Inheritance

Chapter 3 - Inheritance

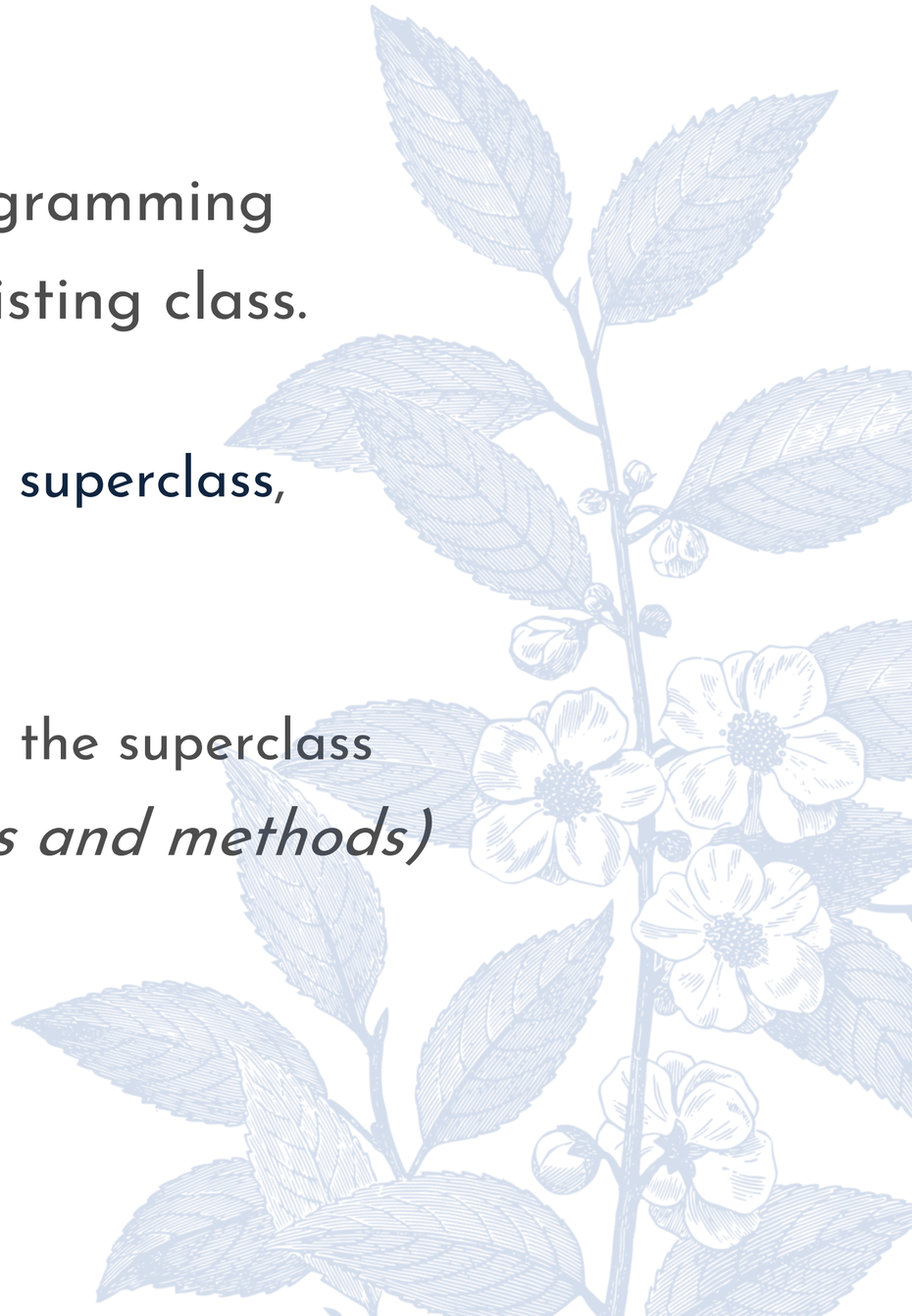
Chapter 3 – Inheritance

Chapter Goals

- ✓What Is Inheritance?
- ✓Calling the Superclass Constructor
- ✓Overriding Superclass Methods
- ✓Protected Members
- ✓Chains of Inheritance
- ✓The Object Class
- ✓Abstract Classes and Abstract Methods

What Is Inheritance?

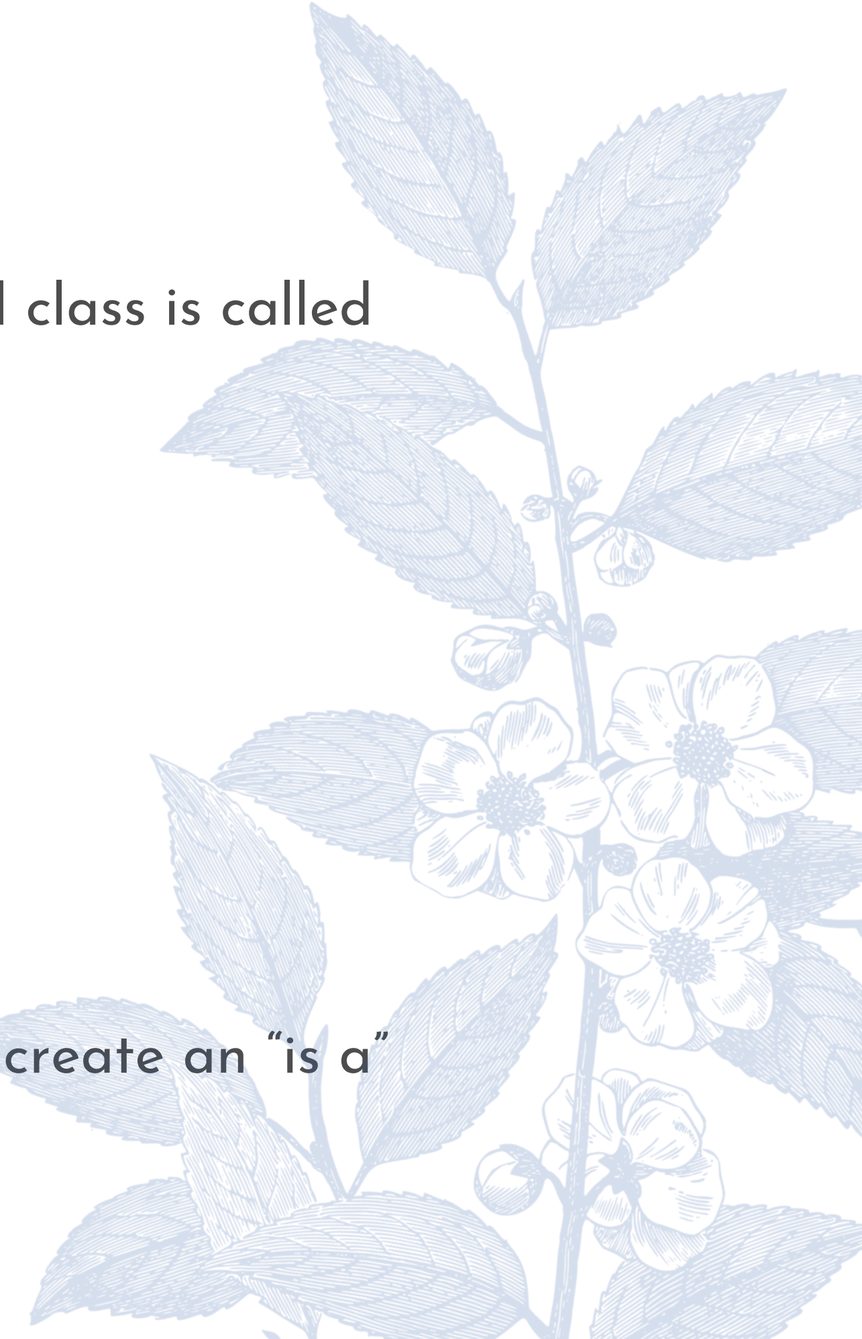
- ✓ Inheritance is a feature of object-oriented programming
- ✓ Inheritance allows a new class to extend an existing class.
  - Called the **subclass**, **child class**, or **derived class**
  - A modified version of an existing class. Called the **superclass**, **parent class**, or **base class**
  - Superclass: more general class
  - Subclass: more specialized class that inherits from the superclass
- ✓ The new class inherits the members *(properties and methods)* of the class it extends.
  - Adding some of its own properties and methods
  - Overriding some of the superclass' methods



What Is Inheritance?

