CMPE 101
Object Oriented Programming

İstanbul
Bilgi University

Dr. Emel Küpçü

Department of Computer Engineering

İstanbul Bilgi University

Week-9: Programming with Inheritance

9.1 Inheritance

- **It is a feature that allows a new class** (called a ***subclass*** or ***derived class***) **to acquire the properties and behaviors** (**fields** and **methods**) **of an existing class** (called a ***superclass*** or ***base class***).
- The new class can then add new features or modify existing ones to better suit its needs.
- **Derived classes** can be created from existing ones; those classes are said to inherit the methods and instance variables of the class they inherited which is **base class**.
  - A derived class is also called a **subclass** or **child class**
  - A base class is also called **superclass** or **parent class**
- It is used to promote code reuse and establish a relationship between classes.

Inheritance (Cont.)

- The **extends** clause in a class declaration establishes an inheritance relationship between two classes.
- The syntax:

```
class BaseClass{
//I body of the class
}
class DerivedClass extends BaseClass{
  //I body of the class
}
```

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Inheritance (Cont.)

This concept offers several benefits:

- **Saves development time:** By reusing well-tested, existing code from the base class, developers can build new functionality more quickly and with fewer errors.
- **Improves code quality:** Since the base class is already debugged and reliable, the subclass inherits that stability, reducing bugs and improving software quality.
- **Enhances maintainability:** Systems built with inheritance are often easier to understand and maintain, as they promote code organization and reduce duplication.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

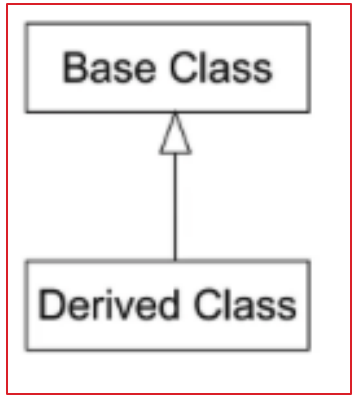
Inheritance (Cont.)

- When creating a class, rather than declaring completely new members, you can designate that the new class should *inherit* the members of an existing class.
  - Existing class is the superclass
  - New class is the subclass
- A subclass can be a superclass of future subclasses.
- A subclass can add its own fields and methods.
- A subclass is more specific than its superclass and represents a more specialized group of objects.
- The subclass exhibits the behaviors of its superclass and can add behaviors that are specific to the subclass.
  - This is why inheritance is sometimes referred to as specialization.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

9.1 Introduction (Cont.)

- The direct superclass is the superclass from which the subclass explicitly inherits.
- An indirect superclass is any class above the direct superclass in the class hierarchy.
- The Java class hierarchy begins with class **`Object`** (in package `java.lang`)
  - *Every* class in Java directly or indirectly extends (or "inherits from") `Object`.
- Java supports only single inheritance, in which each class is derived from exactly one direct superclass.
  - **Single Inheritance**: One child class inherits from one parent class.



Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

9.2 Superclasses and Subclasses

- Figure 9.1 lists several simple examples of superclasses and subclasses
  - Superclasses tend to be “more general” and subclasses “more specific.”
- Since every subclass is also a type of its superclass, and one superclass can have many subclasses, a superclass usually represents more objects than any single subclass.

Superclass	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`, `Sphere`, `Cube`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`
`BankAccount`	`CheckingAccount`, `SavingsAccount`

Fig. 9.1 | Inheritance examples.

Superclasses and Subclasses (Cont.)

- A superclass exists in a hierarchical relationship with its subclasses.
- Fig. 9.2 shows a sample university community class hierarchy
  - Also called an inheritance hierarchy.
- Each arrow in the hierarchy represents an *is-a relationship.*
- Follow the arrows upward in the class hierarchy
  - an `Employee` *is a* `CommunityMember`"
  - "a `Teacher` *is a* `Faculty` member."
- `CommunityMember` is the direct superclass of `Employee`, `Student` and `Alumnus`,`CommunityMember` is an indirect superclass of all the other classes in the diagram.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

