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.

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 an `Rect` object, or to any object of **any compatible type**.

This **compatibility** can be established using **inheritance** or using **interfaces**.

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?

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

---

Exploring the benefits and flexibility of polymorphism

```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();
  }
}
```

---

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

public class Staff {
  private StaffMember[] stafflist;

  public Staff () {
    stafflist = new StaffMember[0];

    stafflist[0] = new Executive("Tony", "123 Main Line", "555-0400", "555-45-6789", 2423.07);
    stafflist[3] = new Hourly("Michael", "678 Fifth Ave.", "555-8888", 9584-7865, 16.55);
    stafflist[4] = new Volunteer("Adriana", "847 Robe Blvd.", "553-8374");
    stafflist[5] = new Volunteer("Benny", "$21 Old Lane", "555-7282");

    ((Executive)stafflist[0]).awardBonus (300.00);
    ((Hourly)stafflist[3]).addHours (40);
  }
```
public void payday() {
    double amount;
    for (int count=0; count < staffList.length; count++) {
        System.out.println(staffList[count].manufacturer);
        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.
* @author Java Foundations
*/
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...

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

/*
* Represents a general paid employee.
* @author Java Foundations
*/
public class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double payRate;
}

/*
* Constructor: Sets up this employee with the specified information.
* @author Java Foundations
*/
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;
}
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;
    }
}

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;
    }