CS244 Advanced Programming Applications

Dr Walid M. Aly

Lecture 4

Polymorphism

Group home page:

http://groups.yahoo.com/group/JAVA-CS244/

Polymorphism

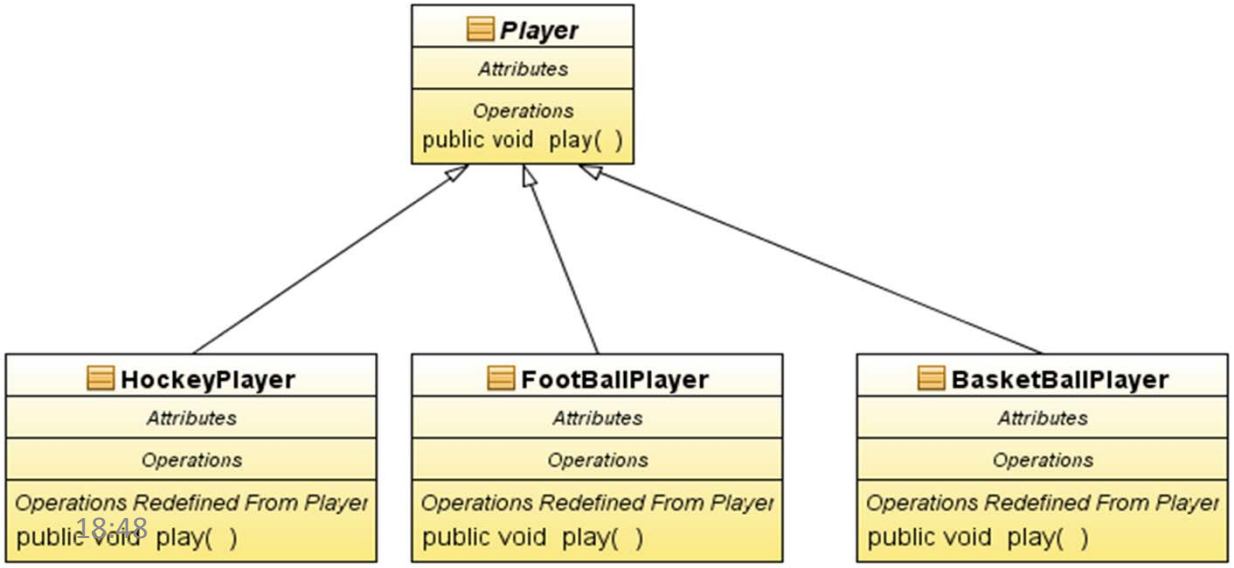
- **polymorphism**: The ability for the same code to behave differently depending on the type of argument used.

class Person
open(Window w) open (Present p) open(BankAccount ba)

Method Overloading



- **polymorphism**: The ability for the same code to be used with several different types of objects and behave differently depending on the type of object used.