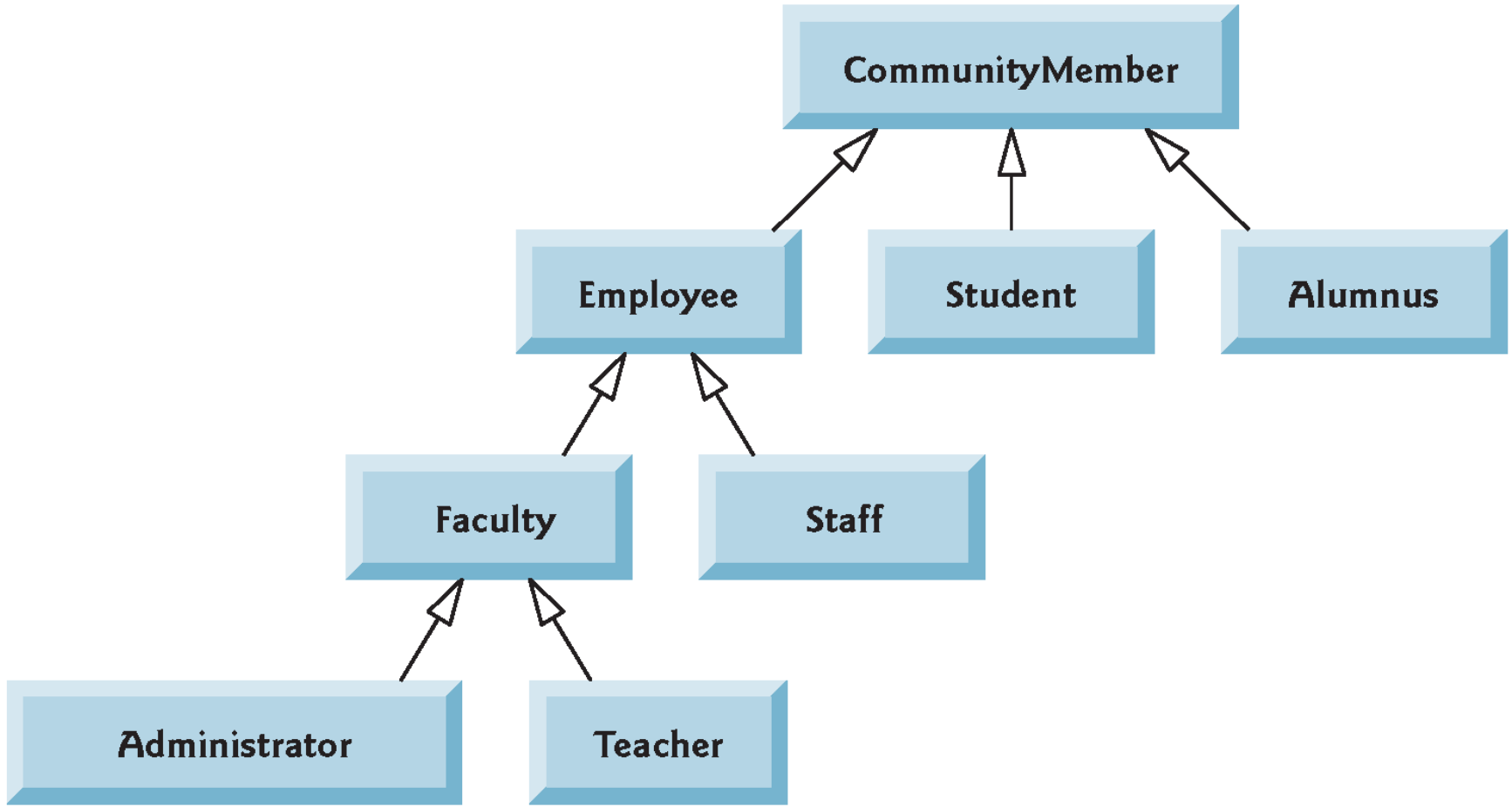


Fig. 9.2 | Inheritance hierarchy UML class diagram for university `CommunityMembers`.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Superclasses and Subclasses (Cont.)

- Fig. 9.3 shows a Shape inheritance hierarchy.
- Van follow the arrows from the bottom of the diagram to the topmost superclass in this class hierarchy to identify several *is-a* relationships.
  - A `Triangle` *is a* `TwoDimensionalShape` and *is a* `Shape`
  - A`Sphere` *is a* `ThreeDimensionalShape` and *is a* `Shape`.

