

Inheritance

Han-fen Hu

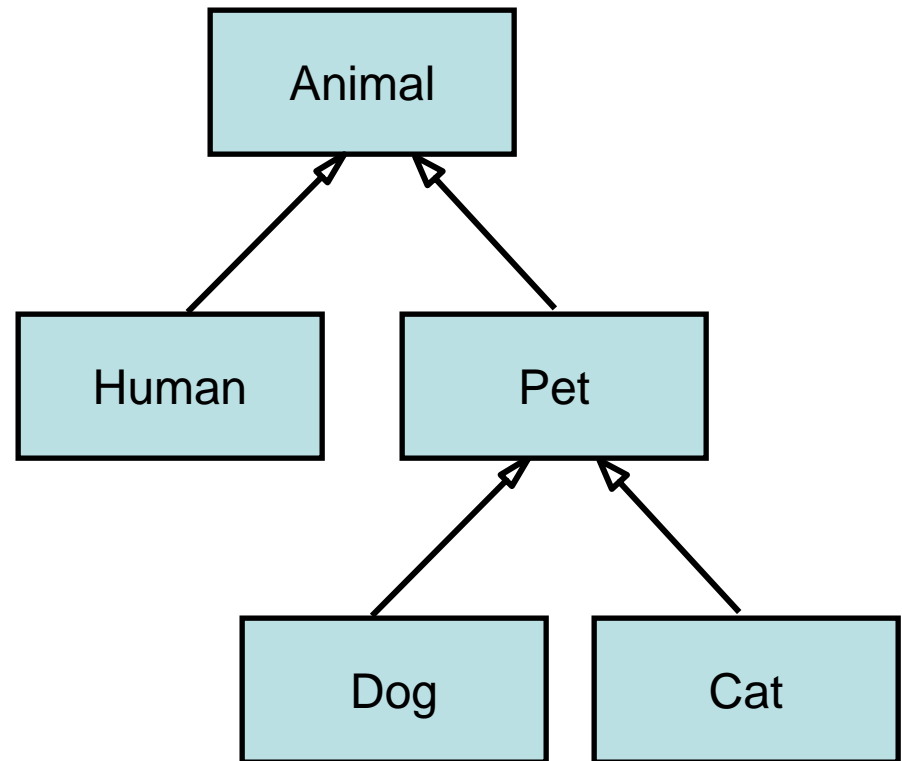
Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

Outline

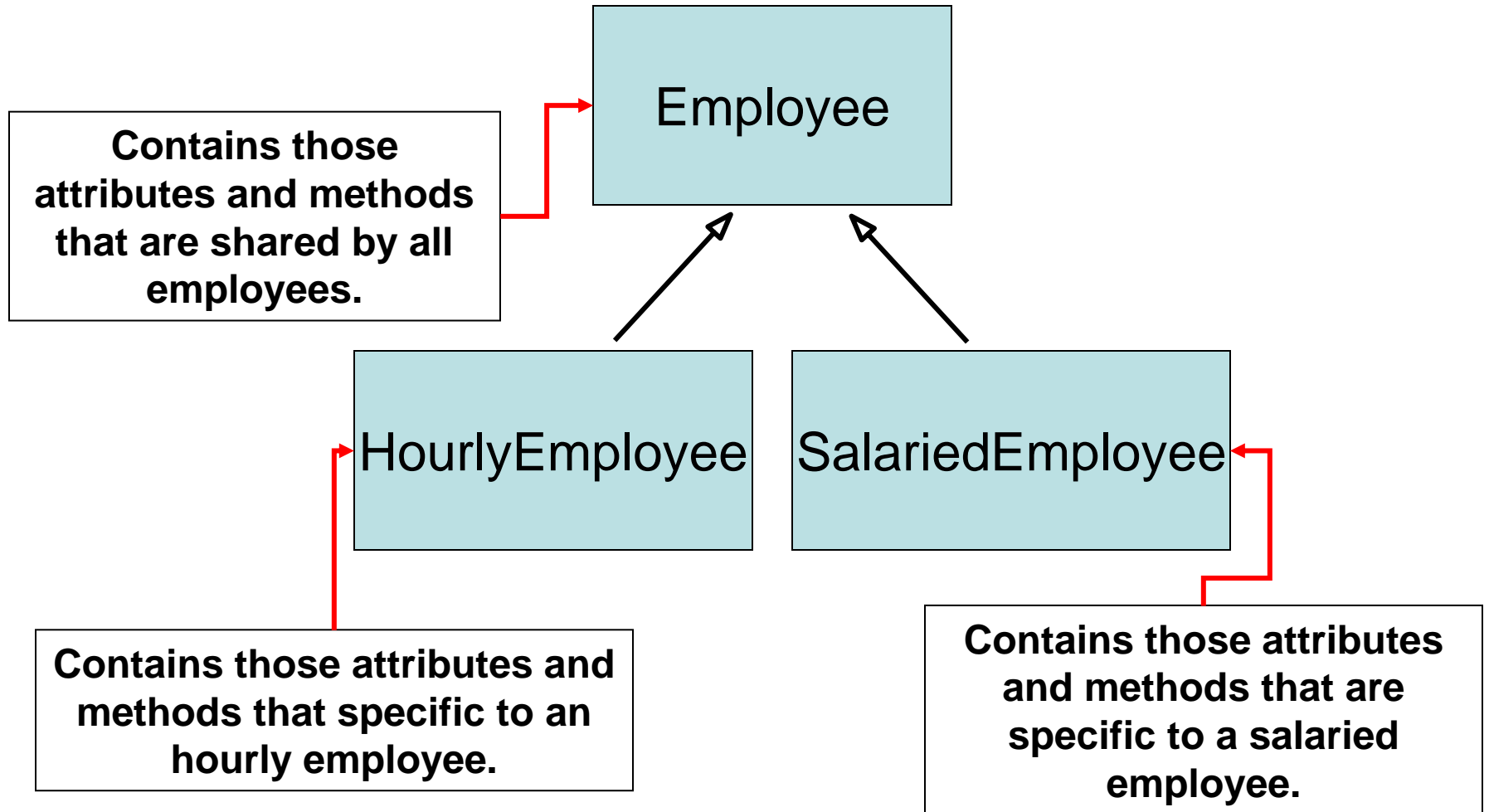
- ❑ Introduction to Inheritance
- ❑ Calling the Superclass Constructor
- ❑ Overriding Superclass Methods
- ❑ Protected Members
- ❑ Chains of Inheritance

