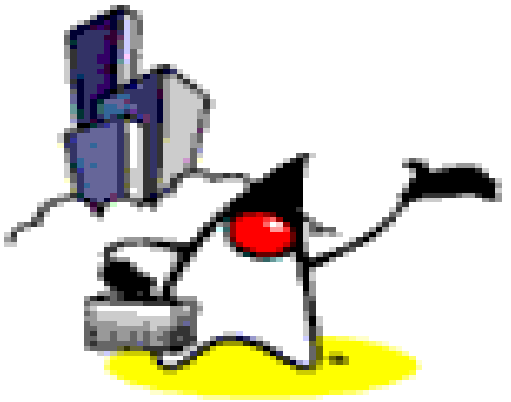
Polymorphism
(Importance scale: ***, Important)

Sang Shin
Michèle Garoche
www.JPassion.com
"Learn with JPassion!"

Agenda

- What is and Why Polymorphism?
- Examples of Polymorphism in Java programs
- 3 forms of Polymorphism



What is & Why Polymorphism?

What is Polymorphism?

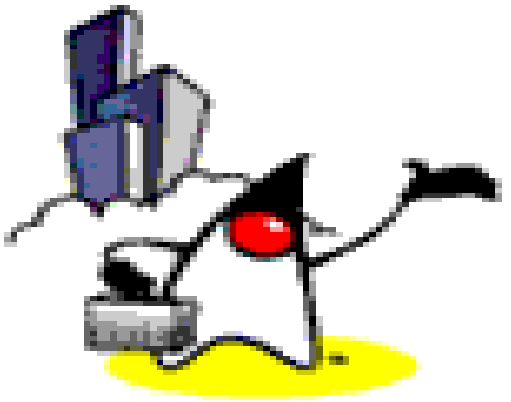
- Generally, polymorphism refers to the ability to appear in many forms
- Polymorphism in a Java program
  - The ability of a reference variable to change behavior according to what object instance it is referring to.
  - This allows multiple objects of different subclasses to be treated as objects of a single super class, while automatically selecting the proper methods to apply of a particular object based on the subclass it belongs to

Polymorphism Mechanics

- Let's say *Shape* is a parent class, and it has *Circle*, *Rectangle* and *Triangle* as child classes.
- *Shape* parent class has *computeArea()* method.
- *Circle*, *Rectangle*, and *Triangle* have their own implementation of *computeArea()* method overriding the *computeArea()* method of the *Shape* parent class
  - Because the logic of computing the area of Circle, Rectangle, and Triangle has to be different

Polymorphism Mechanics (Cont.)

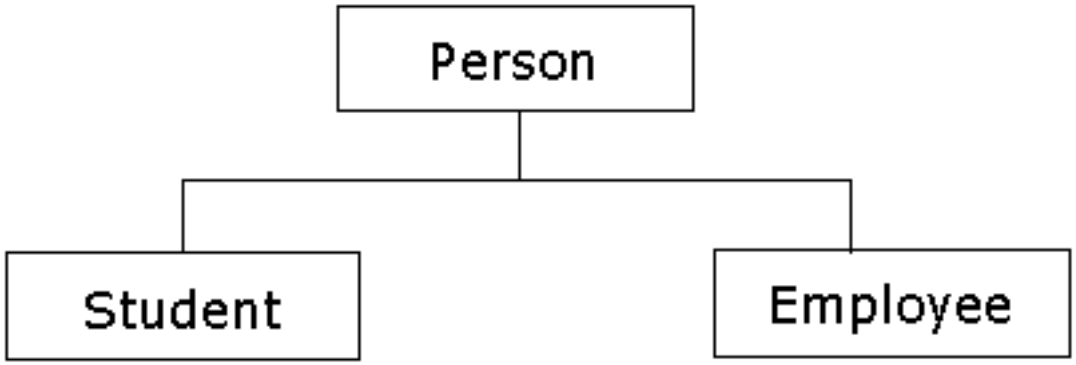
- Polymorphism enables the programmer to provide different implementation of the *computeArea()* method for any number of child classes of *Shape* parent class, such as *Circle*, *Rectangle* and *Triangle*.
- No matter what shape an object is (In other words, no matter what subclass' *computeArea()* method is used), applying the *computeArea()* method of the *Shape* to it will return the correct result
  - Because the correct version of the *computeArea()* method will be invoked (We will see examples in the following slides)



Examples of Polymorphic Behavior in Java Programs

Example #1: Polymorphism

- Let's say the parent class **`Person`** has two subclasses **`Student`**, and **`Employee`**.
- Class hierarchy is as following



Example #1: Polymorphism

- Now suppose we have a getName method in our super class Person, and we override this method in both Student and Employee subclass's

```
public class Student extends Person {
   public String getName(){
      System.out.println("Student Name:" + name);
      return name;
   }
}

public class Employee extends Person {
   public String getName(){
      System.out.println("Employee Name:" + name);
      return name;
   }
}
```

Example #1: Polymorphism

- In Java, we can have a reference variable that is of type super class, *Person*, points to an object of its subclass, *Student*.

```
public static main( String[] args ) {

    Student studentObject = new Student();
    Employee employeeObject = new Employee();

    // refPerson is Person type but points
    // to a Student object
    Person refPerson = studentObject;

    // Calling getName()of refPerson calls the
    // getName() of the Student class not Person
    // class because refPerson points to Student object
    String name = refPerson.getName();
}
```