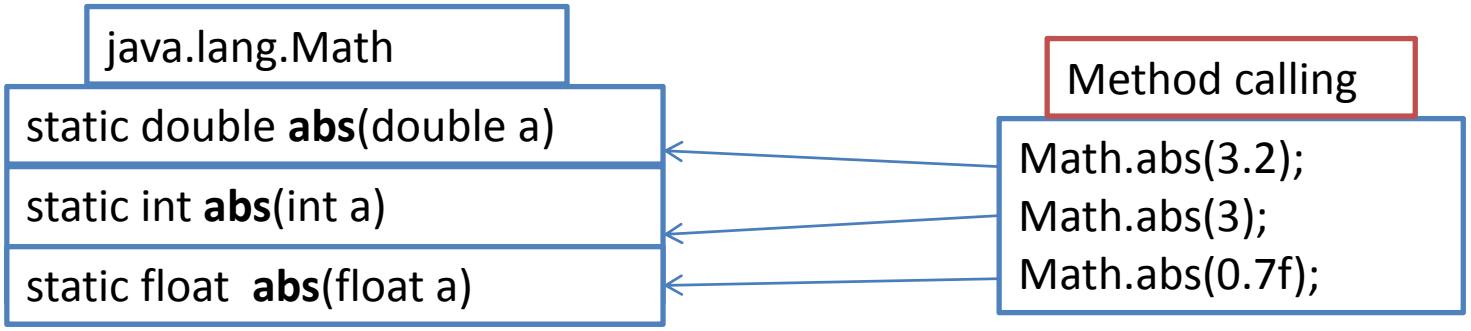
Method Overriding

+

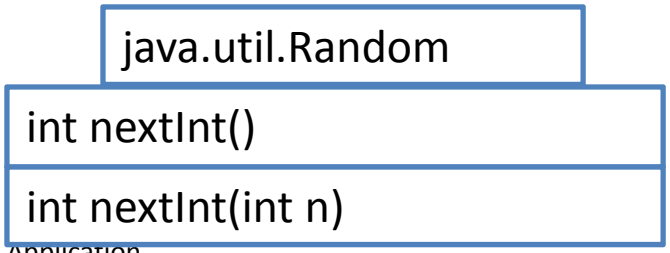
Dynamic method dispatch

Polymorphism with Method Overloading

Method overloading : define two or more methods within the same class that share the same name, as long as their parameter declarations are different.



- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- overloaded methods must differ in the type and/or number of their parameters.
- The return type alone is insufficient to distinguish two versions of a method.



Method Overriding

- when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass *overrides the method in the superclass.*
- *When an overridden method is called by an object from a* subclass, it refers to the version of that method defined by the subclass.
- **By overriding the subclass redefines a the behavior of a method.**

```
class Employee{
public double getSallary(){
return 1000;}}
```

```
class Manager extends Employee{
public double getSallary(){
return 1500;
}}
```

Overriding Rules

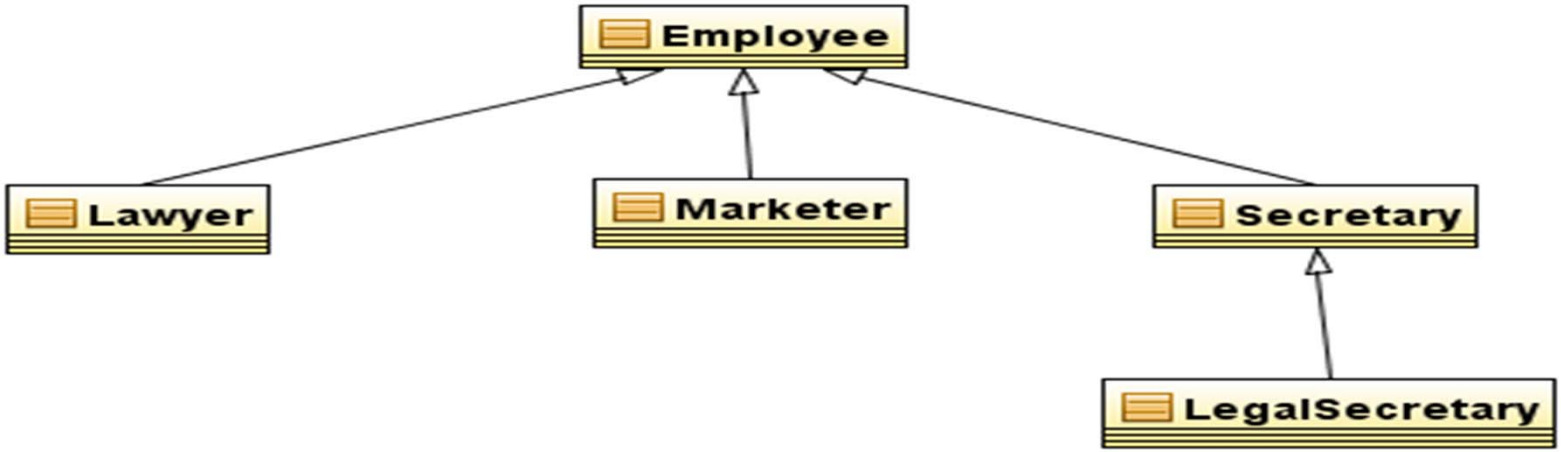
1. The two methods must have :same name-same arguments same return type.
2. The access modifier of an overriding must provide at least as much access as the overridden method,.
3. Static /non static methods can only be overridden by static /non static methods,

```
public class Test{
public static void main(String [] arg){
Employee  emp=new Employee();
Manager  manager=new Manager();
double salary1=emp.getSallary();//1000
double salary2=manager.getSallary();//1500
}}
```