Inheritance (1)

- ❑ We can effectively model similar, but different types of objects using inheritance



Inheritance (2)



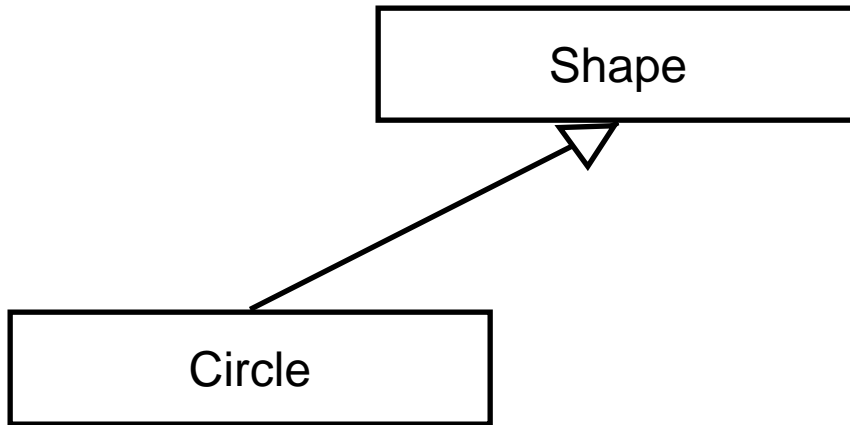
Inheritance (3)

- ❑ We can *extend* the capabilities of a class.
- ❑ Inheritance involves a superclass and a subclass.
 - The *superclass* is the general class and
 - the *subclass* is the specialized class.
- ❑ A specialized object has:
 - All of the characteristics of the general object, **plus**
 - Additional characteristics that make it special.

Inheritance (4)

- ❑ The subclass inherits fields and methods from the superclass without any of them being rewritten.
- ❑ New fields and methods may be added to the subclass.
- ❑ The Java keyword, *extends*, is used on the class header to define the subclass.

