INHERITANCE

Ms. Layal Kazma

Learning Outcomes

- How inheritance promotes software reusability.
- The notions of superclasses and subclasses and the relationship between them.
- To use keyword extends to create a class that inherits attributes and behaviors from another class.
- To access superclass members with super.
- How constructors are used in inheritance hierarchies.
- The methods of class Object, the direct or indirect superclass of all classes.

Introduction

- Inheritance
  - *A form of software reuse in which a new class is created by absorbing an existing class's members and embellishing them with new or modified capabilities.*
  - *Can save time during program development by basing new classes on existing proven and debugged high-quality software.*
  - *Increases the likelihood that a system will be implemented and maintained effectively.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Introduction (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*
- Each 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 1992-2012 by Pearson Education, Inc. All Rights Reserved.

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.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Inheritance

- **Generic** are usually the **superclasses**
- **Specific** are **subclasses** that implement specific behavior
- Why inheritance is useful?
  - *to **reduce** the size of the code and development time*
  - *make the code modification easier and more efficient*
  - *improve **modularity** and promote software **reuse***

Superclasses and Subclasses

- Figure 9.1 lists several simple examples of superclasses and subclasses
  - *Superclasses tend to be "more general" and subclasses "more specific."*
- Because every subclass object *is an* object of its superclass, and one superclass can have many subclasses, the set of objects represented by a superclass is typically larger than the set of objects represented by any of its subclasses.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

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.

© Copyright 1992-2012 by Pearson Education, Inc. 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`.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

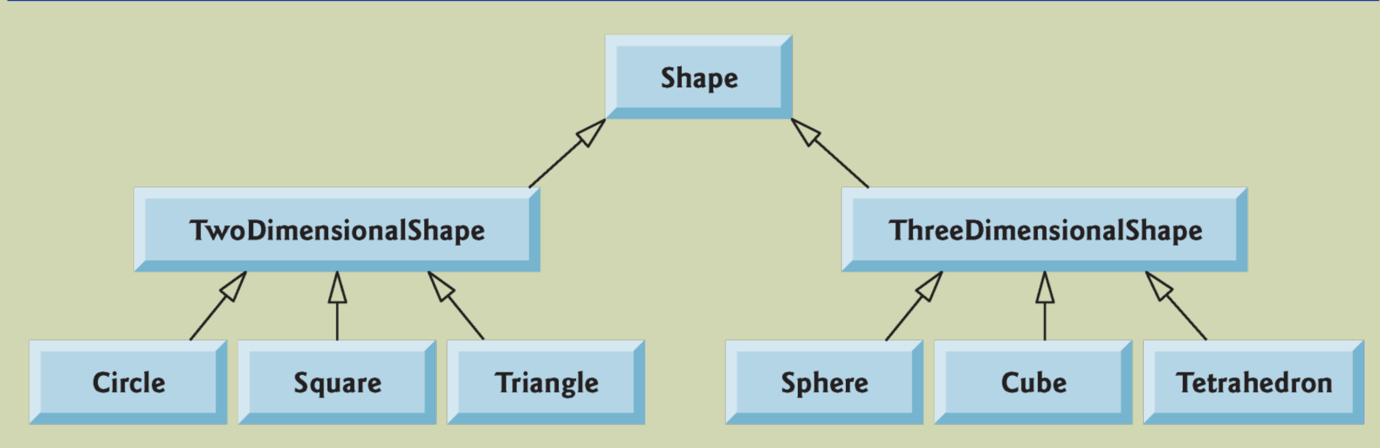

Fig. 9.3 | Inheritance hierarchy for Shapes.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Relationship

- We distinguish between the is-a relationship and the has-a relationship
- *Is-a* represents inheritance
  - *In an is-a relationship, an object of a subclass can also be treated as an object of its superclass*
- *Has-a* represents composition
  - *In a has-a relationship, an object contains as members references to other objects*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Superclasses and Subclasses (Cont.)

- Not every class relationship is an inheritance relationship.
- *Has-a* relationship
  - *Create classes by composition of existing classes.*
  - *Example: Given the classes* `Employee`*,* `BirthDate` *and* `TelephoneNumber`*, it's improper to say that an* `Employee` *is a* `BirthDate` *or that an* `Employee` *is a* `TelephoneNumber`*.*
  - *However, an* `Employee` *has a* `BirthDate`*, and an* `Employee` *has a* `TelephoneNumber`*.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Superclasses and Subclasses (Cont.)

