Week 13 Review – Polymorphism

In OOP, polymorphism refers to processing objects differently depending on their data type or class. Specifically it is the ability to share the functionality of the parent class redefine methods or *reshape* behaviors for derived classes.

Example: given a base class shape, polymorphism enables the programmer to define different area methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the area method to it will return the correct results.

Key Terms:

Abstract Classes and methods

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

**abstract void moveTo(double deltaX, double deltaY);**

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

**public abstract class GraphicObject {**

**// declare fields**

**// declare nonabstract methods**

**abstract void draw();**

**}**

When an abstract class is subclassed, the subclass usually provides implementations for *all* of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Concrete classes

As mentioned an abstract class is meant to be used as the base class from which other classes are derived. The derived class is expected to provide implementations for the methods that are not implemented in the base class. A derived class that implements all the missing functionality is called a *concrete class.*

Abstract Classes Compared to Interfaces

**Abstract classes** are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods.

With **interfaces**, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Which should you use, abstract classes or interfaces?

* Consider using abstract classes if any of these statements apply to your situation:
  + You want to share code among several closely related classes.
  + You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  + You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
* Consider using interfaces if any of these statements apply to your situation:
  + 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 of type.

1. All abstract classes must contain abstract methods.

a. True b. False

2. Polymorphism enables you to:

a. program in the general.

b. program in the specific.

c. absorb attributes and behavior from previous classes.

d. hide information from the user.

3. The following object declaration for the abstract class GraphicalObject is a legal statement. If not why not?

**GraphicalObject g = new** GraphicalObject **(new Point (0,0));**

a. True b. False

4. Which of the following statements about interfaces is *false*?

a. An interface describes a set of methods that can be called on an object, providing a default implementation for the methods.

b. An interface describes a set of methods that can be called on an object, not providing concrete implementation for the methods.

c. Interfaces are useful when attempting to assign common functionality to possibly unrelated classes.

d. Once a class implements an interface, all objects of that class have an is-a relationship with the interface type.

5. For which of the following would polymorphism *not* provide a clean solution?

a. A billing program where there is a variety of client types that are billed with different fee structures.

b. A maintenance log program where data for a variety of types of machines is collected and maintenance schedules are produced for each machine based on the data collected.

c. A program to compute a 5% savings account interest for a variety of clients.

d. An IRS program that maintains information on a variety of taxpayers and determines who to audit based on criteria for classes of taxpayers.

6. Polymorphism allows for specifics to be dealt with during:

a. executionb. compilation c. programming d. debugging

7. Which statement *best* describes the relationship between superclass and subclass types?

a. A subclass reference *cannot* be assigned to a superclass variable and a superclass reference *cannot* be assigned to a subclass variable.

b. A subclass reference *can* be assigned to a superclass variable and a superclass reference *can* be assigned to a subclass variable.

c. A superclass reference *can* be assigned to a subclass variable, but a subclass reference *cannot* be assigned to a superclass variable.

d. A subclass reference *can* be assigned to a superclass variable, but a superclass reference *cannot* be assigned to a subclass variable.

8. It is a UML convention to denote the name of an abstract class in:

a. bold b. italics c. a diamond d. no convention exists to denote abstract classes

9. If the superclass contains only abstract method declarations, the superclass is used for:

a. implementation inheritance b. interface inheritance c. Both d. Neither

10. Which of the following could be used to declare abstract method method1 in abstract class Class1. (method1 returns an int and takes no arguments)?

a. public int method1();

b. public int abstract method1();

c. public abstract int method1();

d. public int nonfinal method1();

11. Which of the following statements about abstract superclasses is *true*?

a. abstract superclasses may contain data.

b. abstract superclasses may *not* contain implementations of methods.

c. abstract superclasses must declare all methods as abstract.

d. abstract superclasses must declare *all* data members not given values as abstract.

12. Consider the abstract superclass below:

public abstract class Foo  
{  
 private int a;  
 public int b;  
  
 public Foo( int aVal, int bVal )  
 {  
 a = aVal;  
 b = bVal;  
 } // end Foo constructor  
  
 public abstract int calculate();  
} // end class Foo

Any *concrete* subclass that *extends* class Foo:

a. Must implement method calculate b. Will *not* access the instance variable a

c. Neither (a) nor (b) d. Both (a) and (b)

13. Consider classes A, B and C, where A is an abstract superclass, B is a concrete class that inherits from A and C is a concrete class that inherits from B. Class A declares abstract method originalMethod, implemented in class B. Which of the following statements is *true* of class C?

a. Method originalMethod cannot be overridden in class C—once it has been implemented in concrete class B, it is implicitly final.

b. Method originalMethod *must be* overridden in class C, or a syntax error will occur.

c. If method originalMethod is not overridden in class C but is called by an object of class C, an error occurs.

d. None of the above.

14. When a superclass variable refers to a subclass object and a method is called on that object, the proper implementation is determined at execution time. What is the process of determining the correct method to call?

a. early binding b. non-binding c. on-time binding d. late binding

15. All of the following methods are implicitly final except:

a. a method in an abstract class. b. a private method.

c. a method declared in a final class. d. static method.