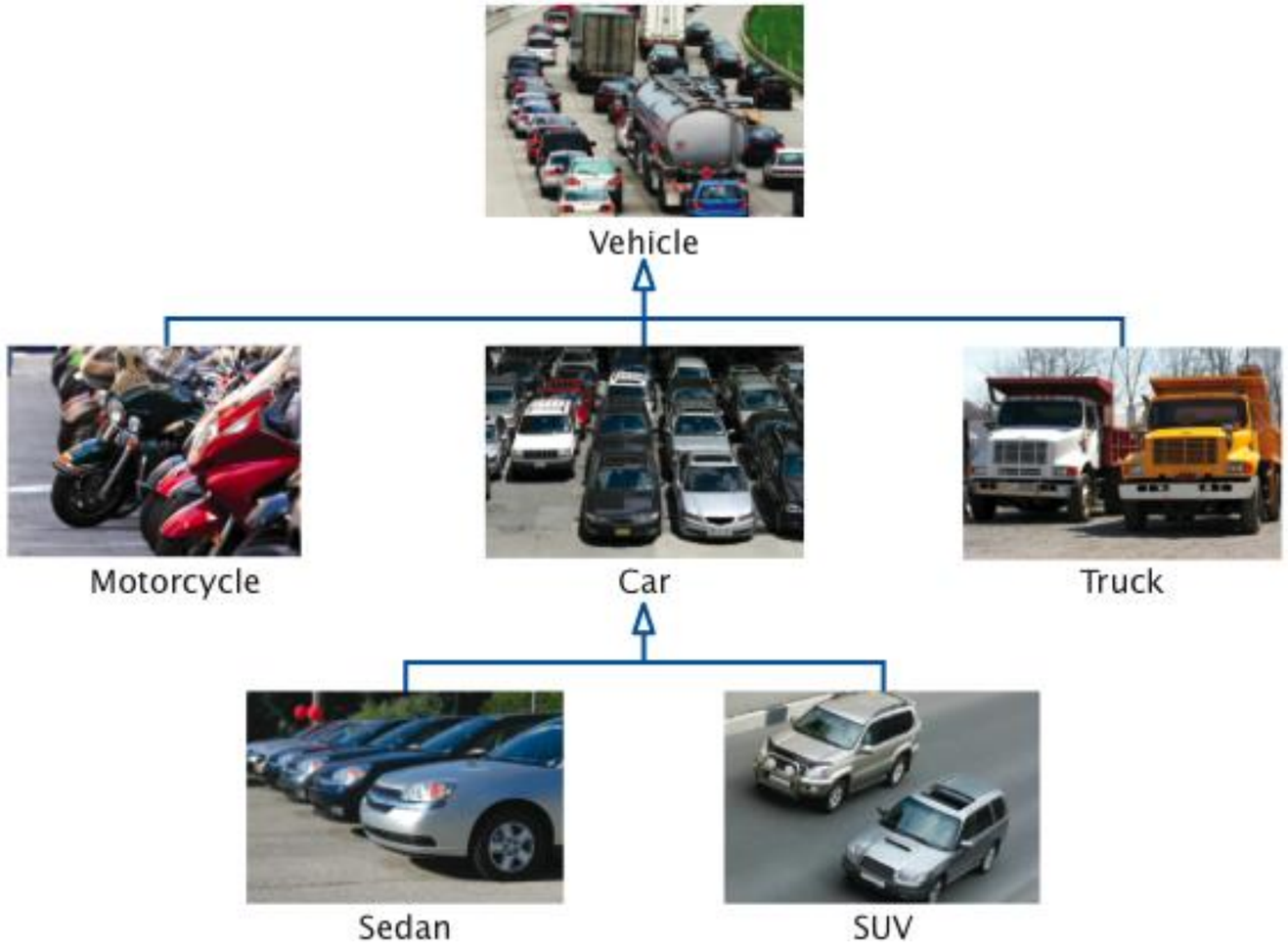
The “is a” Relationship

- ✓ The relationship between a superclass and an inherited class is called an “is a” relationship.
- ✓ Example
  - A grasshopper is a insect.
  - A car is a vehicle.
  - A rectangle is a shape
- ✓ A specialized object has:
  - all of the characteristics of the general object, plus
  - additional characteristics that make it special.
- ✓ In object-oriented programming, inheritance is used to create an “is a” relationship among classes.



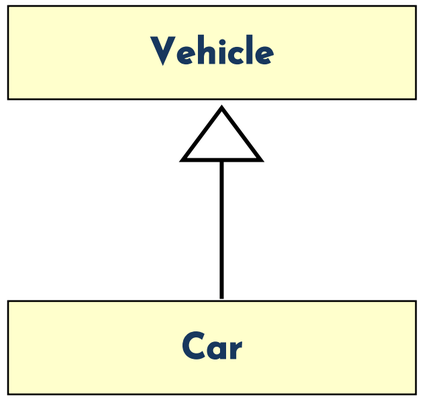
What Is Inheritance?

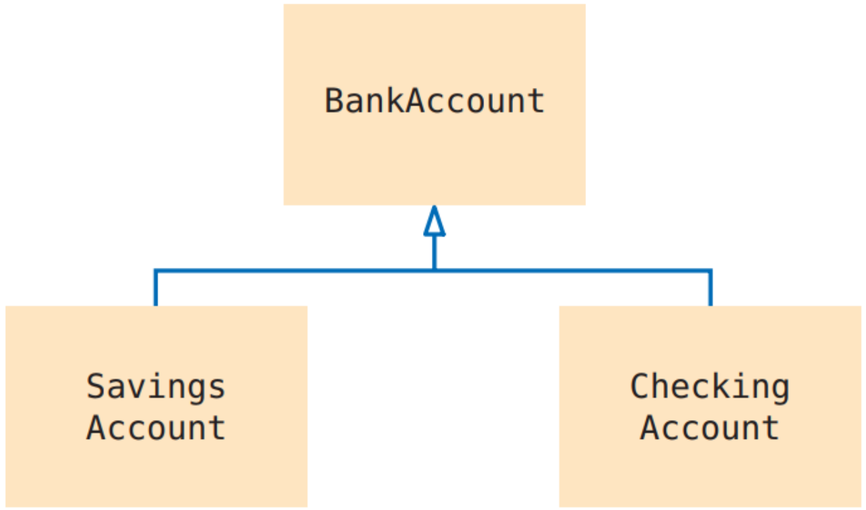
- ✓ A hierarchy of Vehicle types



What Is Inheritance?

