Java Polymorphism

"Code with Passion!"

Topics

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

What is Polymorphism?

- In dictionary terms, polymorphism refers to
  - The ability to appear in many forms
- Polymorphism in a Java program refers to
  - The ability of a reference variable to change behavior according to what object instance it is referring to
  - This allows 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
  - This allows objects of implementation classes to be treated as objects of an interface, while automatically selecting the proper methods to apply of a particular object based on implementation class

Polymorphism Mechanics

- Let's say *Shape* is a parent class, and it has the following child classes
  - *Circle*, *Rectangle* and *Triangle*
- *Shape* parent class has *computeArea()* method.
- *Circle*, *Rectangle*, and *Triangle* child classes 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 must be different

Polymorphism Mechanics (Cont.)

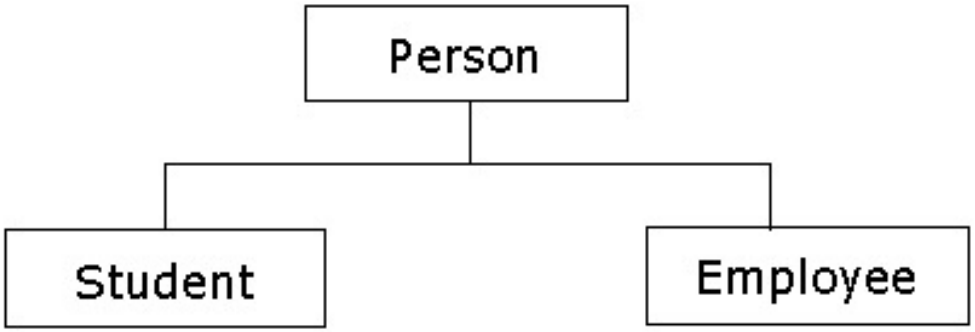
- Polymorphism enables the programmer to provide different implementation of the *computeArea()* method for any number of child classes of *Shape* parent class, such as *Circle*, *Rectangle* and *Triangle*

- 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 child classes **`Student`**, and **`Employee`**.
- Class hierarchy is as following



Example #1: Polymorphism

- Now suppose we have a getName() method in the parent class Person, and we override this method in both Student and Employee child classes

```
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 (*this is very important point to understand)

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

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

   // refPerson reference variable 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 reference variable 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        Student studentObject = new Student();
3        Employee employeeObject = new Employee();

4        // refPerson points to a Student object
5        Person refPerson = studentObject;
6
7        // getName() method of Student class is called
8        String temp= refPerson.getName();
9        System.out.println( temp );

10       // refPerson now points to an Employee object
11       refPerson = employeeObject;
12
13       // getName() method of Employee class is called
14       String temp = refPerson.getName();
15       System.out.println( temp );
16   }
```

So depending on which object it points to, the implementation method of the pointed object gets invoked. This is polymorphic behavior.

