

ICT104

Program Design and Development

Lecture 5– Inheritance

Adopted from: Gaddis & Gaddis (2019) Starting Out with Java: From Control Structures through Objects, 7th Edition.

Focus for this week

Inheritance

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

Activity 1:

Revision Exercise

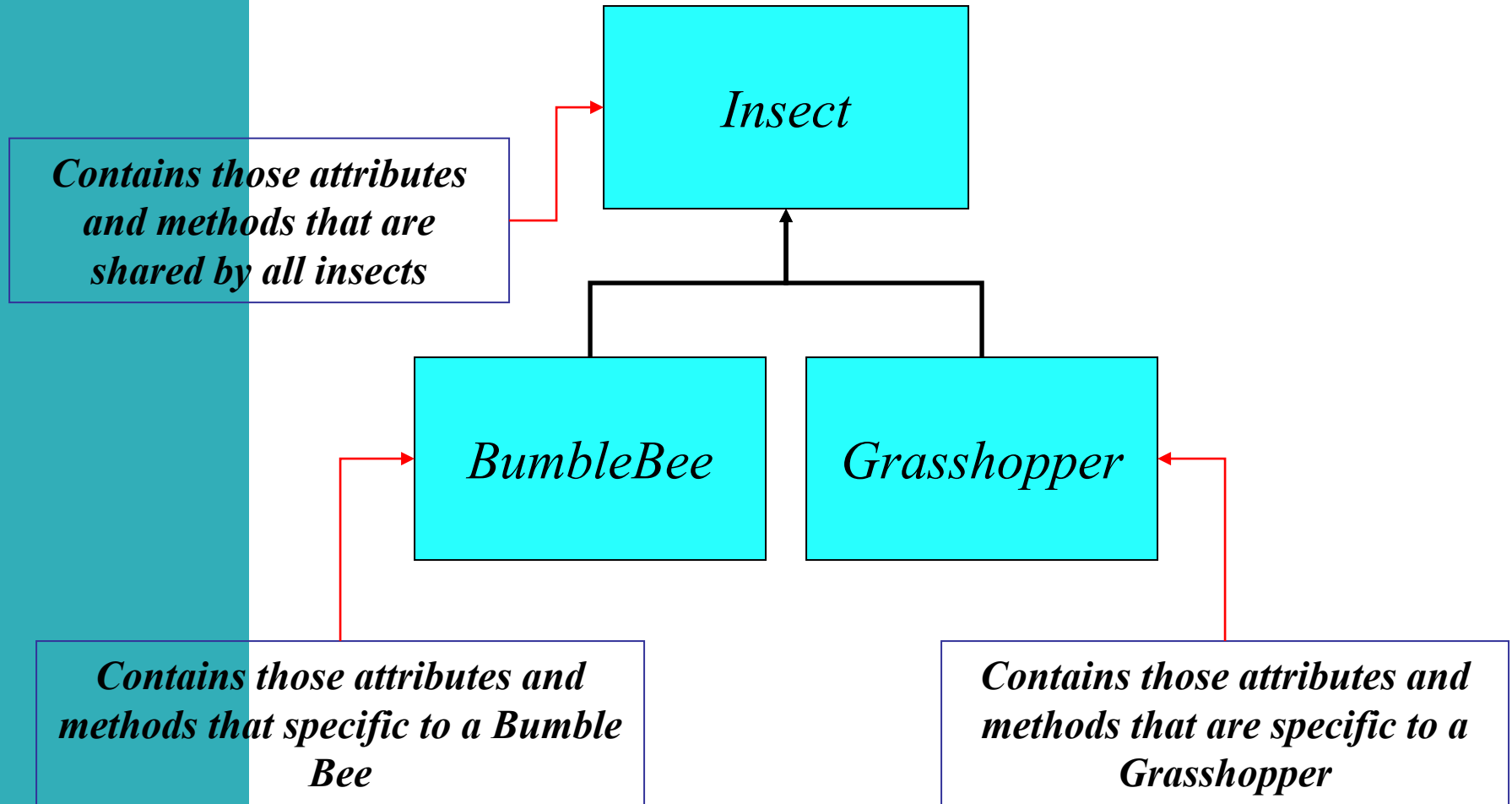
List any three concepts which you can remember from your previous week class

What is Inheritance?