- ✓ Inheritance relationships are shown in a UML class diagram using a solid arrow with an unfilled triangular arrowhead pointing to the parent class





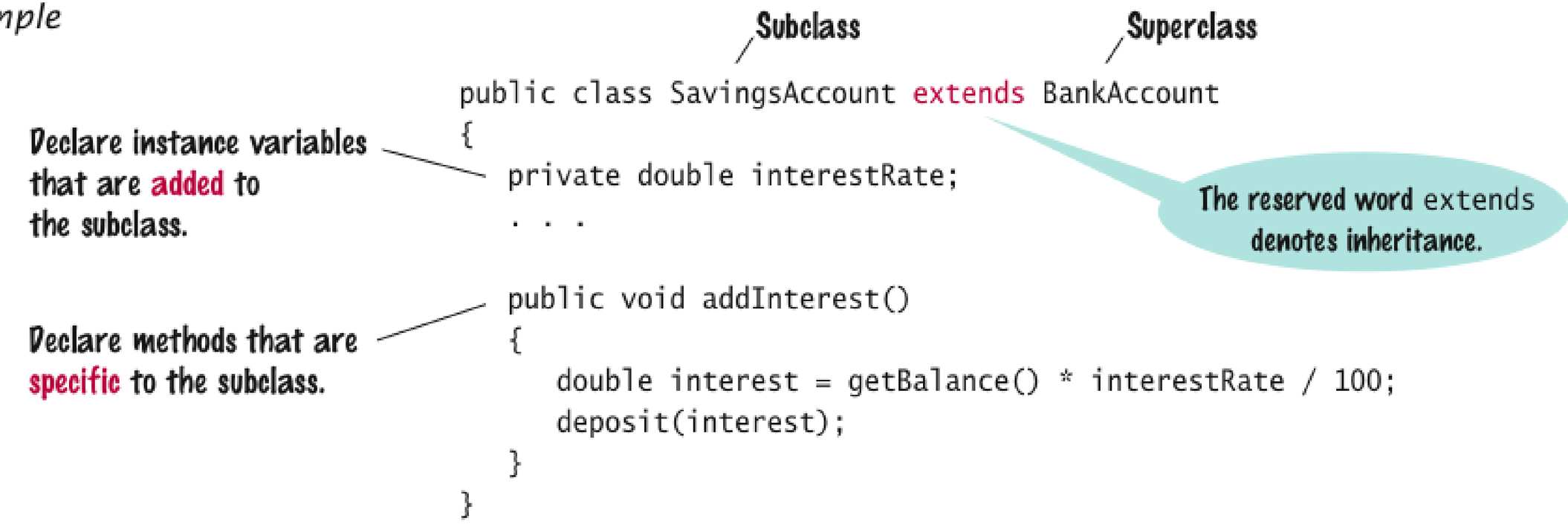
- ✓ A programmer can tailor a derived class as needed by adding new variables or methods, or by modifying the inherited ones
- ✓ One benefit of inheritance is software reuse

What Is Inheritance?

Syntax

```
class SubclassName extends SuperclassName
{
   instance variables
   methods
}
```

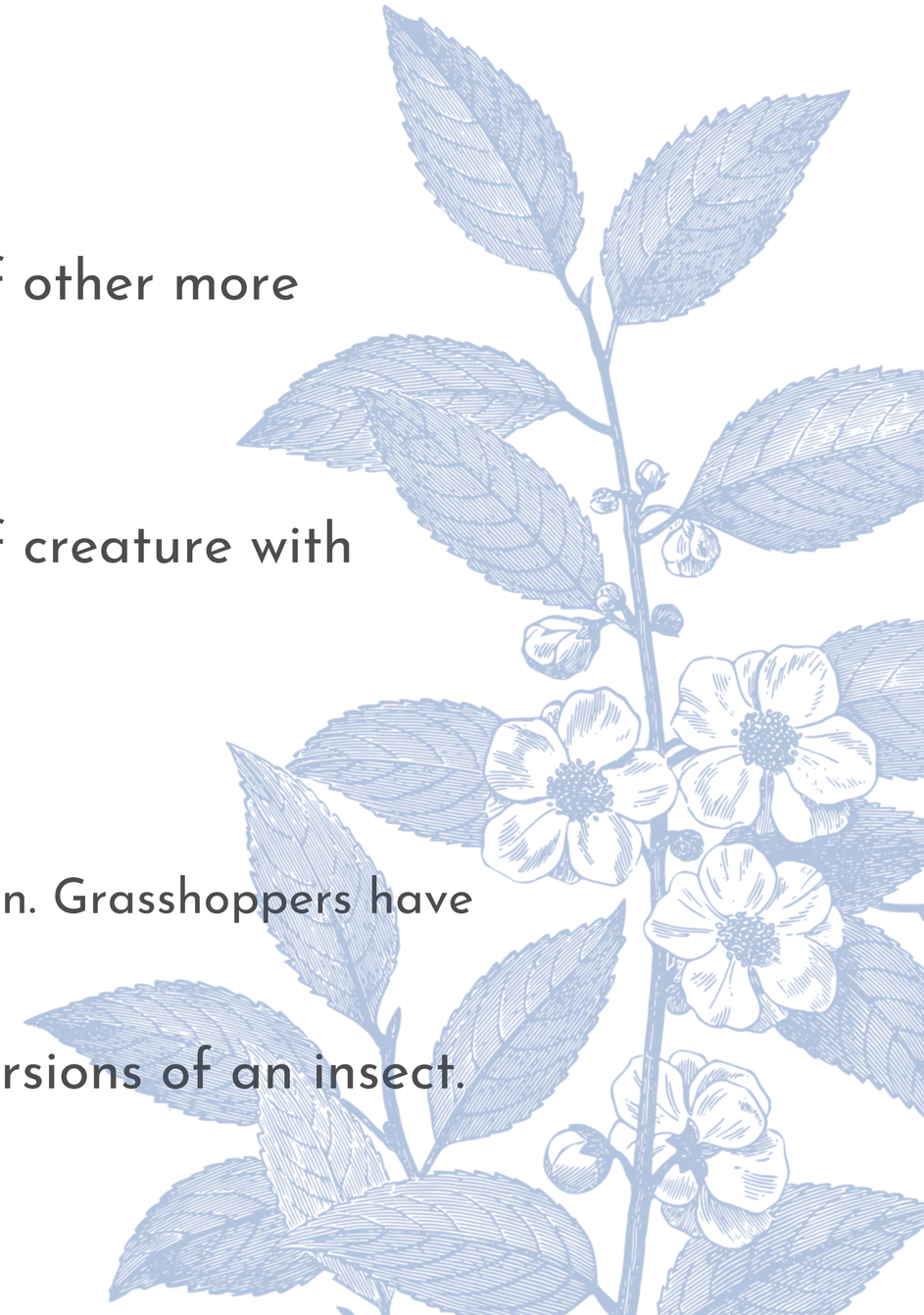
Example



What Is Inheritance?

Generalization vs. Specialization

- ✓ Real-life objects are typically specialized versions of other more general objects.
- ✓ Example
  - The term “insect” describes a very general type of creature with numerous characteristics.
  - Grasshoppers and bumblebees are insects
    - They share the general characteristics of an insect.
    - However, they have special characteristics of their own. Grasshoppers have a jumping ability, and bumblebees have a stinger.
  - Grasshoppers and bumblebees are specialized versions of an insect.

