Stefan Falciglia

Professor Page

Advanced Technical Writing

03/19/2016

**Polymorphism in relation to JAVA programming**

 The dictionary definition of polymorphism refers to a biological principal in which an organism can have many different forms or stages. This principal was applied to object-oriented programming such as Java. *“Polymorphism means that a variable of a supertype can refer to a subtype object.” (Daniel Liang, 421).* Polymorphism is the ability of a method to do different things based on the object that it is acting upon. It allows you to define one interface and have multiple implementations of this. A very common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

* It is a feature that lets one interface be used for a general class of actions.
* An operation may exhibit different behavior in different instances.
* The behavior depends on the types of data used in the programs.
* It plays an important role in allowing objects having different structures to share the same external interface.
* Polymorphism is extensively used in implementing inheritance.

Before we speak about different types of polymorphism it is important to understand inheritance, because it can be used in polymorphism but it is important to distinguish their differences and similarities. As programmers we have a hard time remembering all the information and are subject to confuse different object oriented concepts especially when they are closely related, i.e polymorphism vs. inheritance. *“Inheritance is an important and powerful feature for reusing software.”* (Daniel Liang, 408). Inheritance is used to define a relationship between 2 classes, more of a father-son relationship. In object oriented programming we have a parent (or super class) and the child class (or sub class). The child class inherits the attributes (or methods) of the parent, just like in real life. Polymorphism is the ability to tell an object to behave in multiple ways. For example if a parent variable is holding a reference to a parent or child object, when you call a method on a parent object, it may go to a child’s method, depending on the pointers location at runtime, this is one form of polymorphism.

This concept of polymorphism is important in Java, it is an advanced topic which requires a little more attention to fully implement. An important fact to keep in mind is to remember that the only possible way to access an object is through a reference variable, this reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed. A reference variable can refer to any object of its declared type or any subtype of its declared type. The reference variable can be reassigned to other objects provided that it is not declared final. A reference variable can be declared as a class or interface type. The type of the reference variable would determine the methods that it can invoke on the object.

2 important terms when discussing polymorphism are subtype and supertype. A class defines a type. A type defined by a subclass is called a subtype, and a type defined by its superclass is called a supertype. *“The inheritance relationship enables a subclass to inherit features from its superclass with additional new features. A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa. For example, every circle is a geometric object, but not every geometric object is a circle. Therefore, you can always pass an instance of a subclass to a parameter of its superclass type.”(Daniel Liang, 421)*

 In Java there are 2 kinds of polymorphism: compile time polymorphism (static binding), and runtime polymorphism (dynamic binding). Static polymorphism is achieved through method overloading (example later). Method overloading means there are methods in a class that have the same name but different types and parameters. At the time java compiles a program, the JVM knows which method to invoke by checking the method signatures. Method overloading is an example of static polymorphism, while method overriding is an example of dynamic polymorphism.

Static or compile time polymorphism is just method overloading in Java. *“Overloading means to define multiple methods with the same name but different signatures.” (Daniel Liang, 418).* Method signatures are the parameters that will be passed through the method; these can have a different order/types/number and still be considered different methods. Which comes in handy when you are using methods that are closely related and have the same method name but do different things. At the time of compilation, Java knows which method to invoke by checking the method header; which is why it is referred to as compile time polymorphism & static binding.

For example - Method Overloading:

class Overload{

 public int add(int x, int y){ //method 1

 return x+y;

 }

 public int add(int x, int y, int z){ //method 2

 return x+y+z;

 }

}

class Tester{

 public static void main(String[] args){

 Overload demo=new Overload();

 System.out.println(demo.add(2,3)); //method 1 called

 System.out.println(demo.add(2,3,4)); //method 2 called

 }

}

*Overriding means to provide a new implementation for a method in the subclass. (Daniel Liang, 418).* Method overriding is an example of runtime or dynamic polymorphism. With this kind of polymorphism class A can hold object of class A or an object of any sub classes of class A. In method overriding both the classes (base and child class) have the same method, compile can’t figure out which method to call at compile time. When this happens the JVM will decide which method to call at runtime, hence its name runtime or dynamic polymorphism.

For example - Method Overriding:

public class X

{

 public void methodA() //Base class method (polymorphic)

 {

 System.out.println (" "); //methodA of class X

 }

}

public class Y extends X

{

 public void methodA() //Derived Class method (polymorphic)

 {

 System.out.println (" "); //methodA of class Y

 }

}

public class Z

{

 public static void main (String args []) {

 X obj1 = new X(); // Reference and object X

 X obj2 = new Y(); // X reference but Y object

 obj1.methodA();

 obj2.methodA();

 }

}

 The 3 pillars of object oriented programming are inheritance, encapsulation, and polymorphism. Learning the 3 would make anyone a much stronger, well-rounded programmer. These are advanced topics and will require not only research but real world application to fully grasp the usefulness of these tools.

Works Cited

Y. Daniel Liang. Introduction to Java Programming 9th Edition; Comprehensive Edition. Prentice Hall. New Jersey, 2011. Print.

Java Programming Tutorial OOP - Inheritance & Polymorphism- April 2013. Web.

<http://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html>

Java – Polymorphism- 2016. Web.

<http://www.tutorialspoint.com/java/java_polymorphism.htm>

Polymorphism. (Oracle Reference). 1995,2015. Web.

<http://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>

Sandeep Panda. Quick Guide to Polymorphism in Java- Jan 2013. Web.

<http://www.sitepoint.com/quick-guide-to-polymorphism-in-java/>