Java Enterprise Application Development

Lecture 5
Inheritance and Interfaces

Dr. Fan Hongfei

11 October 2025

Inheritance

- Classes can be derived from other classes, inheriting fields and methods
- Definitions
  - **Subclass** (derived class/extended class/child class)
  - **Superclass** (base class/parent class)
- Every class has one and only one direct superclass (**single inheritance**)
  - Excepting *Object*, which has no superclass
- A subclass inherits **all the members** (fields, methods, and nested classes) from its superclass
  - Constructors are not members

What You Can Do in a Subclass

- Use the inherited members as is, replace them, hide them, or supplement them
  - Declare a field in the subclass with the same name as the one in the superclass, thus **hiding** it (NOT recommended)
  - Write a new instance method in the subclass that has the same signature as the one in the superclass, thus **overriding** it
  - Write a new static method in the subclass that has the same signature as the one in the superclass, thus **hiding** it
  - Write a subclass constructor that **invokes** the constructor of the superclass
- How about private members in a superclass?

Casting Objects

- Implicit casting
- Explicit casting
  - A runtime check will be performed
- The ***`instanceof`*** operator

Overriding and Hiding Methods

- Instance methods
  - The same signature and return type
  - An overriding method can also return a subtype: called a **covariant** return type
- Static methods
  - The same signature
- Distinction between hiding a static method and overriding an instance method
  - An example: Animal, Cat
- Modifiers
  - The access specifier for an overriding method **can allow more**, **but not less**, access than the overridden method
- Polymorphism
  - Virtual method invocation

Using the Keyword "super"

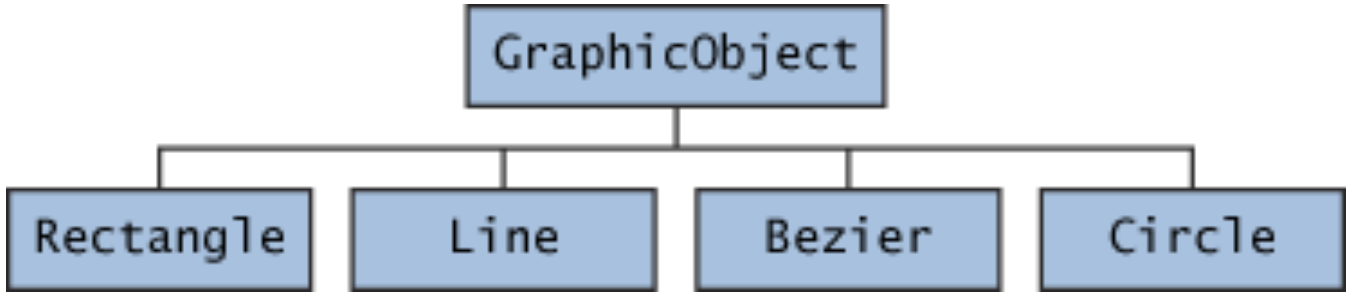
- Accessing superclass members
  - Invoke the overridden method
  - Refer to a hidden field
- Subclass constructors
  - Invocation of a superclass constructor must be the first line in the subclass constructor
  - Constructor chaining

Object as a Superclass

- `protected Object clone() throws CloneNotSupportedException`
- `public boolean equals(Object obj)`
- `protected void finalize() throws Throwable`
- `public String toString()`

Abstract and Final Methods/Classes

- An abstract class is a class declared abstract: it may or may not include abstract methods
- An abstract method is a method declared without an implementation
- Example



- Final methods and classes
  - Methods called from constructors should generally be declared final

Interfaces

- Generally speaking, interfaces are **contracts**
- A reference type, that can contain only constants, method signatures, default methods, static methods, and nested types
- Interfaces cannot be instantiated
  - They can only be implemented by classes or extended by other interfaces
- Interfaces as APIs

Interface Definition

- Consisting of modifiers, keyword, interface name, a comma-separated list of parent interfaces (if any), and the interface body
- Interface body can contain abstract methods, default methods, and static methods
  - Implicitly public, so you can omit the public modifier
- An interface can contain constant declarations
  - Implicitly public, static, and final

Implementing and Using Interfaces

- Include an *implements* clause in the class declaration
  - Your class can implement more than one interface
- Example: *Relatable*
- If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface

Evolving Interfaces

- You can create an interface that extends an existing interface
- Default methods
  - Enabling you to add new functionality and ensure binary compatibility with code written for older versions of those interfaces
  - When you extend an interface that contains a default method, you can
    - Not mention the default method at all
    - Redeclare the default method, which makes it abstract
    - Redefine the default method, which overrides it
- Static methods

Abstract Classes vs. Interfaces

- Similarities and differences
- Consider using abstract classes when
  - You want to **share code** among several closely related classes
  - You expect that classes extending the abstract class have **many common methods or fields**, or require **access modifiers other than public**
  - You want to declare **non-static or non-final** fields
- Consider using interfaces when
  - You expect that **unrelated classes** would implement your interface
  - You want to specify the **behavior** of a particular data type, but not concerned about who implements its behavior
  - You want to take advantage of **multiple inheritance**

Anonymous Classes

- Anonymous classes make code more concise
  - Declare and instantiate a class at the same time
  - They do not have a name: use them if you need to use a local class only once
- The anonymous class expression consists of
  - The new operator
  - The **name of an interface/class to implement/extend**
  - Parentheses that contain the arguments to a constructor
  - The class declaration body
    - You cannot declare constructors in an anonymous class

Programming Practice

- Design and implement a set of classes representing various shapes, and organize them in an **appropriate** way
  - Shape (abstract), which implements the *Relatable* interface that contains an *int compare(Relatable)* method
  - Circle (field: radius)
  - Ellipse (fields: semi-major axis, semi-minor axis)
  - Rectangle (fields: width, height)
  - Triangle (fields: altitude, length of base)
  - Square (field: length of edge)
- Implement the following functionalities **properly** for each class
  - User-friendly constructor(s)
  - getArea()
  - print(): printing shape type, particular(s) and area
- In the main method
  - Use one array to store all types of shapes, and invoke the implemented functionalities in a loop
  - Test the *compare* methods of the shapes