Polymorphism

Inheritance provides Power to OOP

Polymorphism provides flexibility through inheritance
Polymorphism via Inheritance

Class `Rectangle` has a method called `area()`, and the child class `Square` overrides it.

Now consider the following invocation

```java
myShape.area();
```

Which `area()` is invoked?

If `myShape` refers to a `Rectangle` object, it invokes the `Rectangle` version of `area()`.

If `myShape` refers to a `Square` object, it invokes the `Square` version of `area()`!
Static and Dynamic Binding

- Consider the following method invocation:
  
  ```java
  myShape.area();
  ```

- At some point, this invocation is `bound` to the definition of the method that it invokes.
  - If this binding occurred `statically` at `compile` time, then that line of code would call the same method every time.

- Java defers method binding until `run` time: this is called `dynamic binding` or `late binding`.

- Dynamic binding provides `flexibility` in program design.
Polymorphism: "having many forms"

- A *polymorphic reference* is a variable that can refer to different types of objects at different points in time.

- Suppose we create the following reference variable:

  ```java
  Rectangle myShape;
  ```

- Java allows this reference to point to a Rectangle object, or to any object of any compatible type!

- This *compatibility* can be established using *inheritance* or using *interfaces*.
/**
 * Demonstrates polymorphism via inheritance.
 * @author Java Foundations
 */

class Firm {

    /**
     * Creates a staff of employees for a firm and pays them.
     */
    public static void main (String[] args) {
        Staff personnel = new Staff();
        personnel.payday();
    }
}
Exploring the benefits and flexibility of polymorphism

A Program that pays various types of employees using a polymorphic method
**Staff.java**

```java
/**
 * Represents the personnel staff of a particular business.
 * @author Java Foundations
 */

public class Staff {
  private StaffMember[] staffList;

  /**
   * Constructor: Sets up the list of staff members.
   */
  public Staff () {
    staffList = new StaffMember[6];

    staffList[0] = new Executive ("Tony", "123 Main Line", "555-0469", "123-45-6789", 2423.07);
    staffList[2] = new Employee ("Vito", "789 Off Rocker", "555-0000", "010-20-3040", 1169.23);
    staffList[3] = new Hourly ("Michael", "678 Fifth Ave.", "555-0690", "958-47-3625", 10.55);
    staffList[5] = new Volunteer ("Benny", "321 Dud Lane", "555-7282");

    ((Executive)staffList[0]).awardBonus (500.00);

    ((Hourly)staffList[3]).addHours (40);
  }
```
/**
 * Pays all staff
 */

public void payday () {
    double amount;

    for (int count=0; count < staffList.length; count++) {
        System.out.println (staffList[count]);

        amount = staffList[count].pay(); // polymorphic

        if (amount == 0.0)
            System.out.println ("Thanks!");
        else
            System.out.println ("Paid: " + amount);

        System.out.println ("----");
    }
}
/**
 * Represents a generic staff member
 * @author Java Foundations
 */

abstract public class StaffMember {
    protected String name;
    protected String address;
    protected String phone;

    /**
     * Constructor: Sets up this staff member using the specified information.
     */
    public StaffMember (String eName, String eAddress, String ePhone) {
        name = eName;
        address = eAddress;
        phone = ePhone;
    }

    /**
     * Derived classes must define the pay method for each type of employee.
     */
    public abstract double pay();

    toString() omitted...
Volunteer.java

```java
/**
 * Represents a staff member that works as a volunteer.
 * @author Java Foundations
 */
public class Volunteer extends StaffMember {

    /**
     * Constructor: Sets up this volunteer using the specified information.
     */
    public Volunteer (String eName, String eAddress, String ePhone) {
        super (eName, eAddress, ePhone);
    }

    /**
     * @return a zero pay value for this volunteer.
     */
    public double pay() {
        return 0.0;
    }
}
```
public class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double payRate;

    public Employee (String eName, String eAddress, String ePhone,
            String socSecNumber, double rate) {
        super (eName, eAddress, ePhone);
        socialSecurityNumber = socSecNumber;
        payRate = rate;
    }

    public double pay() {
        return payRate;
    }
}
/**
 * Represents an executive staff member, who can earn a bonus.
 * @author Java Foundations
 */

public class Executive extends Employee {
    private double bonus;

    /**
     * Constructor: Sets up this executive with the specified information
     */
    public Executive (String eName, String eAddress, String ePhone, String socSecNumber, double rate) {
        super (eName, eAddress, ePhone, socSecNumber, rate);
        bonus = 0; // bonus has yet to be awarded
    }

    /**
     * Computes and returns the pay for an executive, which is the
     * regular employee payment plus a one-time bonus.
     */
    public double pay() {
        double payment = super.pay() + bonus;
        bonus = 0;
        return payment;
    }
}
/**
 * Represents an employee that gets paid by the hour.
 * @author Java Foundations
 */

public class Hourly extends Employee {
    private int hoursWorked;

    /**
     * Constructor: Sets up this hourly employee using the specified information.
     */
    public Hourly (String eName, String eAddress, String ePhone,
                   String socSecNumber, double rate) {
        super (eName, eAddress, ePhone, socSecNumber, rate);
        hoursWorked = 0;
    }

    /**
     * Computes and returns the pay for this hourly employee.
     */
    public double pay() {
        double payment = payRate * hoursWorked;
        hoursWorked = 0;
        return payment;
    }
}