Example #1: Polymorphism

- Now, if we assign **`Employee`** object to the refPerson variable, the **`getName`** method of **`Employee`** will be called.

```
    // refPerson is Person type but points
    // to an Employee object
    refPerson = employeeObject;

    // Calling getName()of refPerson calls the
    // getName() of the Employee class not Person
    // class because refPerson points to Employee object
    String name = refPerson.getName();
}
```

Example #1: Polymorphism

```
1   public static main( String[] args ) {
2
3       Student studentObject = new Student();
4       Employee employeeObject = new Employee();
5
6      // refPerson points to a Student object
7       Person refPerson = studentObject;
8
9       // getName() method of Student class is called
10      String temp= refPerson.getName();
11      System.out.println( temp );
12
13     // refPerson now points to an Employee object
14      refPerson = employeeObject;
15
16      //getName() method of Employee class is called
17      String temp = refPerson.getName();
18      System.out.println( temp );
19  }
```

Example #2: Polymorphism

- Another example that illustrates polymorphism is when we try to pass a reference to methods as a parameter
- Suppose we have a static method printInformation that takes in a Person reference as parameter.

```
public static printInformation( Person p ){
    // It will call getName() method of the
    // actual object instance that is passed
    p.getName();
}
```

Example #2: Polymorphism

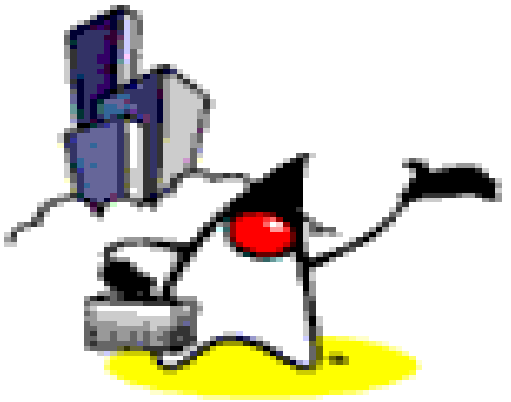
- We can actually pass a reference of type Employee or type Student to the printInformation method as long as it is a subclass of the Person class.

```
public static main( String[] args ){

        Student   studentObject = new Student();
        Employee  employeeObject = new Employee();

        printInformation( studentObject );

        printInformation( employeeObject );
}
```



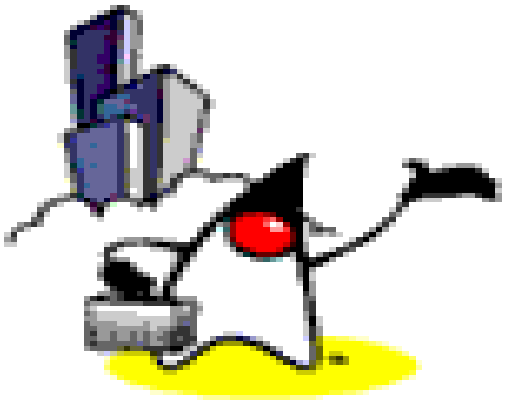
Benefits of Polymorphism

Benefits of Polymorphism: Simplicity

- If you need to write code that deals with a family of sub-types, the code can ignore type-specific details and just interact with the base type of the family
  - Higher level of abstraction results in simpler code
- Even though the code thinks it is using an object of the base class, the object's class could actually be the base class or any one of its subclasses
- This makes your code easier to write and easier for others to understand

Benefits of Polymorphism: Extensibility

- Other subclasses could be added later to the family of types, and objects of those new subclasses would also work with the existing code



3 Forms of Polymorphism

3 Forms of Polymorphism in Java program

- Scheme #1 - Method overriding
  - Methods of a subclass override the methods of a superclass
  - This is the example code we've seen so far
- Scheme #2 - Implementation of abstract methods of an Abstract class
  - Methods of a subclass implement the abstract methods of an abstract class
- Scheme #3 - Implementation of abstracts methods of a Java interface
  - Methods of a concrete class implement the methods of the interface

Scheme #1: Overriding a Parent Class (This is the same example we've seen)

```
1   public static main( String[] args ) {
2
3       Student studentObject = new Student();
4       Employee employeeObject = new Employee();
5
6      // refPerson points to a Student object
7       Person refPerson = studentObject;
8
9       // getName() method of Student class is called
10      String temp= refPerson.getName();
11      System.out.println( temp );
12
13     // refPerson now points to an Employee object
14      refPerson = employeeObject;
15
16      //getName() method of Employee class is called
17      String temp = refPerson.getName();
18      System.out.println( temp );
19  }
```

Scheme #2: Implementing abstract methods of an Abstract Class

```
1   public static main( String[] args ) {
2
3       Student studentObject = new Student();
4       Employee employeeObject = new Employee();
5
6      // refPerson points to a Student object
7       PersonAbstract refPerson = studentObject;
8
9       // getName() method of Student class is called
10      String temp= refPerson.getName();
11      System.out.println( temp );
12
13     // refPerson now points to an Employee object
14      refPerson = employeeObject;
15
16      //getName() method of Employee class is called
17      String temp = refPerson.getName();
18      System.out.println( temp );
19  }
```

Scheme #3: Implementing abstract methods of a Java Interface

```
1   public static main( String[] args ) {
2
3       Student studentObject = new Student();
4       Employee employeeObject = new Employee();
5
6       // refPerson points to a Student object
7       PersonInterface refPerson = studentObject;
8
9       // getName() method of Student class is called
10      String temp= refPerson.getName();
11      System.out.println( temp );
12
13      // refPerson now points to an Employee object
14      refPerson = employeeObject;
15
16      //getName() method of Employee class is called
17      String temp = refPerson.getName();
18      System.out.println( temp );
19  }
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with JPassion!"