The keyword super can be used at the subclass to point to members of superclass

```
class Manager extends Employee{
public double getSallary(){
return super.getSalary()+500;
}}
```

Case Study :Law Firm



- Consider the following employee regulations:
  - Employees make a salary of L.E.40,000 per year, except legal secretaries who make L.E5,000 extra per year (L.E45,000 total), and marketers who make L.E 10,000 extra per year (L.E50,000 total).
  - Employees have 10 days of paid vacation leave per year, except lawyers who get an extra 5 days.

```
class Employee{
private double salary;
private int  vacationDays;
  public Employee(){
  vacationDays=10;
  sallary=40_000;
     }
public double getSalary(){
return salary;
}
public double getVacationDays(){
return vacationDays;
}
}
```

```
class Lawyer extends Employee
{
public double getVacationDays()
{
return super.getVacationDays()+5;
}
.....
}
```

```
class Marketer extends Employee
{
public double getSalary()
{
return super.getSalary()+10000;
}
.....

}
```

```
class Secretary  extends Employee
{
public void takeDectattion(){}

}
```

```
class LegalSecretary  extends Secretary {
public double getSalary()
{
return super.getSalary()+5000;
}
.....
}
```

Class Object

`java.lang.Object`

`public class` **`Object`**
Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass.
All objects, including arrays, implement the methods of this class.

Constructor Summary	
Object()	

Method Summary
protected Object clone() Creates and returns a copy of this object.
public boolean equals(Object obj) : Indicates whether some other object is "equal to" this one.
public String toString() Returns a string representation of the object.

object.toString()

Method toString returns the Hash code for the object, this method is called automatically by System.out.println


```
class A{
int i;
int j;
A(int i,int j)
{
this.i=i;
this.j=j;
}
}
```

```
class Test
{
public static void main (String[] arg){
A a1=new A(3,4);
System.out.println(a1.toString());
}}
```

N.B.: printing an object using System.out.println(object) or print(object) means automatically System.out.println(object.toString()) & print(object.toString())

Overriding object.toString()

```
class A
{
int i;
int j;
A(int i,int j)
{
this.i=i;
this.j=j;
}
public String   toString()
{
String s="i="+i+"  j="+j;
return s;
}

}
```

```
class Test
{
public static void main (String[] arg){
A a1=new A(3,4);
System.out.println(a1);
}}
```

i=3 j=4

polymorphic references

Rule 1

A reference variable of a type class T can legally refer to an object of T **or** any subclass of T.

```
class T
{

}
```

```
class K extends T
{

}
```

T t1=new K();

Rule 2

A reference variable of a type interfaces T can legally refer to an object of classes implementing interface T

```
interface T
{

}
```

```
class K implements T
{

}
```

T t1=new K();

Dynamic method dispatch

Java determines which version of that method to execute based upon the **real** type of the object being referred to at the time the call occurs.

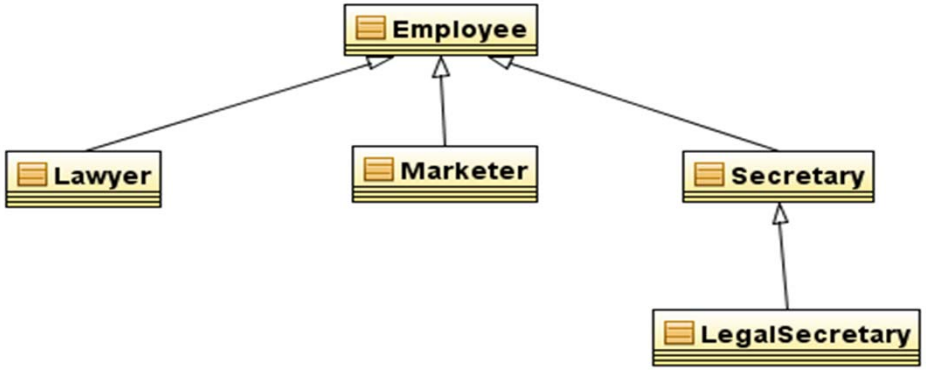
```
class A {
void callme() {
System.out.println("Inside A's callme method");
}
}
class B extends A {
void callme() {
System.out.println("Inside B's callme method");
}
}
```

```
class Test {
public static void main(String args[]) {
A a = new A(); // object of type A
A b = new B(); // object of type B
a.callme(); // calls A's version of callme
b.callme(); // calls B's version of callme
}
}
```

```
Inside A's callme method
Inside B's callme method
```