Inheritance (5)



```
public class Circle extends Shape {  
    .  
    .  
    .  
}
```

Example of Inheritance (1)

```
class Pet {  
    // Field  
    private String name;  
  
    // Constructors  
    public Pet () { }  
    public Pet(String newName){  
        name = newName  
    }  
  
    // Methods  
    public String getName() {  
        return name;  
    }  
    public void setName(String petName) {  
        name = petName;  
    }  
    public String speak( ) {  
        return "I'm your cuddly little pet.";  
    }  
}
```


Example of Inheritance (1)

```
class Cat extends Pet {  
    public String speak( ) {  
        String word = "Don't give me orders.\n"  
        return word;  
    }  
}
```

```
class Dog extends Pet {  
    public String fetch( ) {  
        return "Yes, master. Fetch I will.";  
    }  
}
```

Example of Inheritance (2)

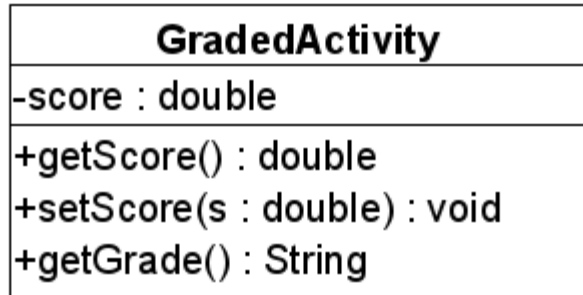
```
public class Employee {  
    // Fields  
    private String name; // name of the employee  
    private int experiencePoint; // experience value of the employee  
  
    /* Constructor  
     * set the default value for experience point*/  
    public Employee() {  
        experiencePoint = 0;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getExperiencePoint() {  
        return experiencePoint;  
    }  
  
    // add a certain number of points to the experience point  
    public boolean earnExperiencePoint(int numOfPoints) {  
        // if the given points is negative  
        // return an error to the calling method  
        if(numOfPoints < 0)  
            return false;  
        // if the given points is positive  
        // add the points  
        else {  
            experiencePoint += numOfPoints;  
            return true;  
        }  
    }  
}
```

Example of Inheritance (3)

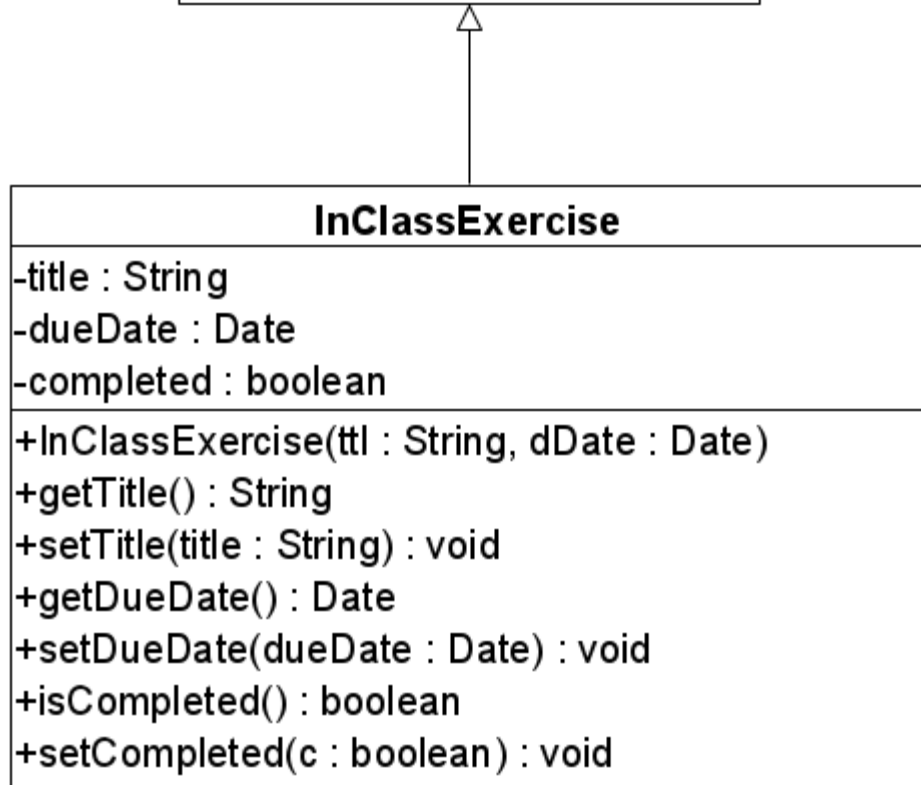
```
public class SalariedEmployee extends Employee {  
    // field  
    private double monthlySalary;  
  
    // calculate salary based on the number of hours given.  
    public double calcWeeklySalary() {  
        // get the salary for two weeks  
        return monthlySalary/4;  
    }  
}
```

```
public class HourlyEmployee extends Employee {  
    // constants  
    final double BASE_PAY = 25; // the base pay rate  
    final double REGULAR_HOURS = 40; // the regular working hours  
  
    // calculate salary based on the number of hours given.  
    public double calcWeeklySalary(int numOfHours) {  
        // a simplified formula. We will revise it when we talk about if-else statement  
        double salary = REGULAR_HOURS * BASE_PAY + (numOfHours-REGULAR_HOURS)*BASE_PAY*1.5;  
        return salary;  
    }  
}
```

Example: GradedActivity



Contains those attributes and methods that are shared by all graded activities.