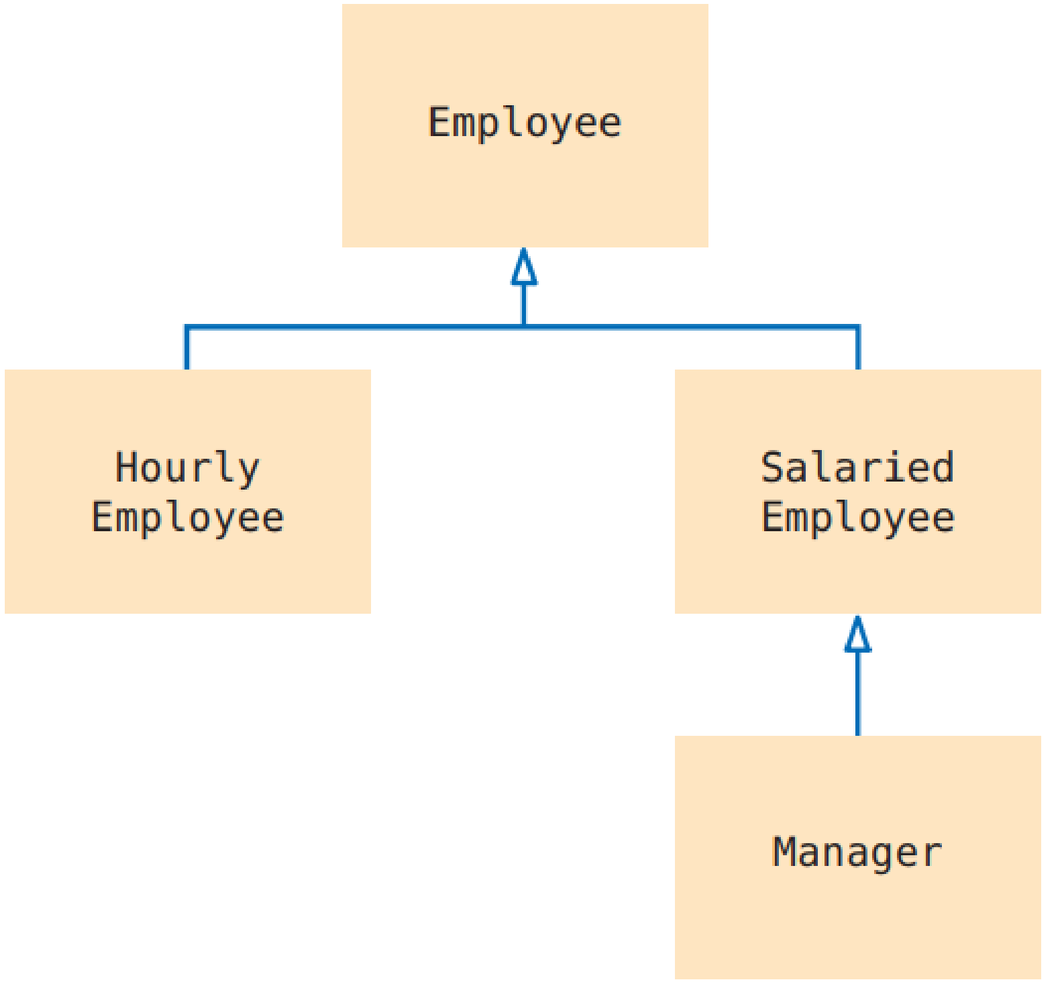


What Is Inheritance?

Demo - WE9-1 *(Big Java Early Objects 7e)*

- ✓Implementing an Employee Hierarchy for Payroll Processing
- ✓Problem Statement
  - Your task is to implement payroll processing for different kinds of employees.
    - —Hourly employees get paid an hourly rate, but if they work more than 40 hours per week, the excess is paid at "time and a half".
    - —Salaried employees get paid their salary, no matter how many hours they work.
    - —Managers are salaried employees who get paid a salary and a bonus.
  - Your program should compute the pay for a collection of employees. For each employee, ask for the number of hours worked in a given week, then display the wages earned.

What Is Inheritance?

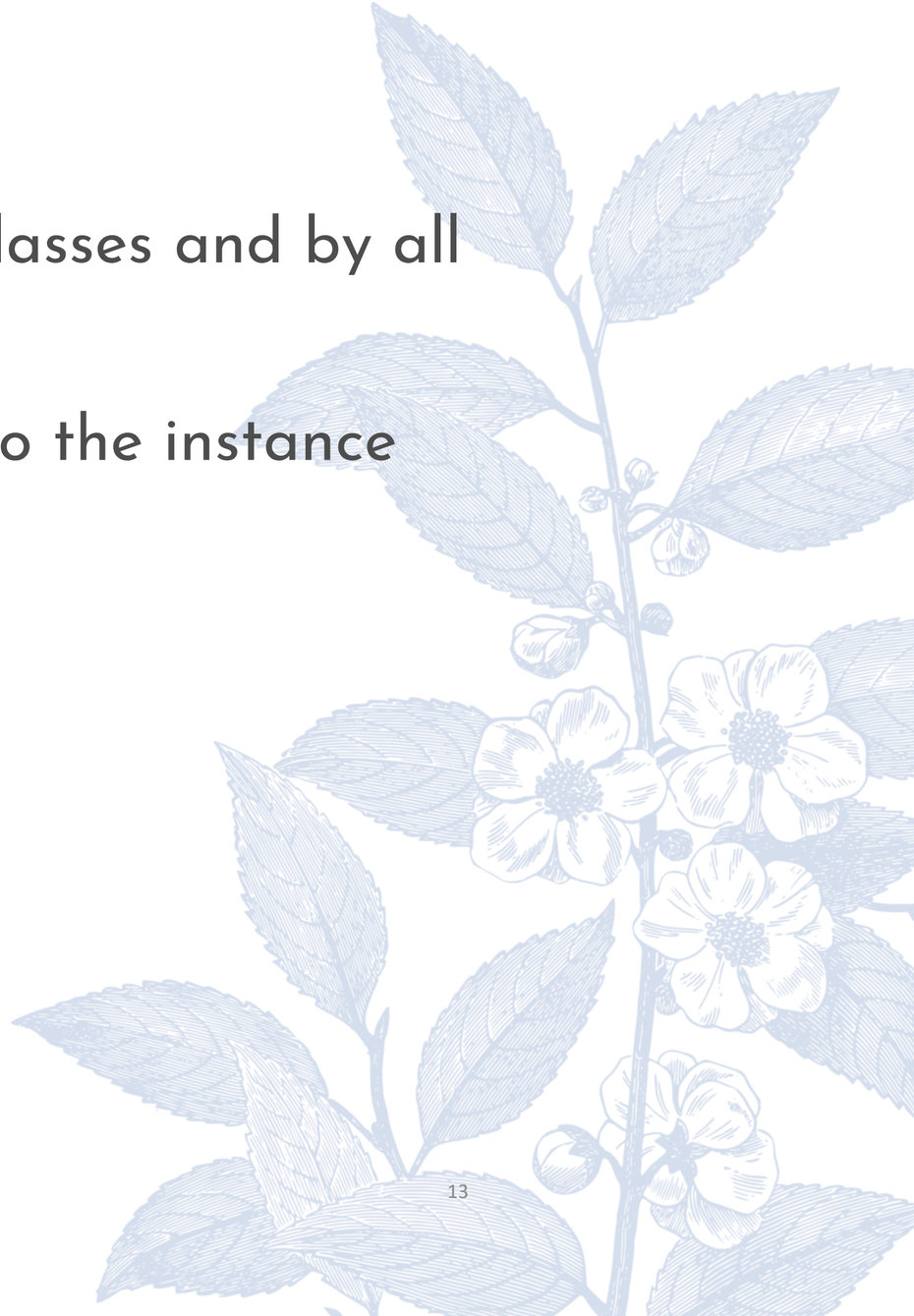


Common Error: Shadowing Instance Variables

- ✓A subclass has no access to the private instance variables of the superclass
- ✓Beginner's error: "solve" this problem by adding another instance variable with same name
  → It doesn't update the correct
- ✓Demo