Dynamic method dispatch : Example from the case study

```
Employee person = new Lawyer();
int days=person.getVacationDays();// days =15
```



Objec t type : class Lawyer

Reference type : class Employee

- The reference **person** of type **Employee** pointing to an object of type **Lawyer** will act as a lawyer , **and call getVacationDays() of class Lawyer not Employee**
- You can call any methods from `Employee` on the variable person, but not any methods specific to `Lawyer` (such as `sue`).
- person.sue() // compile error

Benefits of polymorphism(1)

➢A method declaring in its argument list a super class data type can receive an object of any of its subclasses.

polymorphism

```
class A
{
int i=9;
void m1(){}
}
```

?

```
class B extends A
{
int j=10;
void m1(){}
void m2(){}
}
```

?

```
class C extends A
{
void m1(){}
void m2(){}
}
```

?

```
class D{
public void m(A a){
//////
a.m1();
/////
}
}
```

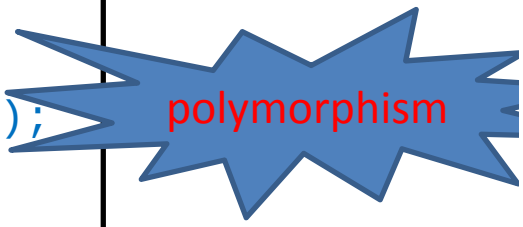
```
Class Test{
public static void main (String [] arg){
D d=new D();
A  a1=new A();
B b1=new B();
C c1=new C();
d.m(a1);
d.m(b1);
d.m(c1);
}
}
```

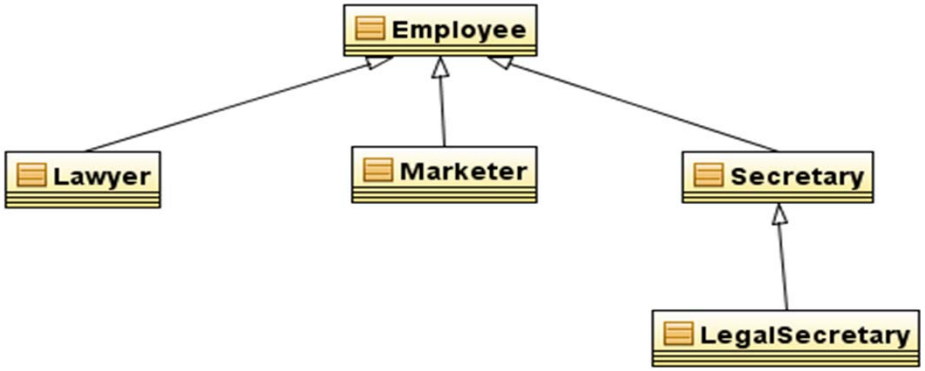
N.B. : Method m can only access the member variables of class A

Benefits of polymorphism(1)

➢A method declaring in its argument list a super class data type can receive an object of any of its subclasses.

```
public class EmployeeMain {
public static void printInfo(Employee empl) {
System.out.println("salary = " + empl.getSalary());
System.out.println("days = " + empl.getVacationDays());
System.out.println();
}
public static void main(String[] args) {
Lawyer lisa = new Lawyer();
Secretary steve = new Secretary();
printInfo(lisa);
printInfo(steve);
}
}
```





Benefits of polymorphism(2)

➢A method declaring in its argument list an interface data type can receive an object of any class implementing the interface.

```
interface Y
{
int x=9;
void m1();
}
```

polymorphism

```
class A implements Y{
public void m1(){}
public void m2(){}
}
```

```
class B implements Y{
public void m1(){}
}
```

```
class C implements Y{
public void m1(){}
}
```

?

?

