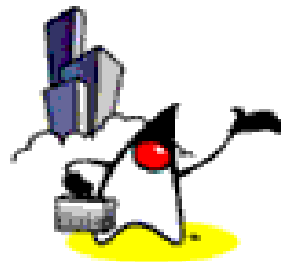




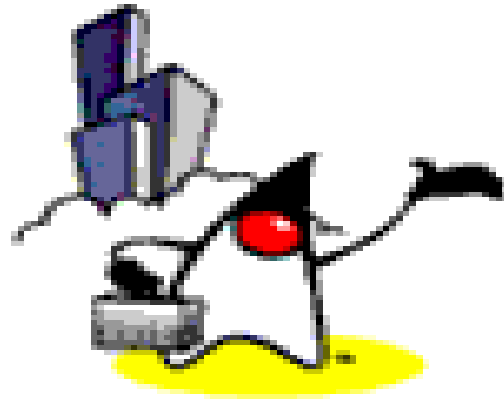
Polymorphism



Agenda

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





What is & Why Polymorphism?

What is Polymorphism?

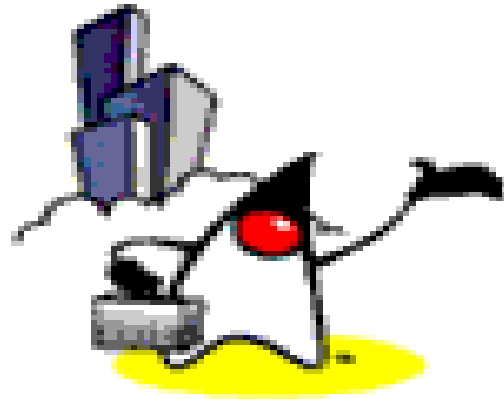
- Generally, polymorphism refers to the ability to appear in many forms
- Polymorphism in a Java program
 - The ability of a reference variable to change behavior according to what object instance it is holding.
 - This allows multiple objects of different subclasses to be treated as objects of a single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



Polymorphism Example

- For 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.

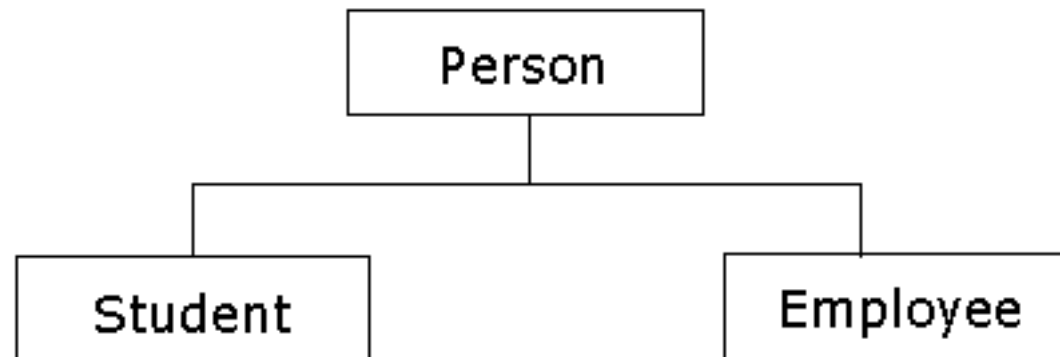




Examples of Polymorphic Behavior in Java Programs

Example #1: Polymorphism

- Given the parent class **Person** and the child class **Student**, we add another subclass of **Person** which is **Employee**.
- Below is the class hierarchy



Example #1: Polymorphism

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

```
public static main( String[] args ) {  
  
    Student studentObject = new Student();  
    Employee employeeObject = new Employee();  
  
    Person ref = studentObject; // Person reference points  
                                // to a Student object  
  
    // Calling getName() of the Student object instance  
    String name = ref.getName();  
}
```



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

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



Example #1: Polymorphism

- Going back to our main method, when we try to call the `getName` method of the reference `Person ref`, the `getName` method of the `Student` object will be called.
- Now, if we assign `ref` to an `Employee` object, the `getName` method of `Employee` will be called.



Example #1: Polymorphism

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

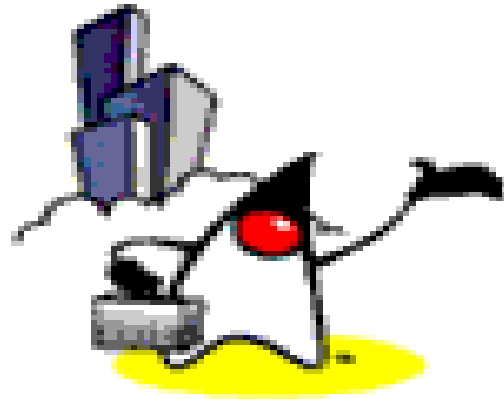


Example #2: Polymorphism

- We can actually pass a reference of type **Employee** and 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

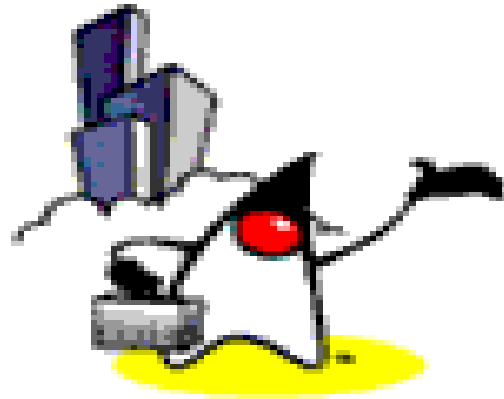
- Simplicity
 - If you need to write code that deals with a family of types, the code can ignore type-specific details and just interact with the base type of the family
 - Even though the code thinks it is using an object of the base class, the object's class could actually be the base class or any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

- Method overriding
 - Methods of a subclass override the methods of a superclass
- Method overriding (implementation) of the abstract methods
 - Methods of a subclass implement the abstract methods of an abstract class
- Method overriding (implementation) through the Java interface
 - Methods of a concrete class implement the methods of the interface





Polymorphism

