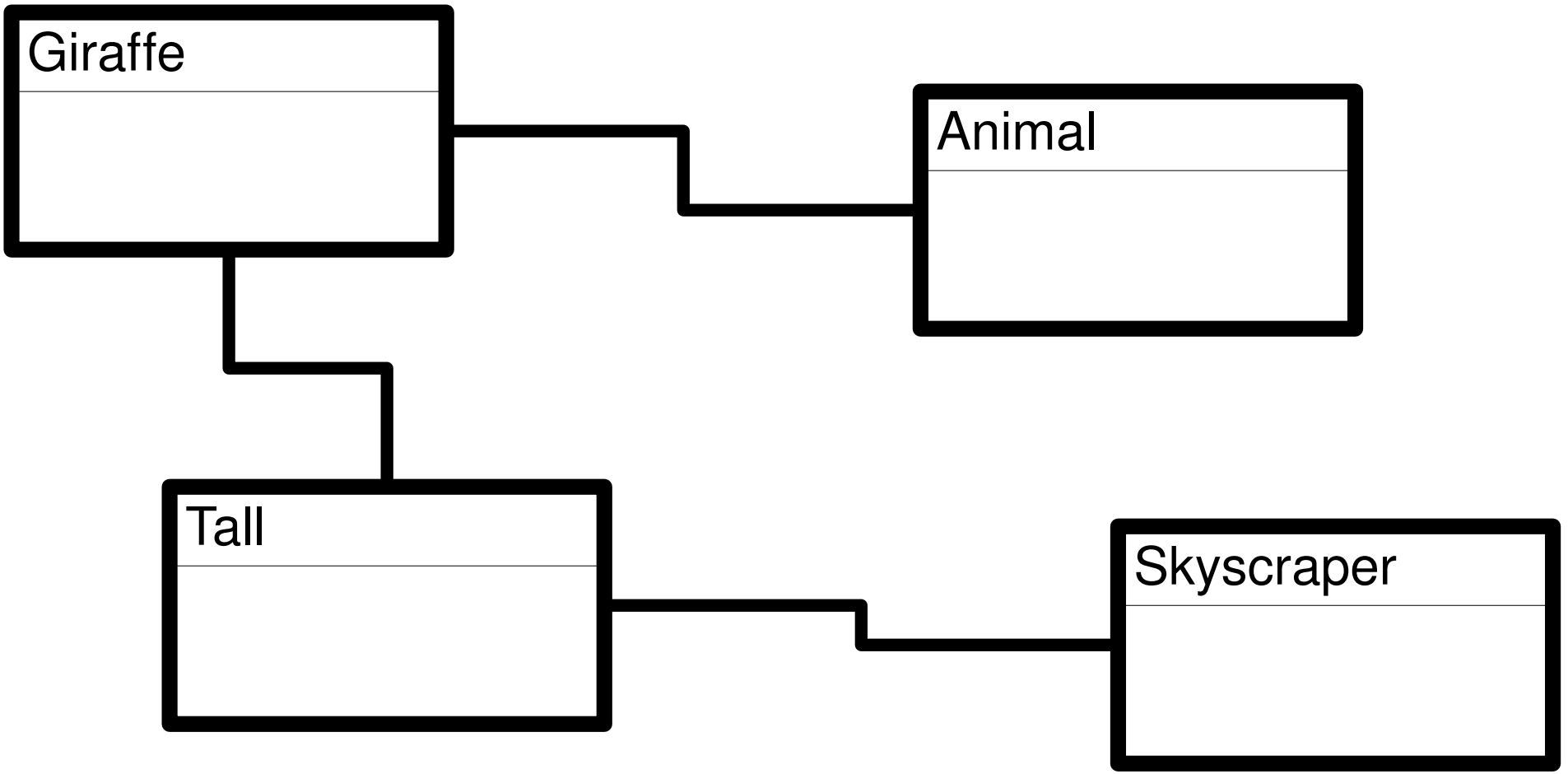
Inheritance and abstract classes

- Java does not allow multiple inheritance.



Interfaces

- Interfaces allow to tie different classes together.