?

```
class D {
public void m2(Y  y )
{
……
y.m1();
}
}
```

```
Class Test{
public static void main (String [] arg){
D d=new D();
A  a1=new A();
B b1=new B();
Y c1=new C();
d.m2(a1);
d.m2(b1);
d.m2(c1);
}}
```

N.B. : reference y can only access the member of interface Y
Dynamic Method Dispatch is applied

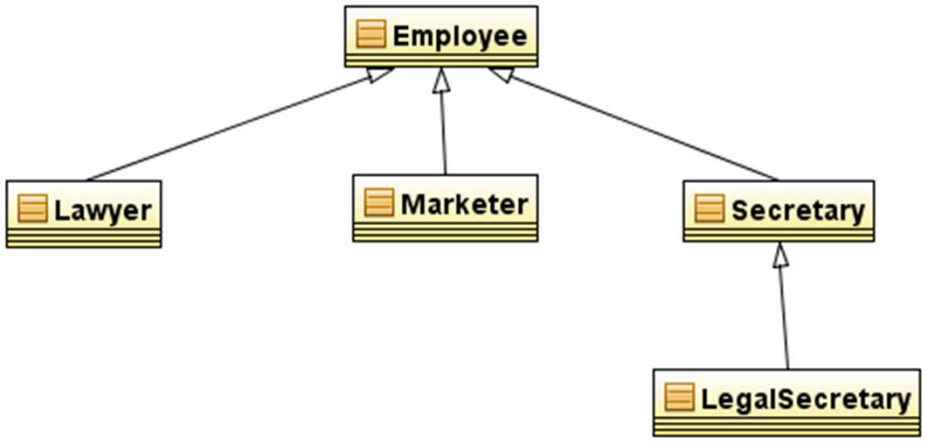
Benefits of polymorphism(3)

- You can declare arrays of superclass types, and store objects of any subtype as elements.

Example

```
public class Test {
public static void main(String[] args) {
Employee[] employees = {new Lawyer(), new Secretary(),
                        new Marketer(), new LegalSecretary()};
for (int i = 0; i < employees.length; i++) {
System.out.println("salary = " +employees[i].getSalary());
System.out.println("vacation days="+employees[i].getVacationDays());
System.out.println();
                }
            }
        }
```

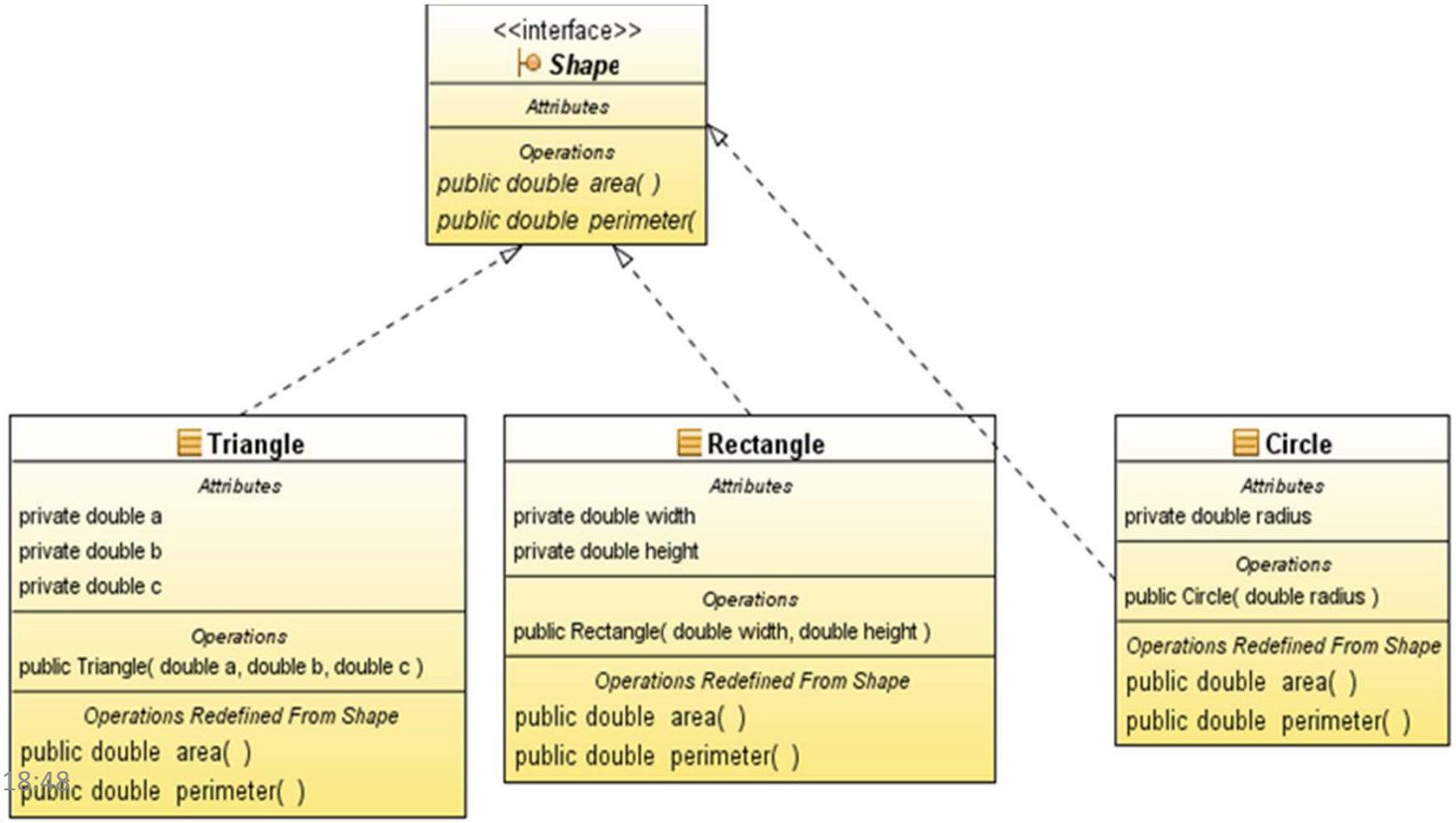
```
salary = 10000.0
vacation days=15.0

salary = 10000.0
vacation days=10.0

salary = 20000.0
vacation days=10.0

salary = 15000.0
vacation days=10.0
```



