**Programming with Java – Inheritance, Polymorphism and Packages**

**Key Terms**

• **abstract class**—A class that is declared with the reserved word abstract. An object of an abstract class cannot be instantiated.

• **abstract method**—A method that has a heading but no body.

• **aggregation**—refer to composition

• **base class**—The existing class, or superclass from which other classes are derived.

• **composition**—A "has a" relationship between two classes.

• **derived class**—A subclass created from an existing class.

• **dynamic binding**—The method to be executed is determined at execution time rather than compile time. Also called run-time binding.

• **inheritance**—A meaningful way to relate two or more classes with the "is-a" relationship between them.

• **interface**—A type of class that contains only abstract methods and/or named constants and is defined using the reserved word interface rather than class.

• **late binding**— refer to dynamic binding

• **multiple inheritance**—The subclass is derived from more than one superclass. Java does not support multiple inheritance, but instead uses interfaces.

• **overriding**—Declaring a method in a subclass with the same signature as a method in its superclass.

• **polymorphic reference variable**—A reference variable that can refer to objects of its own class or to objects of the subclasses inherited from its class.

• **polymorphism**—Assigning multiple meanings to the same method name.

• **redefining**—refer to overriding

• **run-time binding**— refer to dynamic binding

• **single inheritance**—The subclass is derived from only one existing superclass.

• **subclass**—A new class created from an existing class.

• **superclass**—The existing class or base class from which other classes are derived.

**Programming with Java – Inheritance, Polymorphism and Packages**

Polymorphism

1. Java allows us to treat an object of a subclass as an object of its superclass. That is, a reference variable of a superclass type can point to an object of its subclass.

2. Walk through the code examples.

3. Explain that late binding refers to the situation in which the method that gets executed is determined at execution time, not at compile time, and note that late binding is also called dynamic binding or run-time binding.

4. Review code examples of late binding.

**Programming Tip**

Object-oriented programming and an example of dynamic binding.

<http://www.coderanch.com/t/452346/java/java/dynamic-binding>

5. Emphasize that a reference variable can invoke a method of its own class or of its subclass(es).

6. Explain that binding means associating a method definition with its invocation, and in early binding, a method’s definition is associated with its invocation when the code is compiled.

7. Explain that in late binding, a method’s definition is associated with the method’s invocation at execution time.

8. Explain that, except for a few (special) cases, Java uses late binding for all methods.

9. Explain that the term polymorphism means assigning multiple meanings to the same method name, and in Java, polymorphism is implemented using late binding.

10. Walk through code examples.

**Programming Tip**

You can declare a method of a class final by using the keyword final. If a method of a class is declared final, it cannot be overridden with a new definition in a derived class.

**Programming Tip**

You can also declare a class final using the keyword final. If a class is declared final, no other class can be derived from this class.

**Programming Tip**

Java does not use late binding for methods that are marked private, final, or static.

11. Polymorphic reference variables can refer to either objects of their own class or objects of the subclasses inherited from their class.

12. A reference variable of a superclass type can point to an object of its subclass. However, you cannot automatically consider a superclass object to be an object of a subclass.

13. i you try to make a reference variable of the subclass point to the object of the superclass, Java will throw a ClassCastException.

Operator instanceof

1. An object of a subclass type can be considered an object of the superclass type, and by using an appropriate cast operator, you can treat an object of a superclass type as an object of a subclass type.

2. To determine whether a reference variable that points to an object is of a particular class type, Java provides the operator instanceof.

3. Walk through examples to demonstrate the use of the operator instanceof.

**Abstract Methods and Classes**

1. An abstract method is a method that has only the heading with no body, and the heading of an abstract method contains the reserved word abstract and ends with a semicolon.

2. An abstract class is a class that is declared with the reserved word abstract in its heading.

3. Walk through the code example to illustrate the declaration of an abstract class.

4. Abstract classes are used as superclasses from which other subclasses within the same context can be derived, and they can be used to force subclasses to provide certain methods.

**Programming Tip**

Why might a software engineer want to use an abstract class to force subclasses to provide certain methods.

Interfaces

1. Reminder: you have seen the interfaces ActionListener, and other similar classes are WindowListener and MouseListener.

2. Reminder: you previously handled events using an inner class, which was built on the interface ActionListener using the keyword implements.

3. Java does not support multiple inheritance, and a class can be derived from only one existing class.

4. Java allows a class to implement more than one interface, and this is how Java implements multiple inheritance, which is not true multiple inheritance.

5. An interface is a class that contains only abstract methods and/or named constants.

**Programming Tip**

If a class contains an abstract method, it must be declared abstract. Moreover, you cannot instantiate an object of an abstract class. Therefore, if a class implements an interface, it must provide definitions for each of the methods of the interface; otherwise, you cannot instantiate an object of that class type.

**Polymorphism via Interfaces**

1. One of the main uses of interfaces is to allow GUI programs to handle more than one type of event, such as window events, mouse events, and action events. These events are handled by separate interfaces.

2. An interface can also be used in the implementation of abstract data types.

3. Like some other languages, such as C++, you cannot separate the definition of a class from the definitions of its methods.

4. One way to accomplish this is to define an interface that contains the methods headings and/or named constants. Then you can define the class that implements the interface. The user can look at the interface and see what operations are implemented by the class.

5. You can also create polymorphic references using interfaces. You can use an interface name as the type of a reference variable, and the reference variable can point to any object of any class that implements the interface.

6. Because an interface contains only method headings and/or named constants, you cannot create an object of an interface.

**Composition (Aggregation)**

1. Explain that in composition (aggregation), one or more members of a class are objects of another class type; thus, composition is a "has-a" relation.

Programming Example: Grade Report

1. Introduce the programming example to produce a student grade report if the student’s tuition has been paid or to print a message indicating that the grades are being held until the tuition is paid.

2. Explain that the program has two parts. The first part generates the grade report in the window’s console environment and stores the output in a file, and the second part creates a GUI to display the students’ grade reports.

3. Walk through the problem analysis and algorithm design and methods and constructors, and step through the program listing for the main method.

Quick Quiz 1

1. The class InputStreamReader is derived from the class Reader, which is derived from the class \_\_\_\_\_\_\_\_\_\_.

Answer: Object

2. To be able to handle a variety of events, Java allows a class to implement more than one \_\_\_\_\_\_\_\_\_\_.

Answer: interface

3. An abstract method is a method that has only the \_\_\_\_\_\_\_\_\_\_ with no body.

Answer: heading

4. True or False: Composition is an “is-a” relationship.

Answer: False

5. True or False: You cannot instantiate an object of an abstract class type. You can only declare a reference variable of an abstract class type.

Answer: True

Class Discussion Topics

1. Why would a software engineer create an abstract class to force subclasses to provide certain methods, instead of actually implementing the methods that should be provided?

2. When should an interface be used?

3. Do all programming languages have polymorphism and inheritance? Why are they useful?