Contains those attributes and methods that are specific to the **InClassExercise** class. Inherits all non-private attributes and methods from the **GradedActivity** class.

Lab (1)

- ❑ GradedActivity.java
- ❑ InClassExercise.java
- ❑ InClassExerciseDemo.java

Fields and Methods (1)

- ❑ Members of the superclass that are marked *private*:
 - are not inherited by the subclass,
 - exist in memory when the object of the subclass is created
 - may only be accessed from the subclass by public methods of the superclass.
- ❑ Members of the superclass that are marked *public*:
 - are inherited by the subclass, and
 - may be directly accessed from the subclass.

Fields and Methods (2)

- ❑ When an instance of the subclass is created,
 - Non-private methods of the superclass are available through the subclass object.
 - Example:

```
InClassExercise ex =  
    new InClassExercise("Ex. 3", sdf.parse("2021-02-24"));  
  
ex.setCompleted(true);  
  
System.out.println("Score = "+ex.getScore());
```

Constructors (1)

- ❑ When a subclass is instantiated, the superclass default constructor is executed first.
- ❑ Demo
 - SuperClass.java
 - SubClass.java
 - ConstructorDemo.java

Constructors (2)

- ❑ The `super` keyword refers to an object's superclass.
- ❑ The superclass constructor can be explicitly called from the subclass by using the `super` keyword.
- ❑ Calls to a superclass constructor must be the first java statement in the subclass constructors.

Constructors (3)

□ Calling the Superclass Constructor

- If a parameterized constructor is defined in the superclass
 - subclasses must provide a constructor, and
 - subclasses must call the superclass constructor

Lab (2)

❑ Rectangle.java

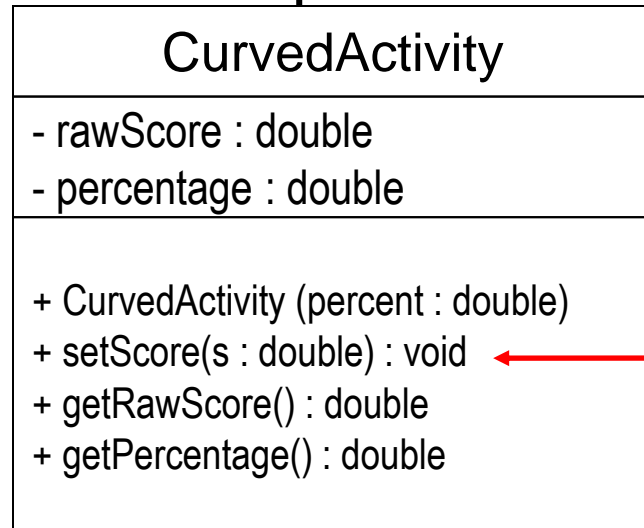
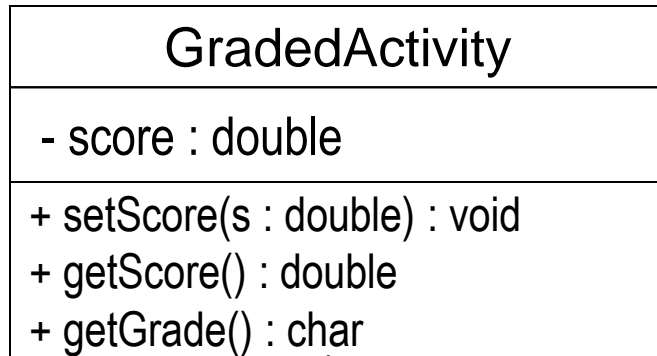
❑ Cube.java

❑ CubeDemo.java

Overriding Superclass Methods (1)

- ❑ A subclass may have a method with the same **signature** as a superclass method.
 - A method signature consists of the method's name and the data types of the method's parameters
- ❑ The subclass method overrides the superclass method.
- ❑ This is known as **method overriding**.

Overriding Superclass Methods (2)



This method is a more specialized version of the **setScore** method in the superclass, **GradedActivity**.

Overriding Superclass Methods (3)

- ❑ A subclass method that overrides a superclass method must have the **same signature** as the superclass method
 - An object of the subclass invokes the subclass's version of the method, not the superclass's.
- ❑ A subclass method can call the overridden superclass method via the `super` keyword.


```
super.setScore(rawScore * percentage);
```

Method Signature and Binding

- ❑ A method **signature** consists of the method's **name and the data types** of the method's parameters, in the order that they appear.

