Lecture 7

Inheritance

References:

1. Tony Gaddis, Chapter 10, Starting out with Java: From Control Structures through Objects, 7 edition
2. Herbert Schildt, Chapter 8, The Complete Reference Java 10 edition, McGraw Hill

1

Chapter Topics

- What Is Inheritance?
- Calling the Superclass Constructor
- Overriding Superclass Methods
- Protected Members
- Chains of Inheritance
- The `Object` Class
- Polymorphism and dynamic method dispatch
- Abstract Classes
- Type Wrapper
- Autoboxing

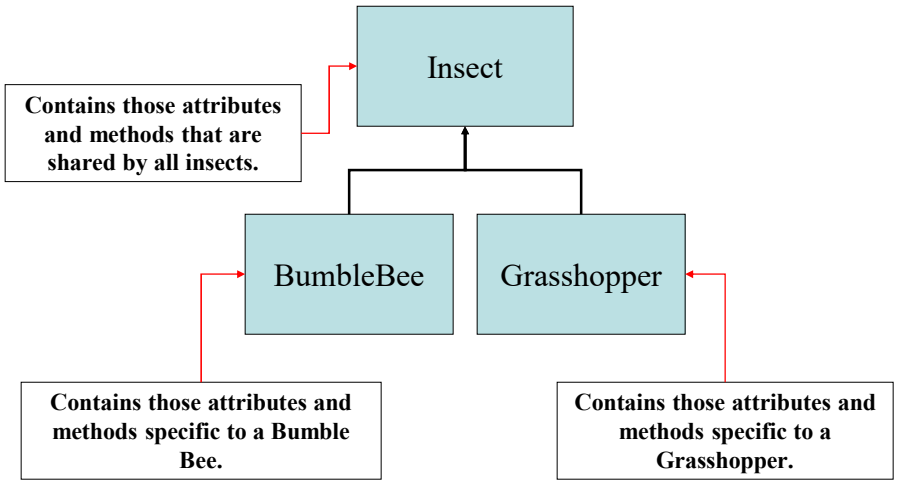
2

What is Inheritance? Generalization vs. Specialization

- Real-life objects
  - Specialized versions of general objects
- The term “insect”
  - A very general type of creature with numerous characteristics
- Grasshoppers and bumblebees are insects
  - Share the general characteristics of an insect
  - However, have special characteristics of their own
    - grasshoppers with a jumping ability
    - bumblebees with a stinger
  - Specialized versions of an insect

3

Inheritance



4

The "is a" Relationship (1 of 2)

- The relationship between a superclass and an inherited class
  - A grasshopper "is a" insect.
  - A poodle "is a" dog.
  - A car "is a" vehicle.
- A specialized object
  - All of the characteristics of the general object
  - Plus additional characteristics
- Inheritance in OOP
  - To crate an "is a" relationship among classes

5

The "is a" Relationship (2 of 2)

- Inheritance
  - The *superclass (base or parent classes):* the general class
  - the *subclass (derived or child classes):* the specialized class
- *Can extend* the capabilities of a class using inheritance
  - The subclass is based on, or extended from, the superclass

6

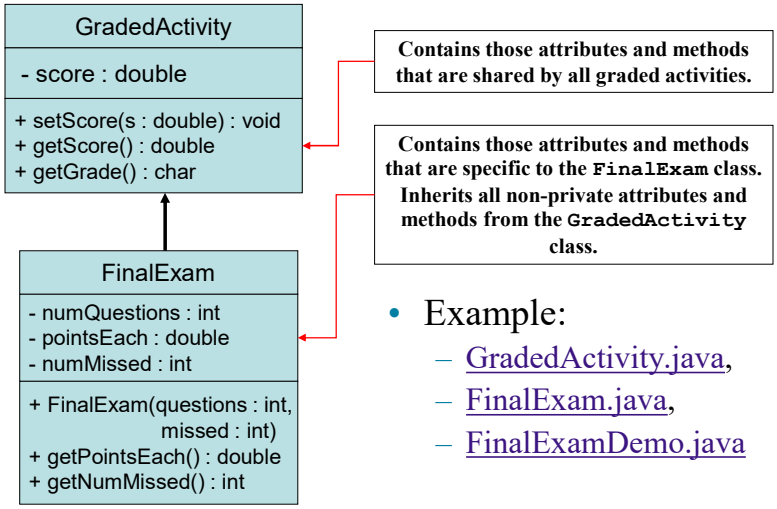
Inheritance

- The subclass inherits fields and methods from the superclass
- Reuse superclass code in subclasses
- New fields and methods may be added to the subclass
- The Java keyword, *extends*,
  - On the class header to define the subclass

```
public class FinalExam extends GradedActivity
```