Protected Access

- ✓Protected features can be accessed by all subclasses and by all classes in the same package
- ✓Solves the problem that methods need access to the instance variable of the superclass



Protected Access

- ✓The designer of the superclass has no control over the authors of subclasses:
  - Any of the subclass methods can corrupt the superclass data
  - Classes with protected instance variables are hard to modify – the protected variables cannot be changed, because someone somewhere out there might have written a subclass whose code depends on them
- ✓Protected data can be accessed by all methods of classes in the same package
- ✓It is best to leave all data private and provide accessor methods for the data

Overriding Methods

- ✓A subclass method overrides a superclass method if it has the same name and parameter types as a superclass method
  - When such a method is applied to a subclass object, the overriding method is executed
- ✓The new method must have the same signature as the parent's method, but can have a different body
- ✓The type of the object executing the method determines which version of the method is invoked
- ✓If you want to modify a private superclass instance variable, you must use a public method of the superclass

Overriding Methods

- ✓Use the super reserved word to call a method of the superclass
- ✓If a method is declared with the final modifier, it cannot be overridden
- ✓The concept of overriding can be applied to data and is called shadowing variables
- ✓Shadowing variables should be avoided because it tends to cause unnecessarily confusing code

Syntax 10.2 Calling a Superclass Method

Syntax `super.methodName(parameters);`

Example

```
public void deposit(double amount)
{
   transactionCount++;
   super.deposit(amount);
}
```

Calls the method of the superclass instead of the method of the current class.

If you omit `super`, this method calls itself.

Subclass Construction

- ✓To call the superclass constructor, use the super reserved word in the first statement of the subclass constructor
- ✓When subclass constructor doesn't call superclass constructor, the superclass must have a constructor with no parameters
  - If, however, all constructors of the superclass require parameters, then the compiler reports an error

Syntax 10.3 Calling a Superclass Constructor

Syntax

```
accessSpecifier ClassName(parameterType parameterName, . . .)
{
   super(parameters);
   . . .
}
```

Example

```
public CheckingAccount(double initialBalance)
{
   super(initialBalance);
   transactionCount = 0;
}
```

Invokes the constructor of the superclass.

Must be the first statement of the subclass constructor.

If not present, the superclass is constructed with its default constructor.

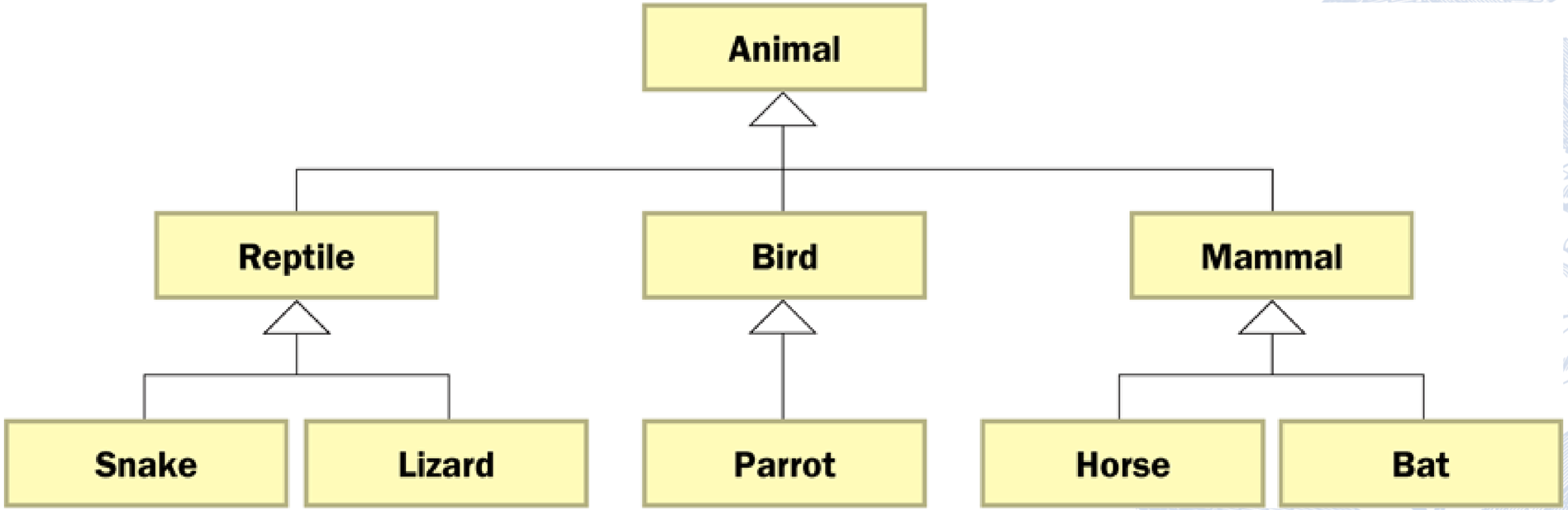
Subclass constructor

Overloading vs. Overriding

- ✓Overloading deals with multiple methods with the same name in the same class, but with different signatures
- ✓Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature
- ✓Overloading lets you define a similar operation in different ways for different parameters
- ✓Overriding lets you define a similar operation in different ways for different object types

Class Hierarchies

- ✓A child class of one parent can be the parent of another child, forming a class hierarchy

