Polymorphism

Flexibility in data types

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 `Rect` object, or to any object of any *compatible type*.
- This compatibility can be established using *inheritance* or using *interfaces*.

**Polymorphism via Inheritance**

- An object reference can refer to an object of its class, **or** to an object of any class related to it by inheritance.
- What determines which method is invoked?

```java
Rectangle myShape;
Square perfect = new Square();
myShape = perfect;
```
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!

A Program that pays a set of diverse employees using a polymorphic method
Firm.java

```java
/**
 * Demonstrates polymorphism via inheritance.
 * @author Java Foundations
 */
public 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();
    }
}
```

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[3];
        stafflist[0] = new Executive("Tony", "123 Main Line", "555-0409", "123-45-6789", 2423.07);
        stafflist[2] = new Employee("Vito", "789 Off Rocker", "555-0000", "012-29-3049", 1109.23);

        (Executive)stafflist[0].awardBonus (500.00);
        (Hourly)stafflist[3].addHours (40);
    }
```
**Staff.java**

```java
/**
 * 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("-------------------------------");
}
```

**StaffMember.java**

```java
/**
 * 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.
    */
    public abstract double pay();
    
    toString() omitted...
```
```java
/**
 * Represents a staff member that works as a volunteer.
 * @author Java Foundations
 */

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;
    }
}
```

```java
/**
 * Represents a general paid employee.
 * @author Java Foundations
 */

class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double payRate;

    /**
     * Constructor: Sets up this employee with the specified information.
     */
    public Employee(String eName, String eAddress, String ePhone,
                     String socSecNumber, double rate) {
        super(eName, eAddress, ePhone);
        socialSecurityNumber = socSecNumber;
        payRate = rate;
    }

    /**
     * @return the pay rate for this employee.
     */
    public double pay() {
        return payRate;
    }
}
```
```java
//**
* 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;
    }
}
```

```java
//**
* 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;
    }
```