`add(int, int)`
`add(String, String)`

Signatures of the
add methods



- ❑ The return type is not part of the signature.
- ❑ The compiler uses the method signature to determine which version of the overloaded method to bind the call to.
 - The process of matching a method call with the correct method is known as **binding**.

Lab (3)

- ❑ GradedActivity.java
- ❑ CurvedActivity.java
- ❑ CurvedActivityDemo.java

Preventing a Method from Being Overridden

- ❑ The `final` modifier will prevent the overriding of a superclass method in a subclass.

```
public final void message()
```

- ❑ If a subclass attempts to override a final method, the compiler generates an error.
- ❑ This ensures that a particular superclass method is used by subclasses rather than a modified version of it.

Protected Members (1)

- ❑ Java provides a third access specification, **protected**.
 - A *protected* member's access is somewhere between *private* and *public*.
- ❑ Protected members of class:
 - may be accessed by methods in a subclass, and
 - by methods in the same package as the class.

Protected Members (2)

- ❑ Using **protected** instead of **private** makes some tasks easier.
 - Any class that is derived from the class, or is in the same package, has unrestricted access to the protected member.
- ❑ It is always better to make all fields **private** and then provide **public** methods for accessing those fields.
- ❑ If no access specifier for a class member is provided, the class member is given *package access* by default.
 - Any method in the same package may access the member.

Access Specifiers

Access Modifier	Accessible to a subclass inside the same package?	Accessible to all other classes inside the same package?
default (no modifier)	Yes	Yes
Public	Yes	Yes
Protected	Yes	Yes
Private	No	No

Access Modifier	Accessible to a subclass outside the package?	Accessible to all other classes outside the package?
default (no modifier)	No	No
Public	Yes	Yes
Protected	Yes	No
Private	No	No

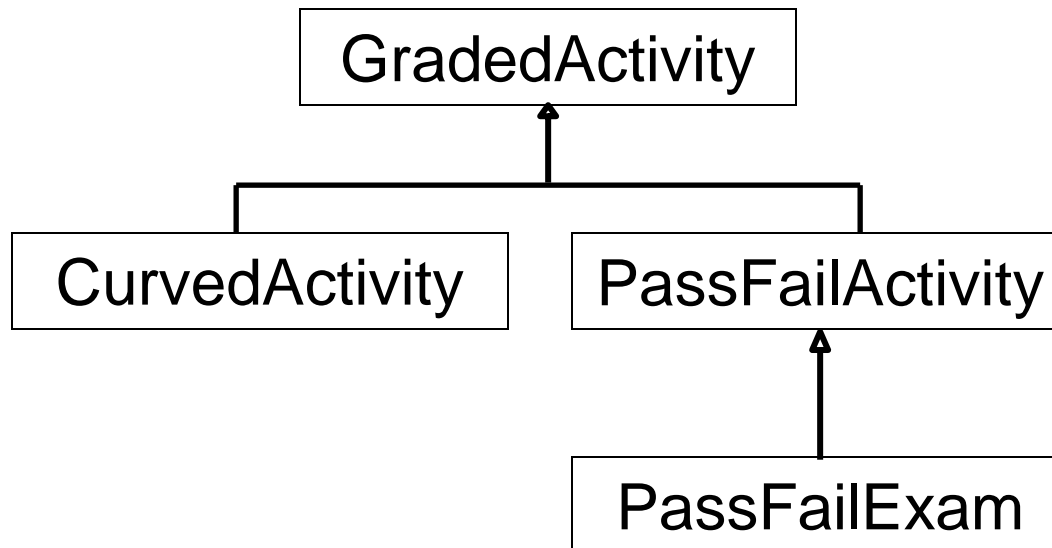
Lab (4)

❑ GradedActivityProtected.java

❑ CurvedActivityProtected.java

Chains of Inheritance

- ❑ A superclass can also be derived from another superclass.



Lab (5)

- ❑ PassFailActivity.java
- ❑ PassFailExam.java
- ❑ PassFailExamDemo.java

Exercise

❑ Manager Class

- Based on the superclass **Staff**, please implement **Manager** class
 - The field level ranges between 1-5. If the value is not within the range, set the value to 1.
 - Getter and setter for the field should also be provided.
 - Please override the calcSalary method. A manager's salary is calculated as regular salary plus a stipend. The stipend is \$150 per level (e.g., \$750 for a level 5 manager).
- Please create an application class to test your **Manager** class.
- The **Staff** class is included in the lab files.