7

The `GradedActivity` Example



- Example:
  - GradedActivity.java,
  - FinalExam.java,
  - FinalExamDemo.java

8

Inheritance, Fields and Methods (1 of 2)

- Superclass's members marked *private*:
  - Not inherited by the subclass
  - Exist in memory when the object of the subclass is created
  - Accessed from the subclass by public methods of the superclass
- Superclass's members marked *public*:
  - Inherited by the subclass
  - Directly accessed from the subclass

9

Inheritance, Fields and Methods (2 of 2)

- Non-private methods of the superclass available through the subclass object:

```
FinalExam exam = new FinalExam();
exam.setScore(85.0);
System.out.println("Score = "
                   + exam.getScore());
```

- Non-private methods and fields of the superclass available within the subclass:

```
setScore(newScore);
```

10

Inheritance and Constructors

- Constructors are not inherited
- When a subclass is instantiated, the superclass default constructor is executed first
- Example:
  - SuperClass1.java
  - SubClass1.java
  - ConstructorDemo1.java

11

The Superclass's Constructor

- The `super` keyword referring to an object's superclass
- The superclass constructor explicitly called from the subclass by using the `super` keyword
- Example:
  - SuperClass2.java, SubClass2.java, ConstructorDemo2.java
  - Rectangle.java, Cube.java, CubeDemo.java

12

Calling The Superclass Constructor

- If a parameterized constructor is defined in the superclass,
  - The superclass must provide a no-arg constructor, or
  - Subclasses must provide a constructor and call a superclass constructor
- Calls to a superclass constructor
  - Must be the first java statement in the subclass constructors

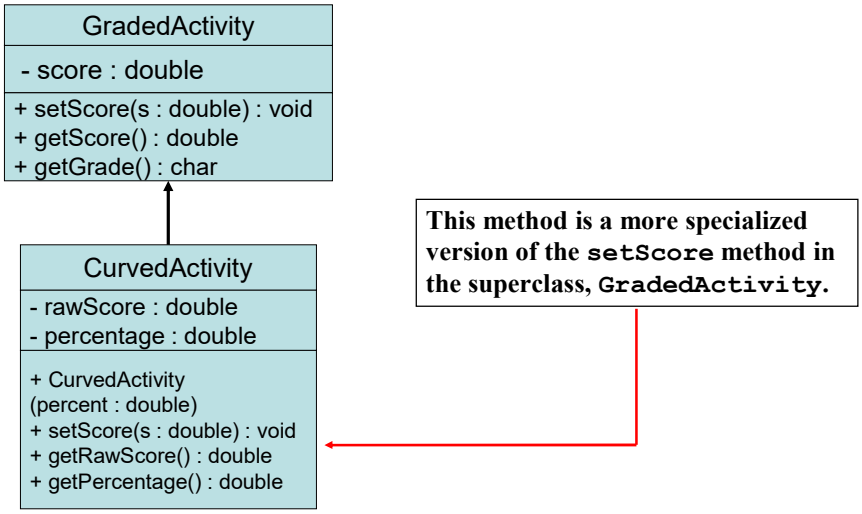
13

Overriding Superclass Methods (1 of 5)

- Method overriding
  - A subclass's method with the same signature as a superclass method
  - The subclass method overrides the superclass method
- Example:
  - GradedActivity.java, CurvedActivity.java, CurvedActivityDemo.java

14

Overriding Superclass Methods (2 of 5)



15

Overriding Superclass Methods (3 of 5)

- Recall that a method's *signature* consists of:
  - The method's name
  - The method's parameter types
  - the appearing order
- Overriding subclass method
  - Must have the same signature as the superclass method
  - Invokes the subclass's version of the method, not the superclass's
- The `@Override` annotation
  - Used just before the subclass method declaration
  - Display an error message if the method overring fails

16

Overriding Superclass Methods (4 of 5)

- Using super
- Two general forms of super
  - Call the superclass' constructor
    - super (arg-list);
  - Access a member of the superclass hidden by a member of a subclass
    - super.member
    - E.g., UseSuper program

17

Overriding Superclass Methods (5 of 5)