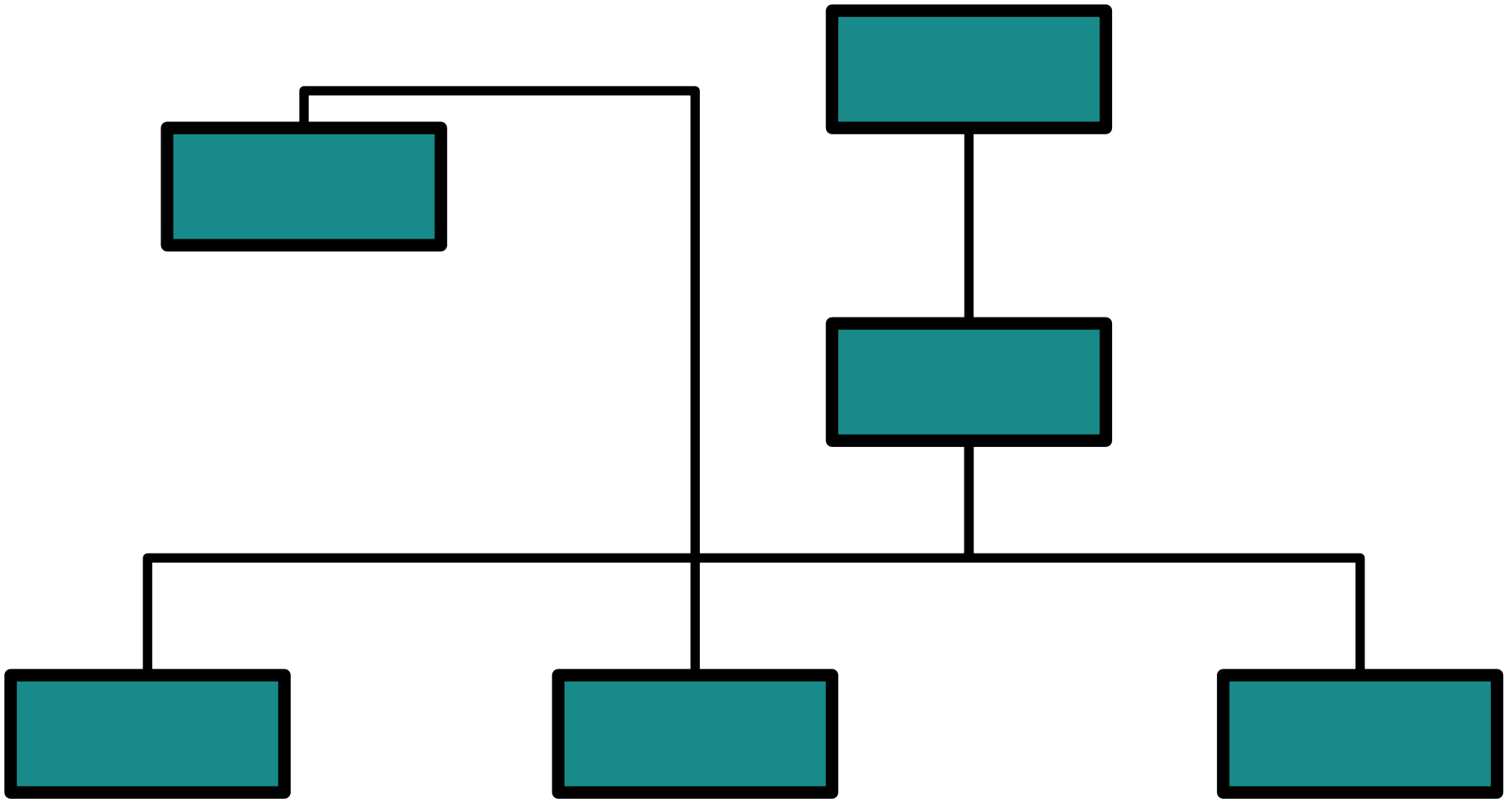


Interfaces

- An interface is like an abstract class, but **all methods are abstract** and **all fields are final**.
- All methods **must** be implemented in the sub-classes.
- You cannot change the value of an interface field.

Interfaces

- Interfaces implement network-like structures.



Polymorphism

- From *poly* (many) + *morph* (form)
- The ability for a method to behave differently depending on the object it is called upon.
  - void spinning (Ball b);
  - void spinning (Image g);

Polymorphism

- **Overloaded methods**
  - Same name, but different input or output, e.g.
    public void spinning (Ball b);
    public void spinning (Image g);
- **Overriden methods**
  - Redefined in a subclass with the same **signature** (same input, same output)

Polymorphism

- **Overloaded methods**

```
public class World {
    public static void fire (Employee e) {
        System.out.println ("Thank you!!");
     }
    public static void fire (Manager e) {
        System.out.println ("Here is $10,000");
     }
    public static void main (String[] args) {
     }
}
```

Polymorphism

- **Overriden methods**

```
public class Employee {
     public void getRaised (int raise) {
        salary += raise;
    }
}
public class Manager extends Employee {
     public void getRaised (int raise) {
        salary += 3 * raise;
    }
}
```

Summary

- Object-oriented programming
- Inheritance and abstract classes
- Interfaces
- Polymorphism

Assignment 6: Graphics strikes back

- Follow the instructions on the Stellar website
- **Respect the checkpoints.... Please!!!**
- Your goal is to apply the concepts of inheritance and polymorphism to the graphics application
- There is an **optional** section. It is not required to pass the assignment.