Benefits of polymorphism(4)

We can create an array of an interface type, and store any object implementing that interface as an element.

```
Circle circ = new Circle(12.0);
Rectangle rect = new Rectangle(4, 7);
Triangle tri = new Triangle(5, 12, 13);
Shape[] shapes = {circ, tri, rect};
```



Object Casting

Syntax:

To cast the reference a to be of type B
A a1=new B();
…….
…….
B b=(B)a1; //casting

Casting checked twice
At compilation : class B is sub class of A
(otherwise compile error)
At runtime: a1 points to an object of class B
(otherwise runtime error:classCastExeption)

Instanceof Operator

- Instanceof is a a boolean java operator that returns true or false
- Usage: ***reference*** *instanceof* ***class***
  returns true if the reference points to an object of this class or any of its subclasses
- Usage: ***reference*** *instanceof* ***interface***
  returns true if the reference points to an object of a class implementing the interface

```
class A
{
}
```

```
class B extends A
{
}
```

```
B b1=new B();
System.out.println(b1 instanceof B);//true
System.out.println(b1 instanceof A);//true
```

Example

```
public abstract class Shape{
public abstract double getArea();
}
```

```
public class Circle extends Shape{
private double radius;
public Circle( double radius)
{
this.radius=radius;
}
public double getRadius(){return radius;}
Public double getArea(){
return 3.14*radius*radius;}
}
```

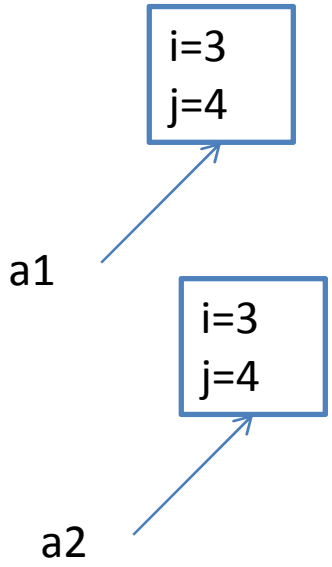
```
public class Rectangle extends Shape{
private double width;
private double height;
public Rectangle(double width,double height){
this .width=width;
this .height=height;
}
public double getWidth(){return width;}
public double getHeight(){return height;}
public double getArea()
{return width*height;}
}
```

```
public class Test {
public static void main(String[] args) {
Shape shape1 = new Circle(2);
Shape shape2 = new Rectangle(3, 4);
displayObject(shape1);
displayObject(shape2);
}
 public static void displayObject(Shape shape) {
System.out.println(“area=“+shape.getArea());

if (shape instanceof Circle) {
Circle c=(Circle)shape;
System.out.println("The circle radius is "
+c.getRadius());
}
else if (shape instanceof Rectangle) {
Rectangle rect=(Rectangle)shape;
System.out.println("The rectangle Width is "
+(rect.getWidth()));
System.out.println("The rectangle height is "
+(rect.getHeight()));
 }
 }
 }
```

object.equlas()