- Overriding
  - Take place in an inheritance relationship
  - Superclass and subclass have the same signature
- Overloading
  - Can take place within a class
  - Has the same method, but different parameters
- Example:
  - SuperClass3.java,
  - SubClass3.java,
  - ShowValueDemo.java

18

Preventing a Method from Being Overridden

- The `final` modifier
  - Prevent the overriding of a superclass method in a subclass

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

- To ensure that a particular superclass method used by subclasses
  - Rather than a modified version of it

19

Protected Members (1 of 2)

- `protected` members of class
  - Accessed by methods in a subclass, and
  - Accessed by methods in the same package as the class
- *`Protected`* member's access
  - Somewhere between *private* and *public*
- Example:
  - GradedActivity2.java
  - FinalExam2.java
  - ProtectedDemo.java

20

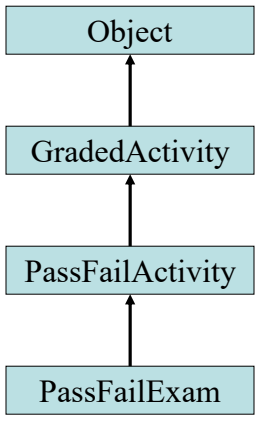
Protected Members (2 of 2)

- `private` field `and public` method
  - Better to make all fields `private` and then provide `public` methods for accessing those fields
- No access specifier for a class member
  - Given *package access* by default (public by default)
  - Any method in the same package may access the member

21

Chains of Inheritance (1 of 2)

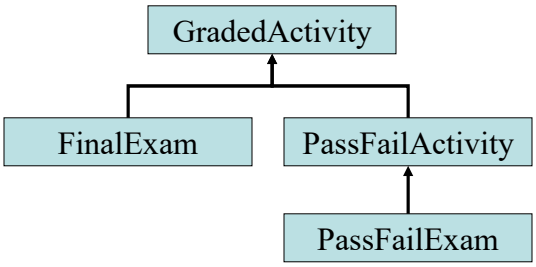
- A superclass derived from another class



22

Chains of Inheritance (2 of 2)

- A class hierarchy shows the inheritance relationships between classes



23

Object Class (1/2)

- One special class defined by Java
- All other classes are subclasses of Object
  - A superclass of all other classes
  - A reference variable of Object class can reference an object of any other class

24

The Object Class (2/2)

- Methods
  - protected Object clone(); require implementing the "Cloneable" interface
  - public boolean equals(Object object)
  - protected void finalize() //deprecated
  - Class<?> getClass() – obtains the class of an object at run time
  - public int hashCode() – returns the hash code associated with the invoking object
  - public String toString() – returns a string that describes the object
  - public final void notify(), public final void notifyAll()
  - public final void wait(), public final void wait(long milliseconds), public final void wait(long milliseconds, int nanoseconds)

25

25

Polymorphism

- Meaning "many forms" from Greek
- Same method name, but different implementations
  - Within a class - overloading
  - On class inheritance - overriding
  - Using interface/abstract class – different implementation

26

Dynamic Method Dispatch (binding) (1/5)

- A superclass reference variable can reference objects of subclasses

```
GradedActivity exam;
```

- The exam variable referencing a `GradedActivity` object.

```
exam = new GradedActivity();
```

- The `GradedActivity` class as the superclass for the `FinalExam` class

```
GradedActivity exam = new FinalExam(50, 7);
```

27

Dynamic Method Dispatch (binding) (2/5)

- Other legal polymorphic references:

```
GradedActivity exam1 = new FinalExam(50, 7);
GradedActivity exam2 = new PassFailActivity(70);
GradedActivity exam3 = new PassFailExam(100, 10, 70);
```

- The `GradedActivity` class
  - Three methods: `setScore`, `getScore`, and `getGrade`
- A `GradedActivity` variable used to call only those three methods

```
GradedActivity exam = new PassFailExam(100, 10, 70);
System.out.println(exam.getScore()); // This works.
System.out.println(exam.getGrade()); // This works.
System.out.println(exam.getPointsEach()); // ERROR!
```