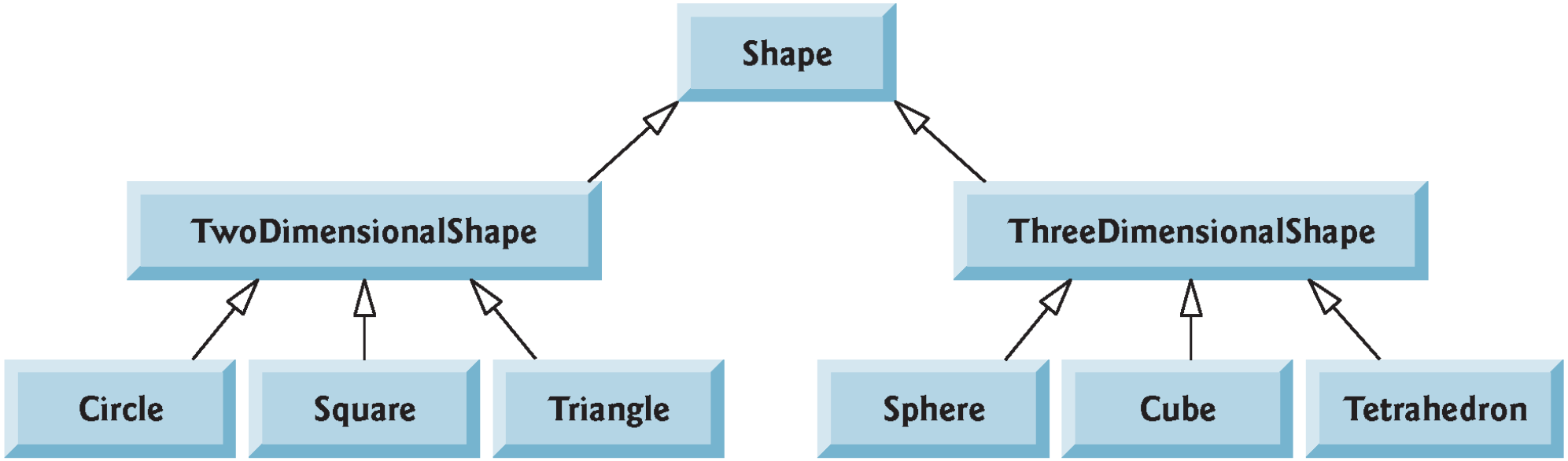


Fig. 9.3 | Inheritance hierarchy UML class diagram for Shapes.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Superclasses and Subclasses (Cont.)

- Objects of all classes that extend a common superclass can be treated as objects of that superclass.
  - Commonality expressed in the members of the superclass.
- Inheritance issue
  - A subclass can inherit methods that it does not need or should not have.
  - Even when a superclass method is appropriate for a subclass, that subclass often needs a customized version of the method.
  - The subclass can override (redefine) the superclass method with an appropriate implementation.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Inheritance Example

```java
// Parent class, superclass
class Animal {
    String name;

    public void eat() {
        System.out.println(name + " is eating...");
    }
    public void sleep() {
        System.out.println(name + " is sleeping...");
    }
}
// Child class, subclass
class Dog extends Animal {

    public void bark() {
        System.out.println(name + " is barking...");
    }
}
```

```java
// Main class to test the inheritance
public class Main_AnimalDog {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.name = "Buddy";

        myDog.eat();    // Inherited from Animal
        myDog.sleep();  // Inherited from Animal
        myDog.bark();   // Defined in Dog
    }
}
```

Output:
Buddy is eating...
Buddy is sleeping...
Buddy is barking...

9.3 protected Members

- A class's `public` members are accessible wherever the program has a reference to an object of that class or one of its subclasses.
- A class's `private` members are accessible only within the class itself.
- `protected` access is an intermediate level of access between `public` and `private`.
  - A superclass's `protected` members can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the *same package*
    - `protected` members also have package access.
  - When a class inherits from another class, all public and protected members (like variables and methods) from the superclass are accessible in the subclass, and they keep the same access level:
    - **public** members are accessible everywhere (in and outside the class).
    - **protected** members are accessible in the subclass and in the same package.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Inheritance Example 2

```java
// Superclass
class Animals {
    public String name = "Generic Animal";
    protected int age = 5;

    public void makeSound() {
        System.out.println("Some sound...");
    }
    protected void sleep() {
        System.out.println("Animal is sleeping...");
    }
}
// Subclass
class Cat extends Animals {
    public void showInfo() {
        System.out.println("Name: " + name);   // public member
        System.out.println("Age: " + age);     // protected member
        makeSound();                           // public method
        sleep();                               // protected method
    }
}
```

```java
// Main class
public class AnimalCat {
    public static void main(String[] args) {
        Cat myCat = new Cat();
        myCat.showInfo();
    }
}
```

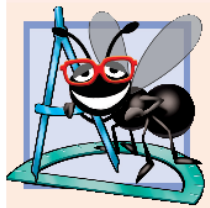
Output:

Name: Generic Animal
Age: 5
Some sound...
Animal is sleeping...

protected Members (Cont.)