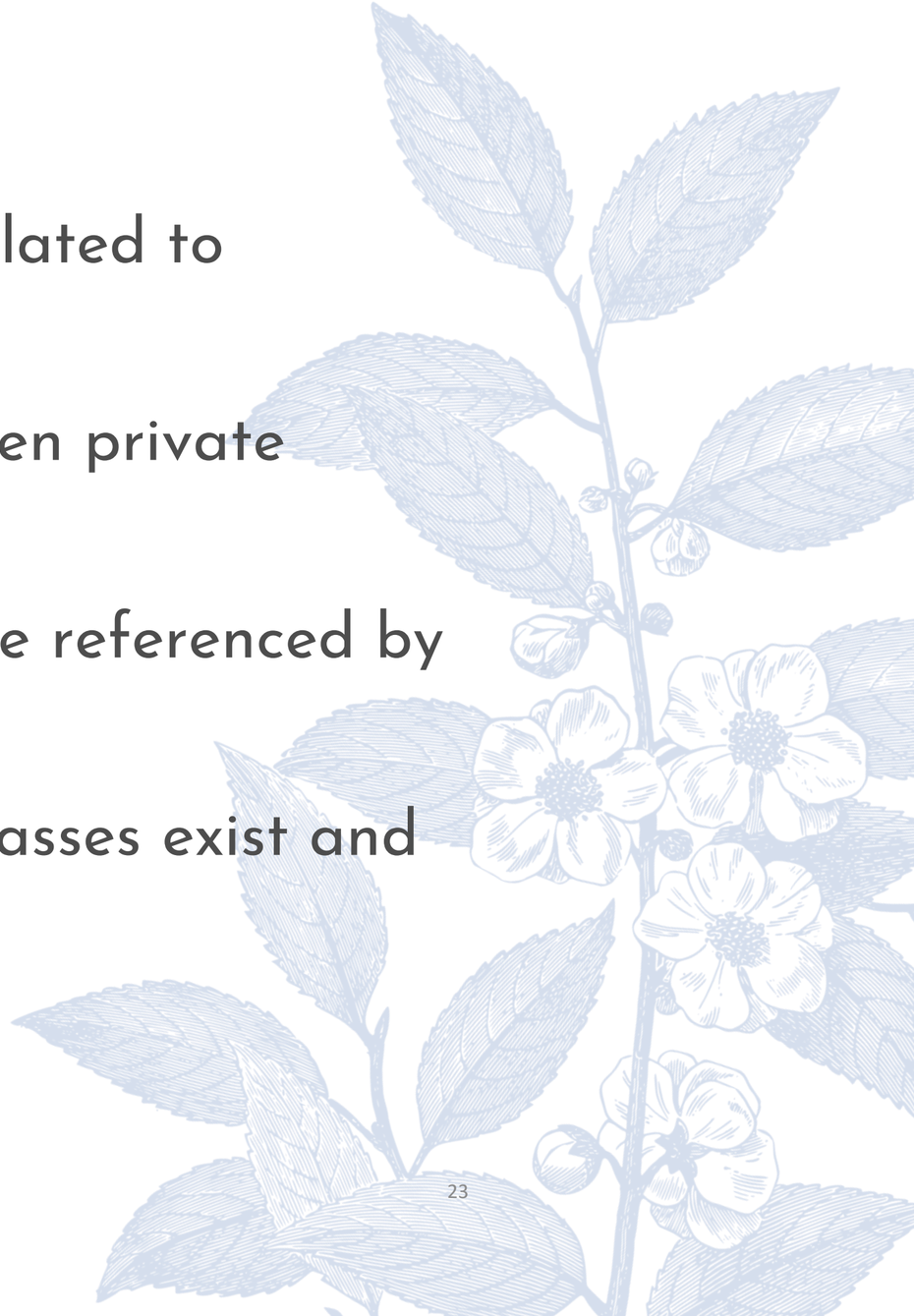


Class Hierarchies

- ✓Two children of the same parent are called siblings
- ✓Common features should be put as high in the hierarchy as is reasonable
- ✓An inherited member is passed continually down the line
- ✓Therefore, a child class inherits from all its ancestor classes
- ✓There is no single class hierarchy that is appropriate for all situations

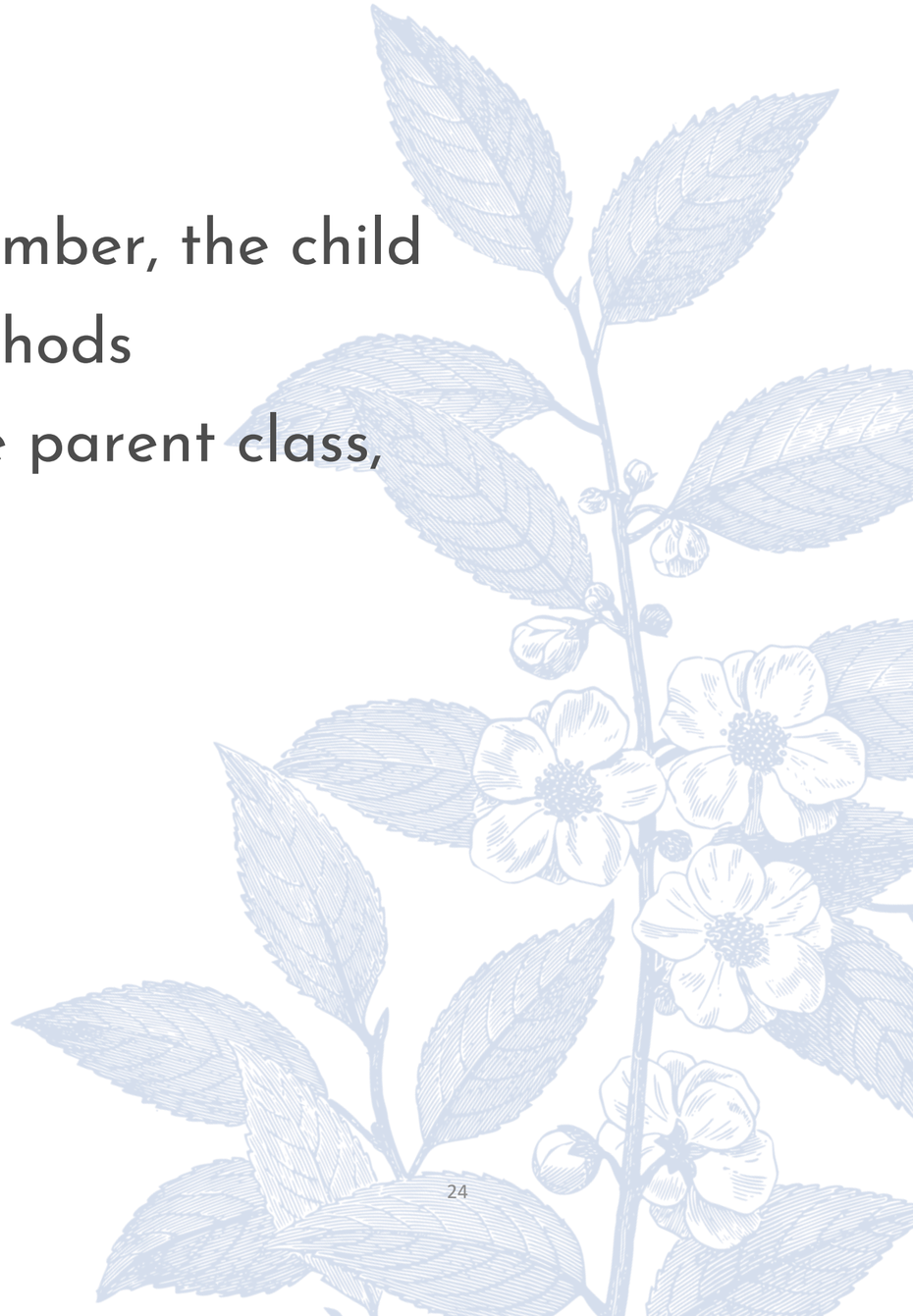
Visibility Revisited

- ✓It's important to understand one subtle issue related to inheritance and visibility
- ✓All variables and methods of a parent class, even private members, are inherited by its children
- ✓As we've mentioned, private members cannot be referenced by name in the child class
- ✓However, private members inherited by child classes exist and can be referenced indirectly



Visibility Revisited

- ✓Because the parent can refer to the private member, the child can reference it indirectly using its parent's methods
- ✓The super reference can be used to refer to the parent class, even if no object of the parent exists

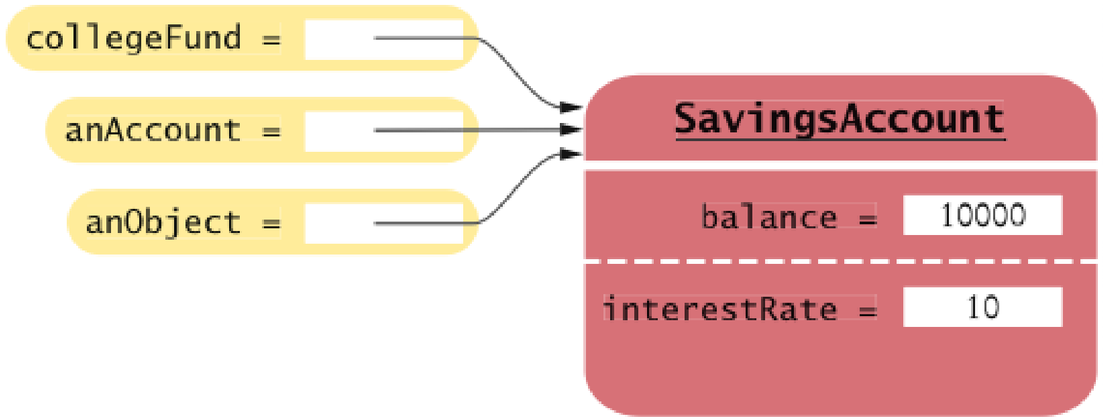


Converting Between Subclass and Superclass Types

- ✓OK to convert subclass reference to superclass reference:

```
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anObject = collegeFund;
```