Method equals in class Object offers only a trivial comparison, as it compares whether the 2 reference for the 2 objects are equal and not the contents of the 2 objects.

```
public class A{
private int i;
private int j;
public  A(int i,int j)
{
this.i=i;
this.j=j;
}
}
```




```
class Test
{
public static void main (String[] arg){
A a1=new A(3,4);
A a2=new A(3,4);
System.out.println(a1.equals(a2));
}}
```

false

Overriding object.equlas()

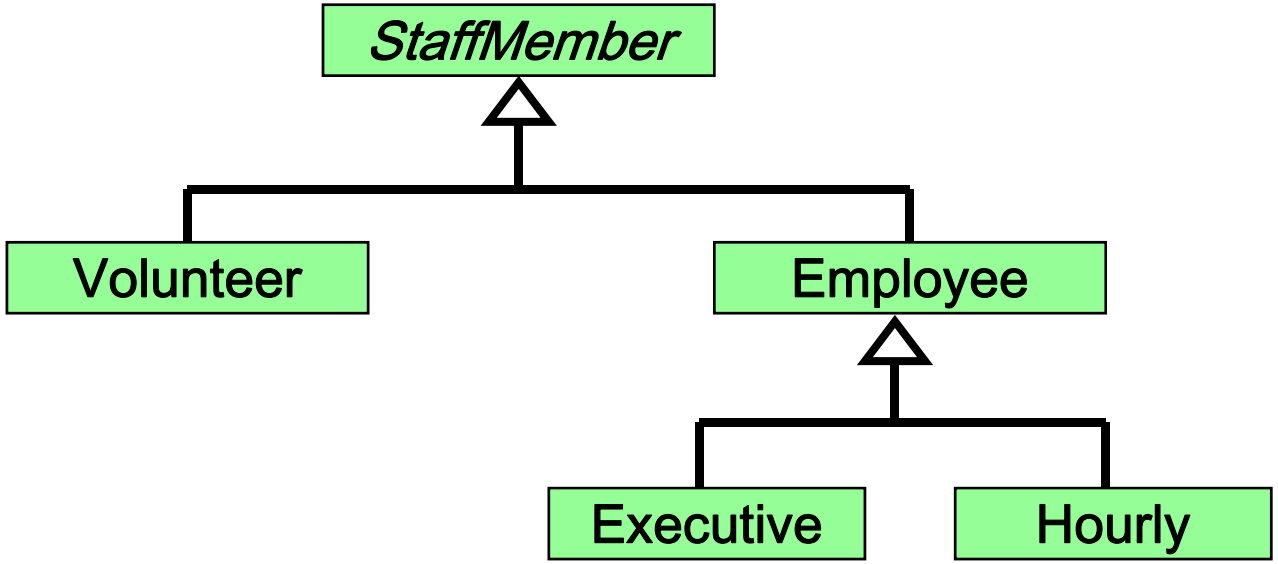
```
class A
{
int i;
int j;
A(int i,int j)
{
this.i=i;
this.j=j;
}

public boolean equals(Object obj)
{
A a2=(A)obj;
if ( (i==a2.i)&&(j==a2.j) )
return true;
else
return false;
}}
```

```
class Test
{
public static void main (String[] arg)
{
A a1=new A(3,4);
A a2=new A(3,4);
System.out.println(a1.equals(a2));
}
}
```

true

Case Study: Simple Payroll System

Consider the class hierarchy shown in Figure. The classes in it represent various types of employees that might be employed at a particular company..



```
//********************************************************************
//  StaffMember.java       Author: Lewis/Loftus
//
//  Represents a generic staff member.
//********************************************************************

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;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns a string including the basic employee information.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = "Name: " + name + "\n";

      result += "Address: " + address + "\n";
      result += "Phone: " + phone;

      return result;
   }

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

```
//********************************************************************
//  Volunteer.java       Author: Lewis/Loftus
//
//  Represents a staff member that works as a volunteer.
//********************************************************************