Generalization vs. Specialization

- Real-life objects are typically specialized versions of other more general objects
- 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

Inheritance



The “is a” Relationship

- The relationship between a superclass and an inherited class is called an “is a” relationship
 - A grasshopper “is a” insect
 - A poodle “is a” dog
 - A car “is a” vehicle
- 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

The “is a” Relationship

- We can *extend* the capabilities of a class
- Inheritance involves a superclass and a subclass
 - The *superclass* is the general class
 - The *subclass* is the specialized class
- The subclass is based on, or extended from, the superclass
 - Superclasses are also called *base classes*
 - Subclasses are also called *derived classes*
- The relationship of classes can be thought of as *parent classes* and *child classes*

Inheritance

- The subclass inherits fields and methods from the superclass without any of them being rewritten
- New fields and methods may be added to the subclass
- The Java keyword, ***extends***, is used on the class header to define the subclass

```
public class FinalExam extends  
    GradedActivity
```

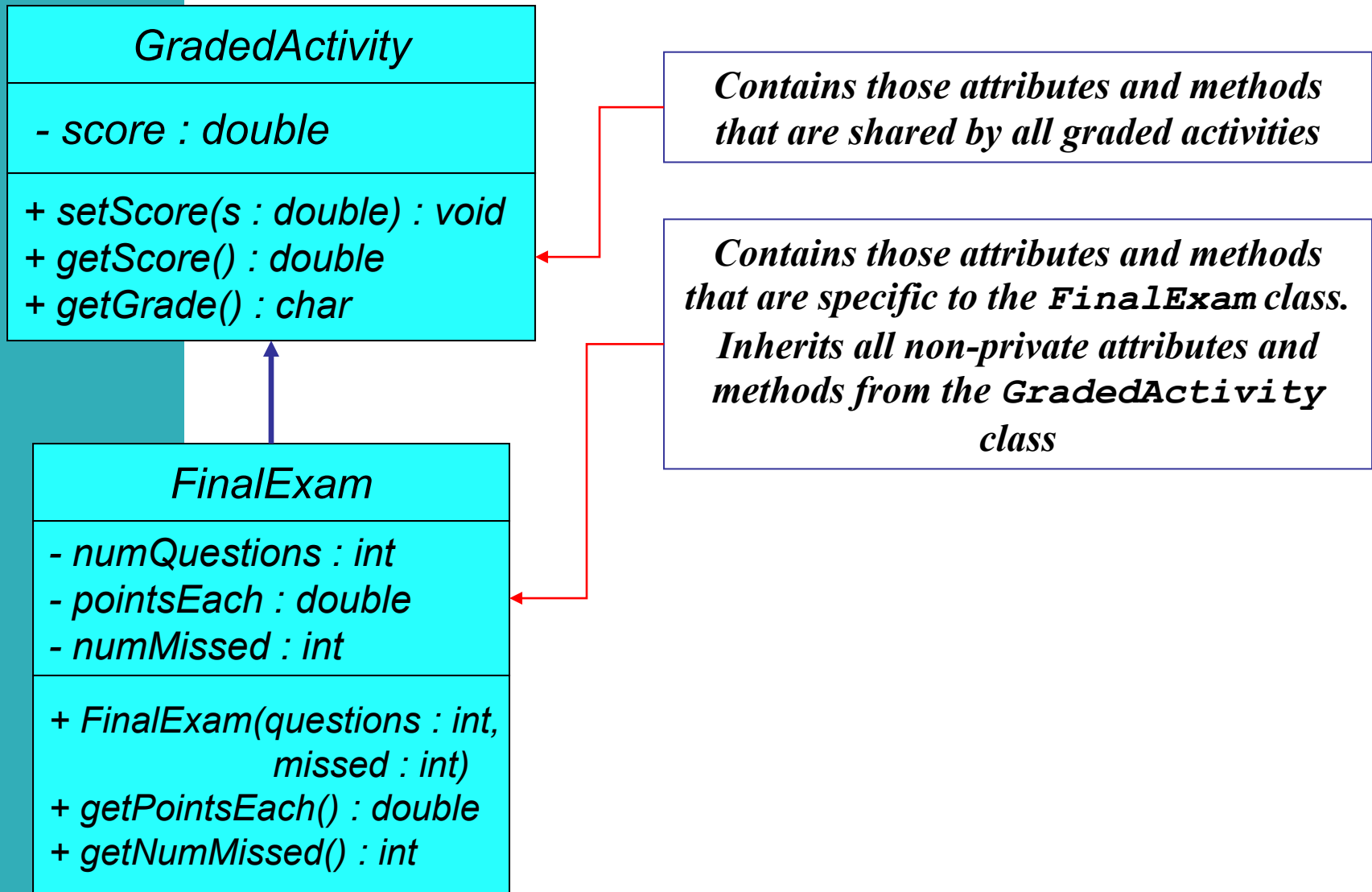

Activity 2:

Poll

1. When an "is a" relationship exists between objects, it means that the specialized object has:

- ☐ some of the characteristics of the general class, but not all, plus additional characteristics
- ☐ some of the characteristics of the general object, but not all
- ☐ none of the characteristics of the general object
- ☐ all the characteristics of the general object, plus additional characteristics