- ✓The three object references stored in collegeFund, anAccount, and anObject all refer to the same object of type SavingsAccount



Converting Between Subclass and Superclass Types

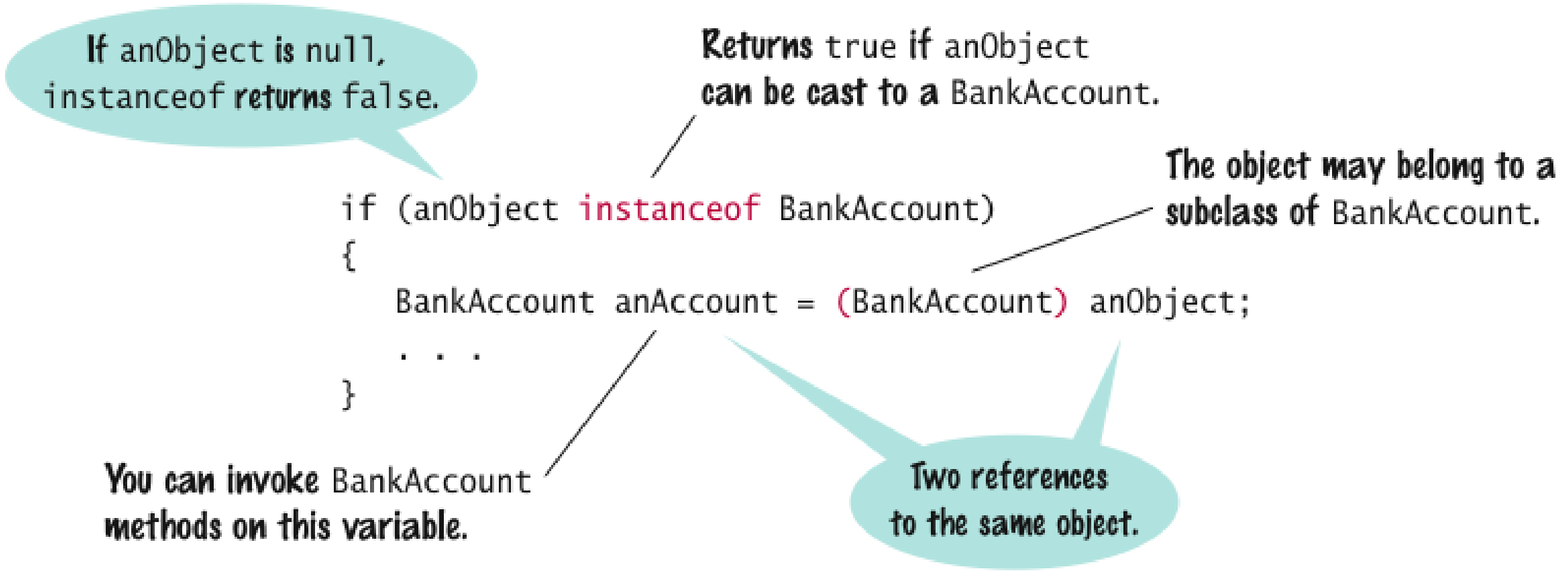
- ✓Superclass references don't know the full story
- ✓Reuse code that knows about the superclass but not the subclass
- ✓Occasionally you need to convert from a superclass reference to a subclass reference
- ✓This cast is dangerous: If you are wrong, an exception is thrown

  Solution: Use the instanceof operator
- ✓ instanceof: Tests whether an object belongs to a particular type

Syntax 10.4 The instanceof Operator

Syntax *object* `instanceof` *TypeName*

Example



The Object Class

- ✓A class called Object is defined in the java.lang package of the Java standard class library
- ✓All classes are derived from the Object class
- ✓If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the Object class
- ✓Therefore, the Object class is the ultimate root of all class hierarchies