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);
   }

   //-----------------------------------------------------------------
   //  Returns a zero pay value for this volunteer.
   //-----------------------------------------------------------------
   public double pay()
   {
      return 0.0;
   }
}
```

```
//********************************************************************
//  Employee.java       Author: Lewis/Loftus
//
//  Represents a general paid employee.
//********************************************************************

public 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;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns information about an employee as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = super.toString();

      result += "\nSocial Security Number: " + socialSecurityNumber;

      return result;
   }

   //-----------------------------------------------------------------
   //  Returns the pay rate for this employee.
   //-----------------------------------------------------------------
   public double pay()
   {
      return payRate;
   }
}
```

```
//********************************************************************
//  Executive.java       Author: Lewis/Loftus
//
//  Represents an executive staff member, who can earn a bonus.
//********************************************************************

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
   }
```

continue

continue

```
    //-----------------------------------------------------------------
    //  Awards the specified bonus to this executive.
    //-----------------------------------------------------------------
    public void awardBonus (double execBonus)
    {
       bonus = execBonus;
    }

    //-----------------------------------------------------------------
    //  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;
    }
}
```

```
//*******************************************************************
//  Hourly.java       Author: Lewis/Loftus
//
//  Represents an employee that gets paid by the hour.
//*******************************************************************

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;
   }
```

continue

continue

```
    //-----------------------------------------------------------------
    //  Adds the specified number of hours to this employee's
    //  accumulated hours.
    //-----------------------------------------------------------------
    public void addHours (int moreHours)
    {
       hoursWorked += moreHours;
    }

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

       hoursWorked = 0;

       return payment;
    }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns information about this hourly employee as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = super.toString();

      result += "\nCurrent hours: " + hoursWorked;

      return result;
   }
}
```

```
//********************************************************************
//  Staff.java       Author: Lewis/Loftus
//
//  Represents the personnel staff of a particular business.
//********************************************************************

public class Staff
{
   private StaffMember[] staffList;

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

continue

continue

```
        staffList[0] = new Executive ("Sam", "123 Main Line",
           "555-0469", "123-45-6789", 2423.07);

        staffList[1] = new Employee ("Carla", "456 Off Line",
           "555-0101", "987-65-4321", 1246.15);
        staffList[2] = new Employee ("Woody", "789 Off Rocker",
           "555-0000", "010-20-3040", 1169.23);

        staffList[3] = new Hourly ("Diane", "678 Fifth Ave.",
           "555-0690", "958-47-3625", 10.55);

        staffList[4] = new Volunteer ("Norm", "987 Suds Blvd.",
           "555-8374");
        staffList[5] = new Volunteer ("Cliff", "321 Duds Lane",
           "555-7282");

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

        ((Hourly)staffList[3]).addHours (40);
    }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Pays all staff members.
   //-----------------------------------------------------------------
   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 ("-----------------------------------");
      }
   }
}
```

```
//********************************************************************
//  Firm.java       Author: Lewis/Loftus
//
//  Demonstrates polymorphism via inheritance.
//********************************************************************

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();
   }
}
```

Class Firm is known as the driver class as it contains the main method,

The Firm class shown contains a main driver that creates a Staff of employees and invokes the payday method to pay them all. The program output includes information about each employee and how much each is paid (if anything).

```
//********************************************************************
//  Firm.java       Author: Lewis/Loftus
//
//  Demonstrates polymorphism via inheritance.
//********************************************************************

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();
   }
}
```

Output

```
Name: Sam
Address: 123 Main Line
Phone: 555-0469
Social Security Number: 123-45-6789
Paid: 2923.07
-----------------------------------
Name: Carla
Address: 456 Off Line
Phone: 555-0101
Social Security Number: 987-65-4321
Paid: 1246.15
-----------------------------------
Name: Woody
Address: 789 Off Rocker
Phone: 555-0000
Social Security Number: 010-20-3040
Paid: 1169.23
-----------------------------------
```

Output (continued)

```
Name: Diane
Address: 678 Fifth Ave.
Phone: 555-0690
Social Security Number: 958-47-3625
Current hours: 40
Paid: 422.0
-----------------------------------
Name: Norm
Address: 987 Suds Blvd.
Phone: 555-8374
Thanks!
-----------------------------------
Name: Cliff
Address: 321 Duds Lane
Phone: 555-7282
Thanks!
-----------------------------------
```