The GradedActivity Example



Inheritance, Fields and Methods

- Members of the superclass that are marked *private*:
 - are not inherited by the subclass
 - exist in memory when the object of the subclass is created
 - may only be accessed from the subclass by public methods of the superclass
- Members of the superclass that are marked *public*:
 - are inherited by the subclass
 - may be directly accessed from the subclass

Inheritance, Fields and Methods

- When an instance of the subclass is created, the non-private methods of the superclass are 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 are available in the subclass

```
setScore(newScore);
```

Inheritance and Constructors

- Constructors are not inherited
- When a subclass is instantiated, the superclass default constructor is executed first

The Superclass's Constructor

- The `super` keyword refers to an object's superclass
- The superclass constructor can be explicitly called from the subclass by using the `super` keyword

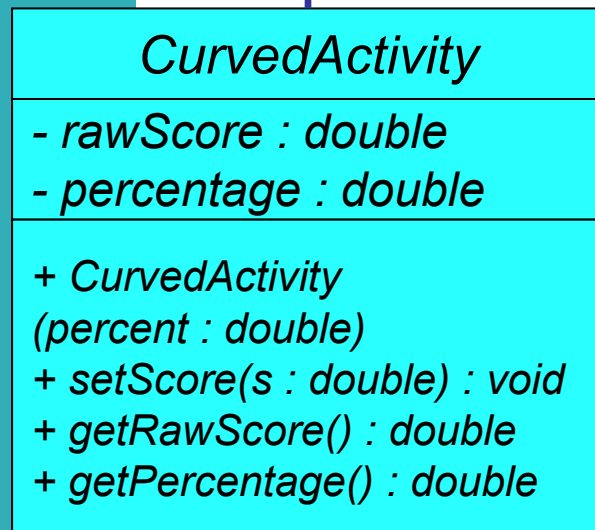
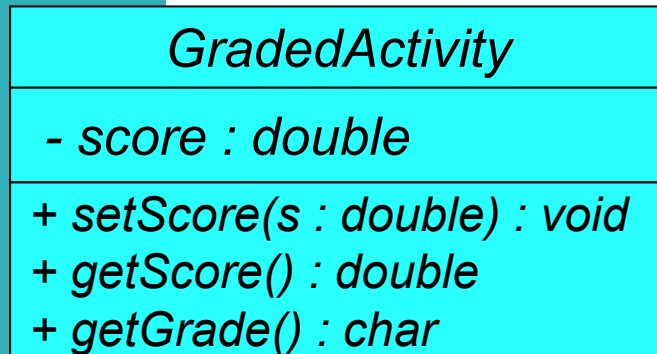
Calling The Superclass Constructor

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

Overriding Superclass Methods

- A subclass may have a method with the same signature as a superclass method
- The subclass method overrides the superclass method
- This is known as *method overriding*

Overriding Superclass Methods



*This method is a more specialized version of the **setScore** method in the superclass, **GradedActivity***

Activity 3:

Discussion question

Discuss the UML diagram components shown in the previous slide

Overriding Superclass Methods

- Recall that a method's *signature* consists of:
 - the method's name
 - the data types method's parameters in the order that they appear
- A subclass method that overrides a superclass method must have the same signature as the superclass method
- An object of the subclass invokes the subclass's version of the method, not the superclass's
- The `@Override` annotation should be used just before the subclass method declaration
 - This causes the compiler to display an error message if the method fails to correctly override a method in the superclass

Overriding Superclass Methods

- A subclass method can call the overridden superclass method via the super keyword

```
super.setScore(rawScore * percentage);
```

- There is a distinction between overloading a method and overriding a method
- Overloading is when a method has the same name as one or more other methods, but with a different signature
- When a method overrides another method, however, they both have the same signature

Overriding Superclass Methods

- Both overloading and overriding can take place in an inheritance relationship
- Overriding can only take place in an inheritance relationship

Preventing a Method from Being Overridden

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

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

- If a subclass attempts to override a final method, the compiler generates an error
- This ensures that a particular superclass method is used by subclasses rather than a modified version of it

Protected Members

- Protected members of class may be accessed:
 - by methods in a subclass
 - by methods in the same package as the class
- Java provides a third access specification called `protected`
- A *protected* member's access is somewhere between *private* and *public*

Protected Members

- Using `protected` instead of `private` makes some tasks easier
- However, any class that is derived from the class, or is in the same package, has unrestricted access to the protected member
- It is always better to make all fields `private` and then provide `public` methods for accessing those fields
- If no access specifier for a class member is provided, the class member is given *package access* by default
- Any method in the same package may access the member

Access Specifiers

Access Modifier	Accessible to a subclass inside the same package?	Accessible to all other classes inside the same package?
default (no modifier)	Yes	Yes
Public	Yes	Yes
Protected	Yes	Yes
Private	No	No