- Objects of all classes that extend a common superclass can be treated as objects of that 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 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Example Without Using Inheritance

- In what follows, we have two classes of CommissionEmployee, and BasePlusComissionEmployee.
- The only difference is that the BasePlusComissionEmployee has a base salary in addition to attributes of ComissionEmployee, and thus its earning method is slightly different.
- The code shows the implementation of both classes without using inheritance.
- Observe the following: There is a lot of redundancy in both codes
- BasePlusComissionEmployee has more than 110 lines of code.
- Certainly, this programming style is not efficient. It is also not easy to modify the code of the related classes.
- Solution: using inheritance as shown in the next section

See code Without Using Inheritance

Relationship between Superclasses and Subclasses

- Inheritance hierarchy containing types of employees in a company’s payroll application
- Commission employees are paid a percentage of their sales
- Base-salaried commission employees receive a base salary plus a percentage of their sales.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Example Using Inheritance

- Considering the drawbacks of the first approach, we now use an important concept in Object Oriented Programming that is inheritance. Observe the following:
- The class CommissionEmployee, that is the generic class (or superclass) remains unchanged (same code as above)
- The class BasePlusComissionEmployee now extends the class CommissionEmployee
- How much code is reduced with using inheritance: 50 lines with inheritance as compared to more than 110 lines without using inheritance in the BasePlusComissionEmployee

See Code **Using Inheritance**

Creating and Using a `CommissionEmployee` Class

- Constructors are not inherited.
- The first task of a subclass constructor is to call its direct superclass's constructor explicitly or implicitly
  - *Ensures that the instance variables inherited from the superclass are initialized properly.*
- If the code does not include an explicit call to the superclass constructor, Java implicitly calls the superclass's default or no-argument constructor.
- A class's default constructor calls the superclass's default or no-argument constructor.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating and Using a `CommissionEmployee` Class (Cont.)

- To override a superclass method, a subclass must declare a method with the same signature as the superclass method
- @Override annotation
  - *Indicates that a method should override a superclass method with the same signature.*
  - *If it does not, a compilation error occurs.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating and Using a `BasePlus-CommissionEmployee` Class

- Class `BasePlusCommissionEmployee` contains a first name, last name, social security number, gross sales amount, commission rate and base salary.
  - *All but the base salary are in common with class `CommissionEmployee`.*
- Class `BasePlusCommissionEmployee`'s `public` services include a constructor, and methods `earnings`, `toString` and *get* and *set* for each instance variable
  - *Most of these are in common with class `CommissionEmployee`.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating and Using a `BasePlus-CommissionEmployee` Class (Cont.)

- Much of `BasePlusCommissionEmployee`'s code is similar, or identical, to that of `CommissionEmployee`.
- `private` instance variables `firstName` and `lastName` and methods `setFirstName`, `getFirstName`, `setLastName` and `getLastName` are identical.
  - *Both classes also contain corresponding get and set methods.*
- The constructors are almost identical
  - *`BasePlusCommissionEmployee`'s constructor also sets the `base-Salary`.*
- The `toString` methods are nearly identical
  - *`BasePlusCommissionEmployee`'s `toString` also outputs instance variable `baseSalary`*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating and Using a `BasePlus-CommissionEmployee` Class (Cont.)

- We literally *copied* `CommissionEmployee`'s code, pasted it into `BasePlusCommissionEmployee`, then modified the new class to include a base salary and methods that manipulate the base salary.
  - *This "copy-and-paste" approach is often error prone and time consuming.*
  - *It spreads copies of the same code throughout a system, creating a code-maintenance nightmare.*

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy

- Class `BasePlusCommissionEmployee` class extends class `CommissionEmployee`
- A `BasePlusCommissionEmployee` object *is a* `CommissionEmployee`
  - *Inheritance passes on class `CommissionEmployee`'s capabilities.*
- Class `BasePlusCommissionEmployee` also has instance variable `baseSalary`.
- Subclass `BasePlusCommissionEmployee` inherits `CommissionEmployee`'s instance variables and methods

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (Cont.)

- Each subclass constructor must implicitly or explicitly call its superclass constructor to initialize the instance variables inherited from the superclass.
  - *Superclass constructor call syntax—keyword* ***`super`****, followed by a set of parentheses containing the superclass constructor arguments.*
  - *Must be the first statement in the subclass constructor's body.*
- If the subclass constructor did not invoke the superclass's constructor explicitly, Java would attempt to invoke the superclass's no-argument or default constructor.
  - *Class `CommissionEmployee` does not have such a constructor, so the compiler would issue an error.*
- You can explicitly use `super()` to call the superclass's no-argument or default constructor, but this is rarely done.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating a `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy (Cont.)

- Compilation errors occur when the subclass attempts to access the superclass's `private` instance variables.
- These lines could have used appropriate *get* methods to retrieve the values of the superclass's instance variables.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.