ORACLE ACADEMY

Section 7 Lesson 5: Understanding Polymorphism
Try It: Practice Activities

Objectives

- Apply superclass references to subclass objects
- Write code to override methods
- Use dynamic method dispatch to support polymorphism
- Create abstract methods and classes
- Recognize a correct method override
- Use the final modifier
- Explain the purpose and importance of the Object class
- Write code for an applet that displays two triangles of different colors
- Describe object references

Vocabulary

Identify the vocabulary word for each definition below.

	A concept in object oriented programming that allows classes to have many forms and behave like their superclasses.
	Implementing methods in a subclass that have the same prototype (the same parameters, method name, and return type) as another method in the superclass.
	A keyword in Java used to limit subclasses from extending a class, overriding methods or changing data.
	A property of a static class that makes the class unable to be extended or data to be changed.
	Implementing a method with the same name as another method in the same class that has different parameters or a different return type.
	The process by which Java is able to determine which method to invoke when methods have been overridden.
	A keyword in Java that allows classes to be extended, but the classes cannot be instantiated (constructed) and when applied to methods, dictates that the methods should be implemented in all subclasses of the class.

Try It/Solve It

1. What will be the output of the following code?

```
class A {
     void callthis() {
```

```
            System.out.println(“Inside Class A's Method!”);
      }
}
class B extends A {
      void callthis() {
            System.out.println(“Inside Class B's Method!”);
      }
}
class C extends A{
      void callthis() {
            System.out.println(“Inside Class C's Method!”);
      }
}

class DynamicDispatch {
      public static void main(String args[])  {
            A a = new A();
            B b = new B();
            C c = new C();
            A ref;

            ref = b;
            ref.callthis();

            ref = c;
            ref.callthis();

            ref = a;
            ref.callthis();
      }
}
```

2. What is the difference between an abstract class and an interface? When is it appropriate to use an abstract class or an interface?

3. Given the information for the following, determine whether they will result: Always compile, sometimes compile, or does not compile.

```
public interface A

public class B implements A

public abstract class C
```

```
public class D extends C
public class E extends B
```

Each class have been initialized, but it is not clear what they have been initialized to:

```
A a = new...
B b = new...
C c = new...
D d = new...
E e = new...
```

The following methods are included:

interface A specifies method void methodA()

class C has the abstract method void methodC()

Code:	Always Compile, Sometimes Compile, or Does Not Compile?
a = new B();	
d = new C();	
b.methodA();	
e.methodA();	
c = new C();	
(D)c.methodC();	

4. Override the toString() method for the class below to output the results, matching the given output. The toString() method should print all the values from 1 to the number specified in num and then print the final value using the provided getFactorial method.

Assume the variable int num is a public global value:

“Factorial: 10! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800”

```
int getFactorial(){
int factorial;
     for(i = num; num > 0; i--){
          factorial *= num;
     }
     return factorial;
   }
public String toString() {
```

```
}
```