The Object Class

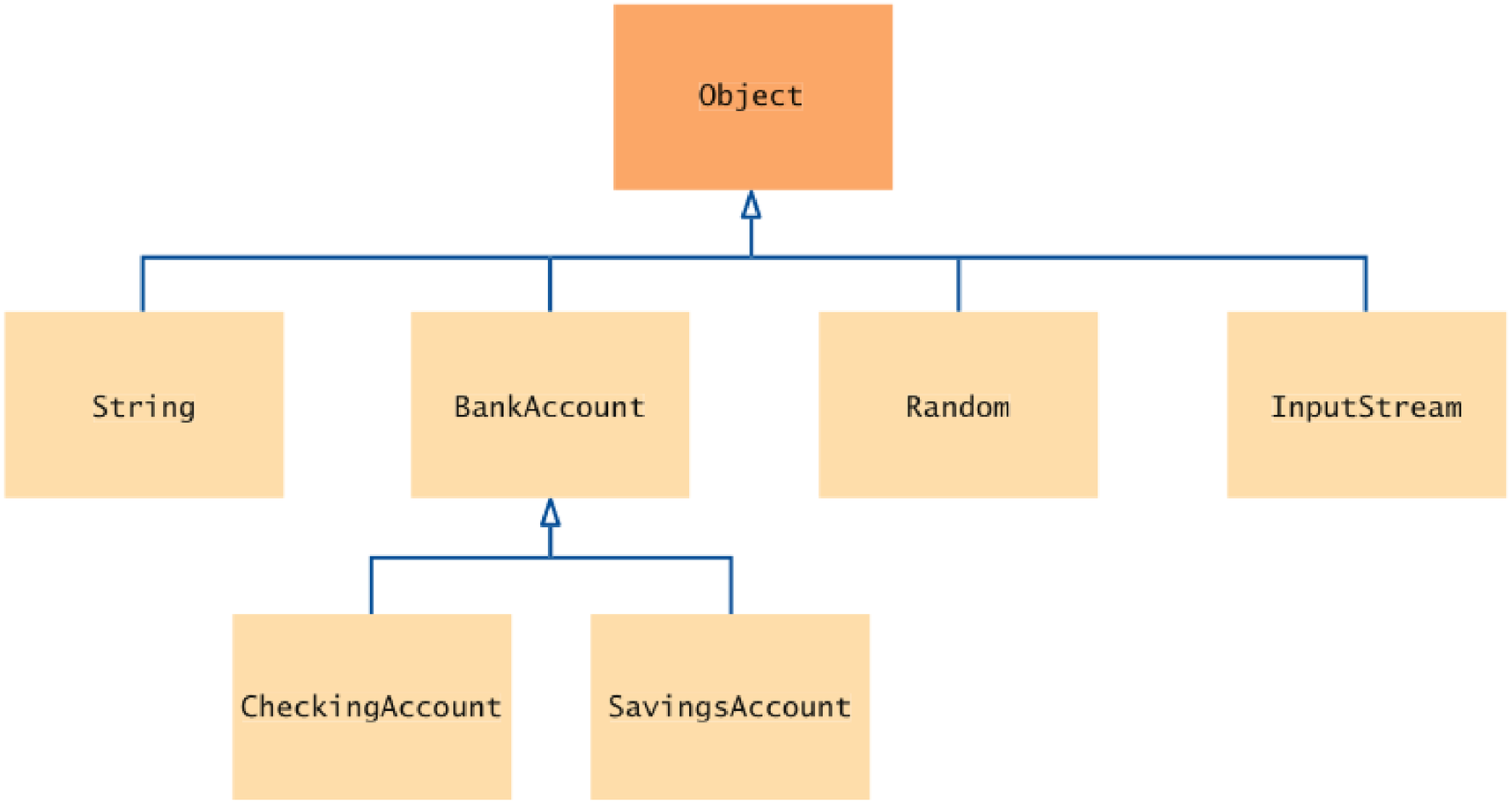


Figure 7 The `Object` Class Is the Superclass of Every Java Class

The Object Class

- ✓The Object class contains a few useful methods, which are inherited by all classes
  - For example, the toString method is defined in the Object class
- ✓Every time we define the toString method, we are actually overriding an inherited definition
- ✓The toString method in the Object class is defined to return a string that contains the name of the object's class along with a hash code

The Object Class

- ✓The equals method of the Object class returns true if two references are aliases
- ✓We can override equals in any class to define equality in some more appropriate way
- ✓As we've seen, the String class defines the equals method to return true if two String objects contain the same characters
- ✓The designers of the String class have overridden the equals method inherited from Object in favor of a more useful version

Overriding the equals Method

- ✓equals tests for same contents:

```
if (coin1.equals(coin2)) . . .
// Contents are the same
```

- ✓== tests for references to the same object:

```
if (coin1 == (coin2)) . . .
// Objects are the same
```

- ✓Need to override the equals method of the Object class
- ✓You should also override the hashCode method so that equal objects have the same hash code

The clone Method

- ✓Copying an object reference gives two references to same object:

```
BankAccount account = new BankAccount(1000);
BankAccount account2 = account;
```

- ✓account2.deposit(500); // Now both account and account2 // refer to a bank account with a balance of 1500
- ✓Sometimes, need to make a copy of the object
- ✓Implement clone method to make a new object with the same state as an existing object
- ✓Must cast return value because return type is Object

Abstract Classes

- ✓An abstract class is a placeholder in a class hierarchy that represents a generic concept
- ✓An abstract class cannot be instantiated
- ✓We use the modifier abstract on the class header to declare a class as abstract

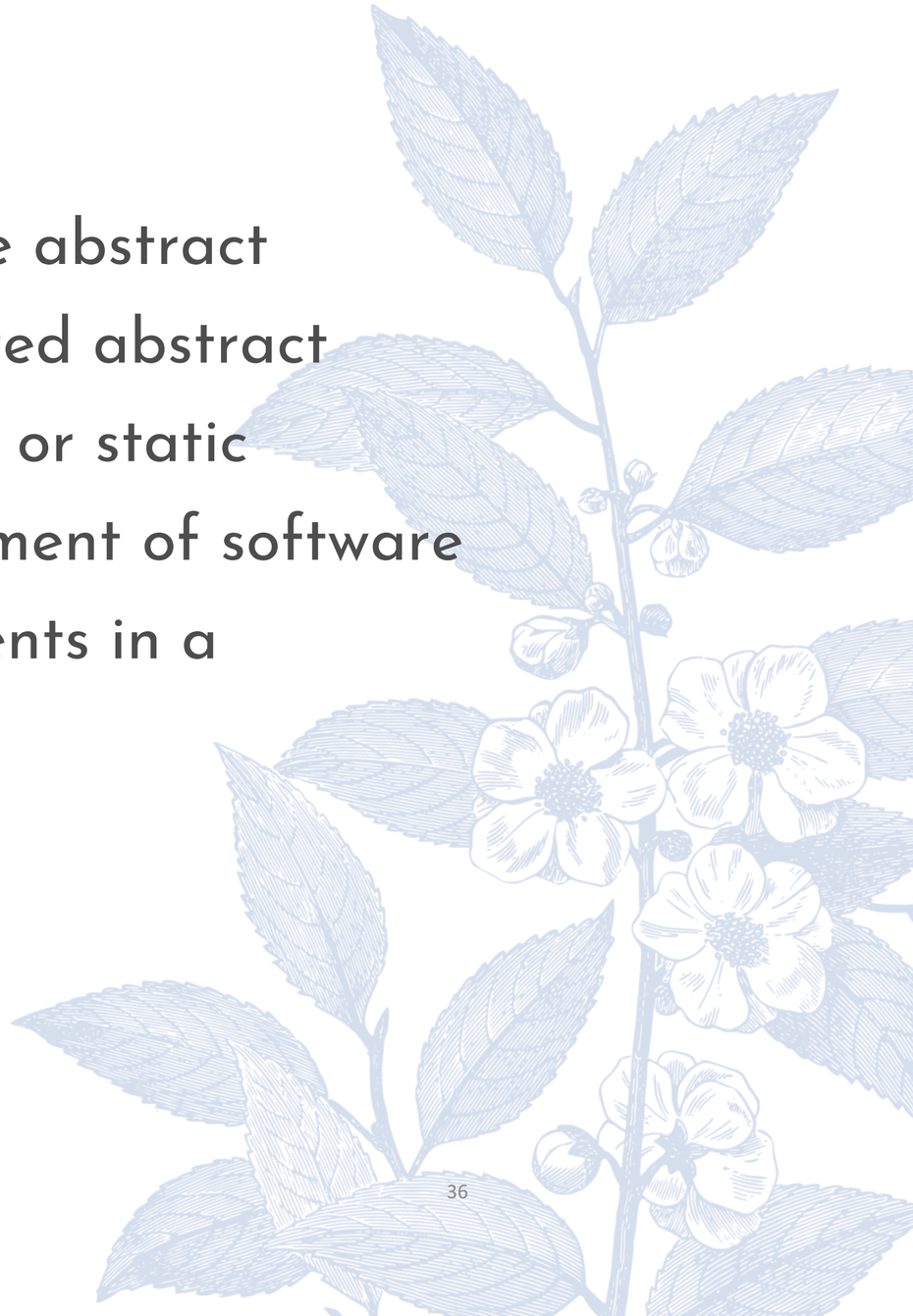
```
public abstract class Product
{
    // class contents
}
```

Abstract Classes

- ✓An abstract class often contains abstract methods with no definitions *(like an interface)*
- ✓Unlike an interface, the abstract modifier must be applied to each abstract method
- ✓Also, an abstract class typically contains non-abstract methods with full definitions
- ✓A class declared as abstract does not have to contain abstract methods -- simply declaring it as abstract makes it so

Abstract Classes

- ✓The child of an abstract class must override the abstract methods of the parent, or it too will be considered abstract
- ✓An abstract method cannot be defined as final or static
- ✓The use of abstract classes is an important element of software design – it allows us to establish common elements in a hierarchy that are too general to instantiate



Interface Hierarchies

- ✓Inheritance can be applied to interfaces
- ✓That is, one interface can be derived from another interface
- ✓The child interface inherits all abstract methods of the parent
- ✓A class implementing the child interface must define all methods from both interfaces
- ✓Class hierarchies and interface hierarchies are distinct (they do not overlap)

Restricting Inheritance

- ✓If the final modifier is applied to a method, that method cannot be overridden in any derived classes
- ✓If the final modifier is applied to an entire class, then that class cannot be used to derive any children at all
- ✓Therefore, an abstract class cannot be declared as final