Example #2: Polymorphism

- Another example that illustrates polymorphism is when we try to pass a reference type as an argument
- Suppose we have a method printInformation that takes in a Person reference type as argument

```
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 implementation 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 be the base class or any one of its subclasses
  - Depending on which type of object gets used
- 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 subtypes, and objects of those new subclasses would also work with the existing code
  - No code change is required in order to accommodate the addition of the other subtypes in the future

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

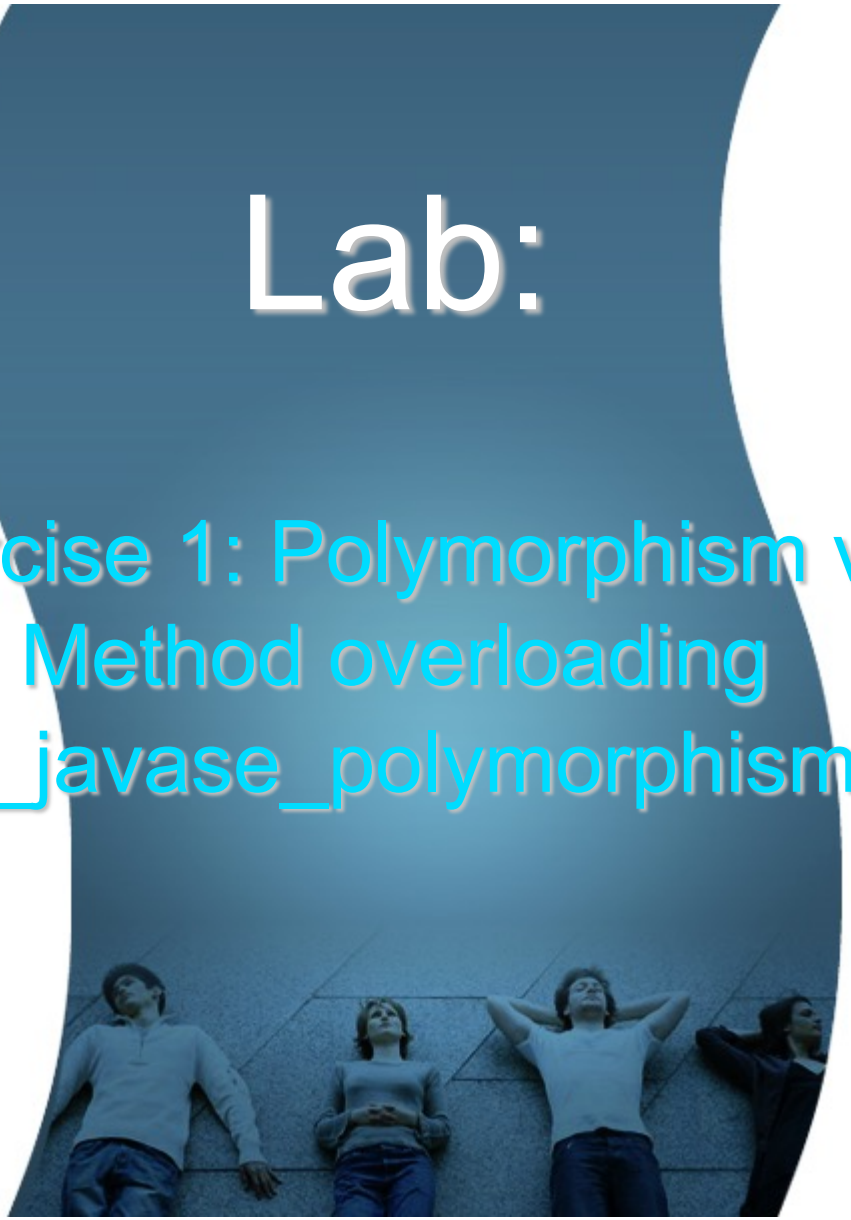
Scheme #1: Overriding a Parent Class (This is the same example we've seen)

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

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

4         // refPerson points to a Student object
5         Person refPerson = studentObject;
6
7         // getName() method of Student class is called
8         String temp= refPerson.getName();
9         System.out.println( temp );

10        // refPerson now points to an Employee object
11        refPerson = employeeObject;
12
13        // getName() method of Employee class is called
14        String temp = refPerson.getName();
15        System.out.println( temp );
16    }
```

Lab:

Exercise 1: Polymorphism via Method overloading

1025_javase_polymorphism.zip



Scheme #2: Implementing abstract methods of an Abstract Class

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

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

4         // refPerson points to a Student object
5         PersonAbstract refPerson = studentObject;
6
7         // getName() method of Student class is called
8         String temp= refPerson.getName();
9         System.out.println( temp );