Access Modifier	Accessible to a subclass outside the package?	Accessible to all other classes outside the package?
default (no modifier)	No	No
Public	Yes	Yes
Protected	Yes	No
Private	No	No

Activity 4:

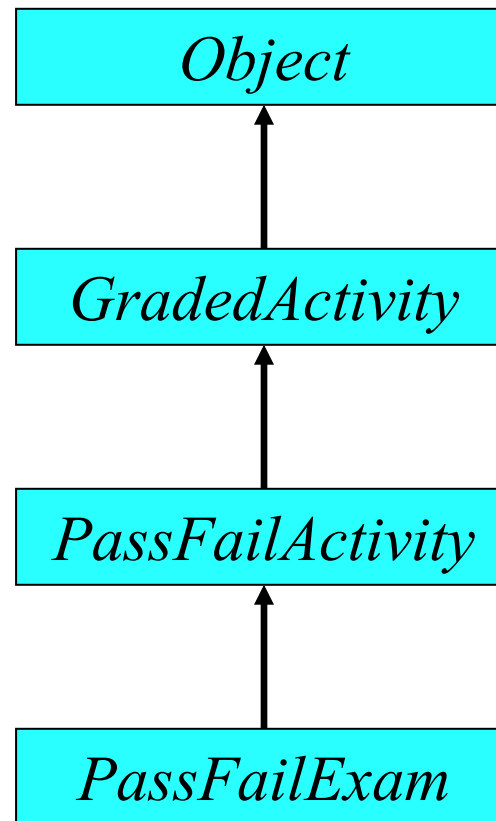
Discussion question

Identify the different access modifiers shown in the previous slide for:

- **Accessible to a subclass inside the same package**
- **Accessible to all other classes inside the same package**
- **Accessible to a subclass outside the package**
- **Accessible to all other classes outside the package**

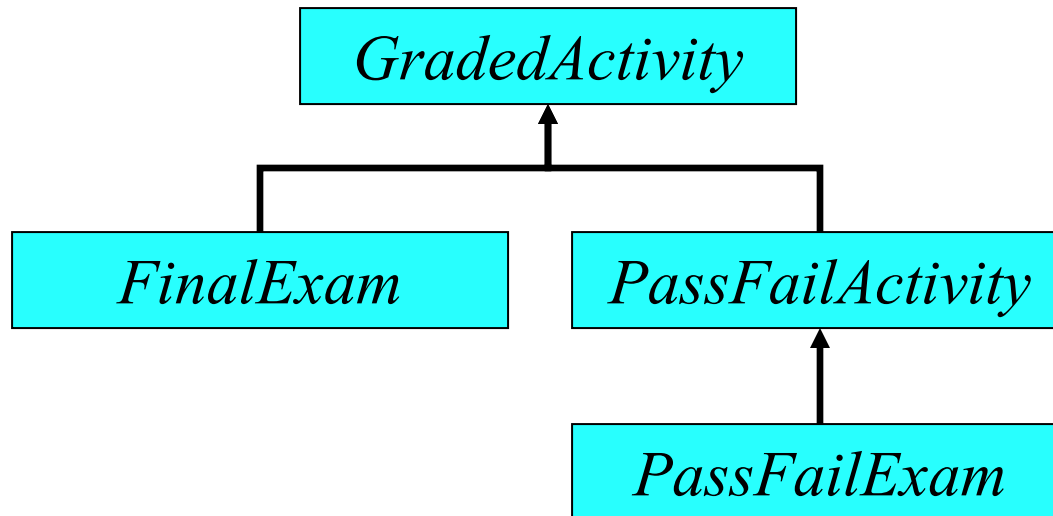
Chains of Inheritance

- A superclass can also be derived from another class



Chains of Inheritance

- Classes often are depicted graphically in a *class hierarchy*
- A class hierarchy shows the inheritance relationships between classes



Activity 5:

Discussion question

Identify the different subclasses and super classes hierarchy from the previous two slides

The Object Class

- All Java classes are directly or indirectly derived from a class named `Object`
- `Object` is in the `java.lang` package
- Any class that does not specify the `extends` keyword is automatically derived from the `Object` class

```
public class MyClass
{
    // This class is derived from
    Object.
}
```

- Ultimately, every class is derived from the `Object` class

The Object Class

- Because every class is directly or indirectly derived from the `Object` class:
 - every class inherits the `Object` class's members
 - example: `toString()` and `equals()`
- In the `Object` class, the `toString()` method returns a string containing the object's class name and a hash of its memory address
- The `equals()` method accepts the address of an object as its argument and returns true if it is the same as the calling object's address

Summary of today's lesson

Inheritance

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

Activity 6: Reflection Exercise

*List any four concepts you have learnt
in today's lesson*

Activity 7: Homework Exercise

Write a program using Eclipse or NetBeans to implement any one concept you have learnt in today's lesson