28

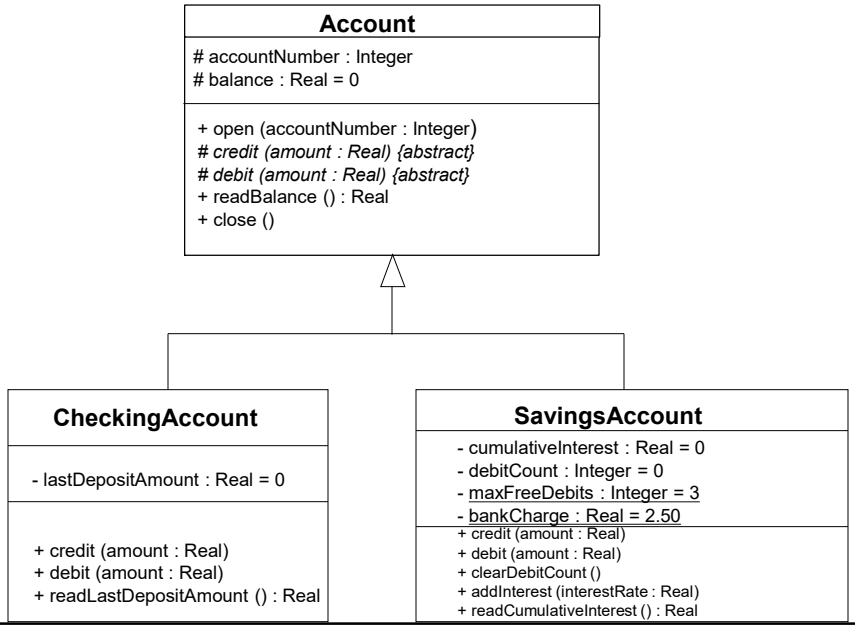
Dynamic Method Dispatch (binding) (3/5)

- Dynamic Method Dispatch
  - Determining which version of a (overriding) method executes
- Overriding methods
  - Determined by the type of the object being referenced
- Non-overriding methods
  - Determined by the type of the object reference variable
- E.g., Dispatch program

29

Dynamic Method Dispatch (binding) (4/5)

superclass and subclass



30

Dynamic Method Dispatch (5/5)

- Variable may reference objects of different classes at different times, e.g.:

```
Prompt customer for account type and withdrawal amount
if customer responds checking
   then anAccount := customerCheckingAccount
   elseif customer responds savings
       Then anAccount := customerSavingsAccount
       …
endif….
anAccount.debit (amount)
```


31

Abstract Classes

- An *Abstract class* serves as a superclass for other classes
  - Represents the generic or abstract form of all the classes
  - Other classes are derived from it
  - Cannot be instantiated
- A class becomes abstract with the abstract key word in the class definition

```
public abstract class ClassName
```

32

Abstract Methods (1 of 2)

- An abstract method
  - has only a header and no body
  - the key word `abstract` appears in the header

```
AccessSpecifier abstract ReturnType
  MethodName(ParameterList);
```

- An abstract method
  - Must be overridden in a subclass
- Example:
  - Student.java, CompSciStudent.java, CompSciStudentDemo.java

33

Abstract Methods (2 of 2)

- Abstract class
  - Contains one or more abstract methods
- Why abstract method used?
  - To ensure that a subclass implements the method

34

Type Wrappers

- Primitive types
  - Not a class for the sake of performance
  - Are not part of the object hierarchy
  - Do not inherit Object
  - Need to be object representation
    - Cannot **pass** a primitive type by reference to a method
- Many Java standard data structures (e.g., queue, stack, list …)
  - Operate on objects

Type Wrappers

- Type wrapper
  - Encapsulate a primitive type within an object
  - Double, Float, Long, Integer, Short, Byte, Character, Boolean
    - Type of wrappers
- Character

  Character (char ch)

  char charValue()

Type Wrappers

- Boolean
  - Boolean (boolean boolValue)
  - Boolean (String boolString)
  - boolean booleanValue()
- Numeric Type Wrappers
  - Integer (int num)
  - Integer (String str)
  - int intValue()
  - …..
- All of the type wrappers override toString()
- E.g., Wrap program

Autoboxing

- Boxing/Unboxing
  - The process of encapsulating a value within an object
- Autoboxing and auto-unboxing
  - Automatically encapsulate a primitive type into its equivalent type wrapper if necessary
  - No need to manually construct an object
  - E.g., AutoBox program

Autoboxing

- Autoboxing/Unboxing occurs
  - when an argument is passed to a method and
  - when a value is returned by a method
- Autoboxing/Unboxing occurs in expressions
- Autoboxing/Unboxing Boolean and Character values
  - E.g., AutoBox5 program