10        // refPerson now points to an Employee object
11        refPerson = employeeObject;
12
13        //getName() method of Employee class is called
14        String temp = refPerson.getName();
15        System.out.println( temp );
16    }
```

Lab:

Exercise 2: Polymorphism via Abstract Class

1025_javase_polymorphism.zip

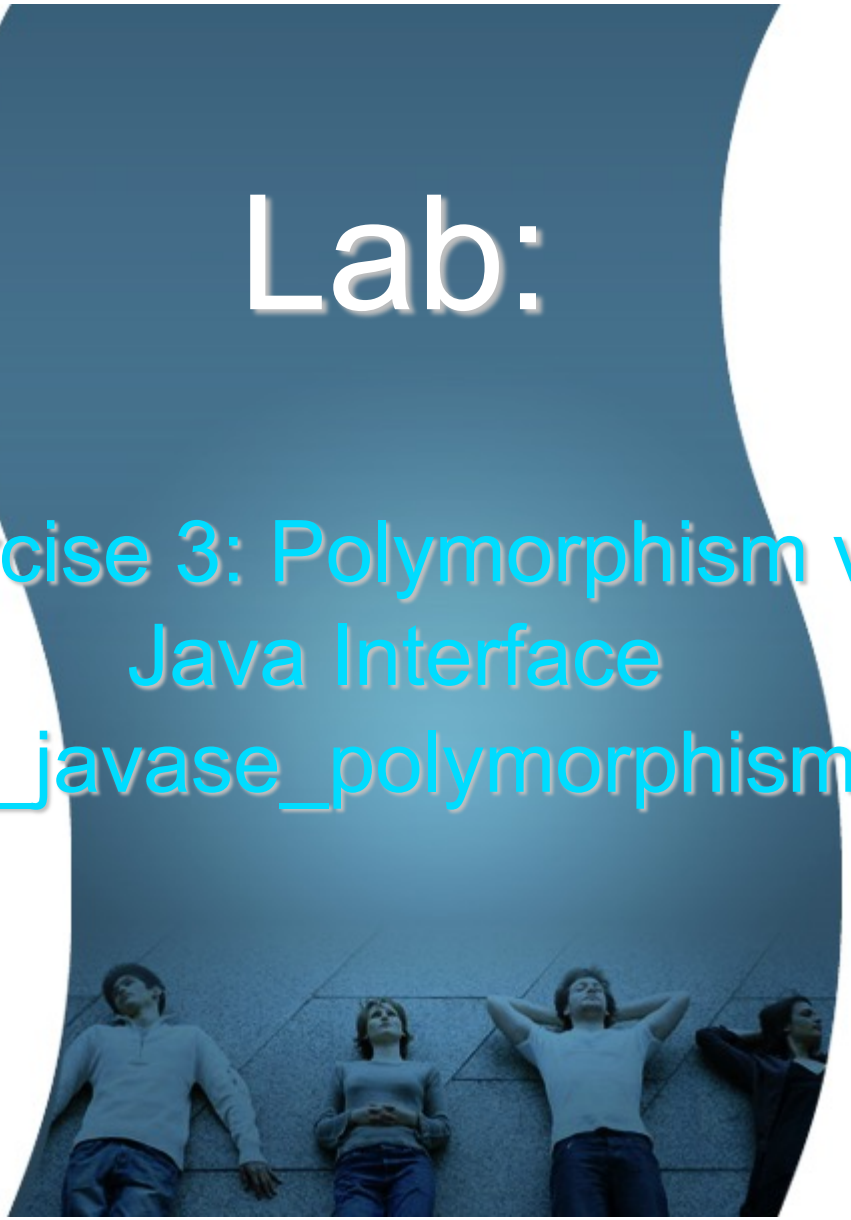
Scheme #3: Implementing abstract methods of a Java Interface

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

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

4         // refPerson points to a Student object
5         PersonInterface refPerson = studentObject;
6
7         // getName() method of Student class is called
8         String temp= refPerson.getName();
9         System.out.println( temp );

10        // refPerson now points to an Employee object
11        refPerson = employeeObject;
12
13        //getName() method of Employee class is called
14        String temp = refPerson.getName();
15        System.out.println( temp );
16    }
```

Lab:

Exercise 3: Polymorphism via Java Interface

1025_javase_polymorphism.zip



Code with Passion!