- A superclass's `private` members are *hidden* from its subclasses
  - They can be accessed only through the `public` or `protected` methods inherited from the superclass
- Subclass methods can refer to `public` and `protected` members inherited from the superclass simply by using the member names.
- When a subclass method *overrides* an inherited superclass method, the *superclass* version of the method can be accessed from the *subclass* by preceding the superclass method name with keyword `super` and a dot (`.`) separator.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved



Software Engineering Observation 9.1

Methods of a subclass cannot directly access `private` members of their superclass. A subclass can change the state of `private` superclass instance variables only through non-`private` methods provided in the superclass and inherited by the subclass.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Inheritance Example 3

```
class Faculty {
    private String id = "F123";          // private
    protected String department = "CS";    // protected

    public String getId() {
        return id;
    }
    public void introduce() {
        System.out.println("I'm part of the faculty.");
    }
}
class Professor extends Faculty {
    @Override
    public void introduce() {
        super.introduce();                    // call superclass method
        System.out.println("I teach in " + department); // access protected
        System.out.println("Faculty ID: " + getId());   // access private via public method
    }
}
```

```
public class MainFaculty {
    public static void main(String[] args) {
        Professor prof = new Professor();
        prof.introduce();
    }
}
```

Output:
I'm part of the faculty.
I teach in CS
Faculty ID: F123

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

9.5 Constructors in Subclasses

- Instantiating a subclass object begins a chain of constructor calls
  - The subclass constructor, before performing its own tasks, explicitly uses `super` to call one of the constructors in its direct superclass or implicitly calls the superclass's default or no-argument constructor
- If the superclass is derived from another class, the superclass constructor invokes the constructor of the next class up the hierarchy, and so on.
- The last constructor called in the chain is *always* `Object`'s constructor.
- Original subclass constructor's body finishes executing *last*.
- Each superclass's constructor manipulates the superclass instance variables that the subclass object inherits.

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved

Example - Inheritance

Output:
Dr. Smith is working in the hospital.
Dr. Smith is treating a patient in the Cardiology department.
Nurse Jane is working in the hospital.
Nurse Jane is assisting a doctor during a 8-hour shift.

```
class HospitalStaff {
    String name;
    int age;

    public HospitalStaff(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void work() {
        System.out.println(name
                + " is working in the hospital.");
    }
}
```

Base Class

```
class Doctor extends HospitalStaff {
    String specialty;
    public Doctor(String name, int age, String specialty) {
        super(name, age);
        this.specialty = specialty;
    }
    void treatPatient() {
        System.out.println(name + " is treating a patient in the "
                + specialty + " department.");
    }
}
```

Child Class

```
class Nurse extends HospitalStaff {
    int shiftHours;

    public Nurse(String name, int age, int shiftHours) {
        super(name, age); // Call parent class constructor
        this.shiftHours = shiftHours;
    }
    void assistDoctor() {
        System.out.println(name
                + " is assisting a doctor during a "
                + shiftHours + "-hour shift.");
    }
}
```

Child Class

```
public class Main_BasicInheritance {
    public static void main(String[] args) {
        // Creating an object for Doctor
        Doctor doctor = new Doctor("Dr. Smith", 45, "Cardiology");
        doctor.work();          // Calling inherited method
        doctor.treatPatient(); // Calling Doctor-specific method

        // Creating an object for Nurse
        Nurse nurse = new Nurse("Nurse Jane", 30, 8);
        nurse.work();          // Calling inherited method
        nurse.assistDoctor();  // Calling Nurse-specific method
    }
}
```

Main Class

What will be the output of the following Java program?

```
class Parent {
    public Parent() {
        System.out.println("Parent constructor");
    }
}

class Child extends Parent {
    public Child() {
        System.out.println("Child constructor");
    }
}

public class ConstructorTest {
    public static void main(String[] args) {
        Child c = new Child();
    }
}
```

A.

Child constructor

Parent constructor

B.

Parent constructor

Child constructor

C.

Child constructor

D.

Parent constructor

Group Discussion Question

Implement a Java class hierarchy to represent students in a university system using inheritance. Follow the specifications below:

Create a class named Person2 with:

1. A **protected** instance variable name (String).
2. A constructor that accepts a name parameter and initializes the variable.

Create a class named Student2 that:

1. **Extends** the Person2 class.
2. Has a **protected** instance variable major (String).
3. A constructor that accepts name and major as parameters, calls the superclass constructor, and initializes major.

Create a class named BachelorStudent that:

1. **Extends** the Student2 class.
2. Has a **protected** instance variable currentYear (int).
3. A constructor that accepts name, major, and currentYear, calls the superclass constructor, and initializes currentYear.

Add a method displayInfo() to the BachelorStudent class that prints the following:

1. Name: <student's name>
2. Major: <student's major>
3. Current Year: <student's current year>

In a class named MainPerson, create an instance of BachelorStudent with the following values:

1. Name: "Emily”
2. Major: "Software Engineering”
3. Current Year: 2
   Then call displayInfo() on the created object.

Questions?

Copyright © 2018 Pearson Education, Ltd. All Rights Reserved