


Polymorphism



What Is Polymorphism?

Polymorphism allows a single variable to refer to objects from different subclasses in the same inheritance hierarchy

- `B o = new A(); // Will compile iif :`
 - A is a subclass of B
 - A is a concrete Class
- For example, if Circle and Rectangle are subclasses of Shape, then the following statements are valid:
- `Shape s1,s2;`
- `Rectangle r = new Rectangle(5,2);`
- `Circle c = new Circle(3);`
- `s1 = r; s2 = c;`
- `Shape s3 = new Rectangle(6,4);`

Application Design Using Polymorphism

- There are three design steps required for polymorphism:
 - create representative base classes:
 - could create an **Employee** class to represent the general behavior of the Employee class family.
 - create specialized subclasses:
 - Implement the specialization specific to this class by **overriding** the required methods. classes such as **Manager** and **Contractor** could be created for the Employee class family.

Application Design Using Polymorphism

- use the base class type as reference variables:
 - use variables of type Employee to operate on objects of type Employee, Manager, or Contractor. This can be justified by recalling that a Manager object has all the attributes, both member variables and methods, of the parent class Employee.
 - any operation that is legitimate on an Employee object is also legitimate on a Manager object. If the Employee object has methods raiseSalary and fire, then the Manager class does also.

```
public class Employee
public class Manager extends Employee
public class Contractor extends Employee
```

- **We can maintain our class Employee using an array, combining objects from the Employee, Manager and Contractor classes.**

Creating the staff Array

- ```
Employee [] staff = new Employee[80];
staff[0] = new Manager();
staff[1] = new Employee(); // Employee is a concrete class
staff[2] = new Contractor();
// and so on
```
- Using the polymorphic features, we can then perform the following:
  - ```
// In the Employee class  
public TaxRate findTaxRate(Employee e) {  
    double sal = e.getSalary();  
    // e could actually be a Manager object  
    if (sal > x) { .... }
```

Using the instanceof Operator

```
public void static main (String[] args) {  
    int empCount, mgrCount, contCount;  
    empCount = mgrCount= contCount = 0;  
  
    Employee staff[] = new Employee[80];  
    ...  
  
    for (int i=0; i < staff.length; i++ ) {  
  
        if (staff[i] instanceof Manager)  
            mgrCount++;  
        else if (staff[i] instanceof Contractor)  
            contCount++;  
        else  
            empCount++;  
  
    }  
    ...  
}
```

Casting an Object to its Subclass Form

- Consider the following line of code:

```
Employee e = new Manager();
```

- Using the variable e as is, we can access only the parts of the object that are part of an Employee object;
- the parts specific to a Manager object are hidden.
- Access is limited because, as far as the compiler is concerned, the variable e is an Employee object, not a Manager object. Therefore, the following is not allowed:

```
e.department = "Finance";
```


- The workaround for this would be to use a variable of type Manager and assign the object to it by casting:

```
Manager m = (Manager) e; // cast
m.department = "Finance";
```

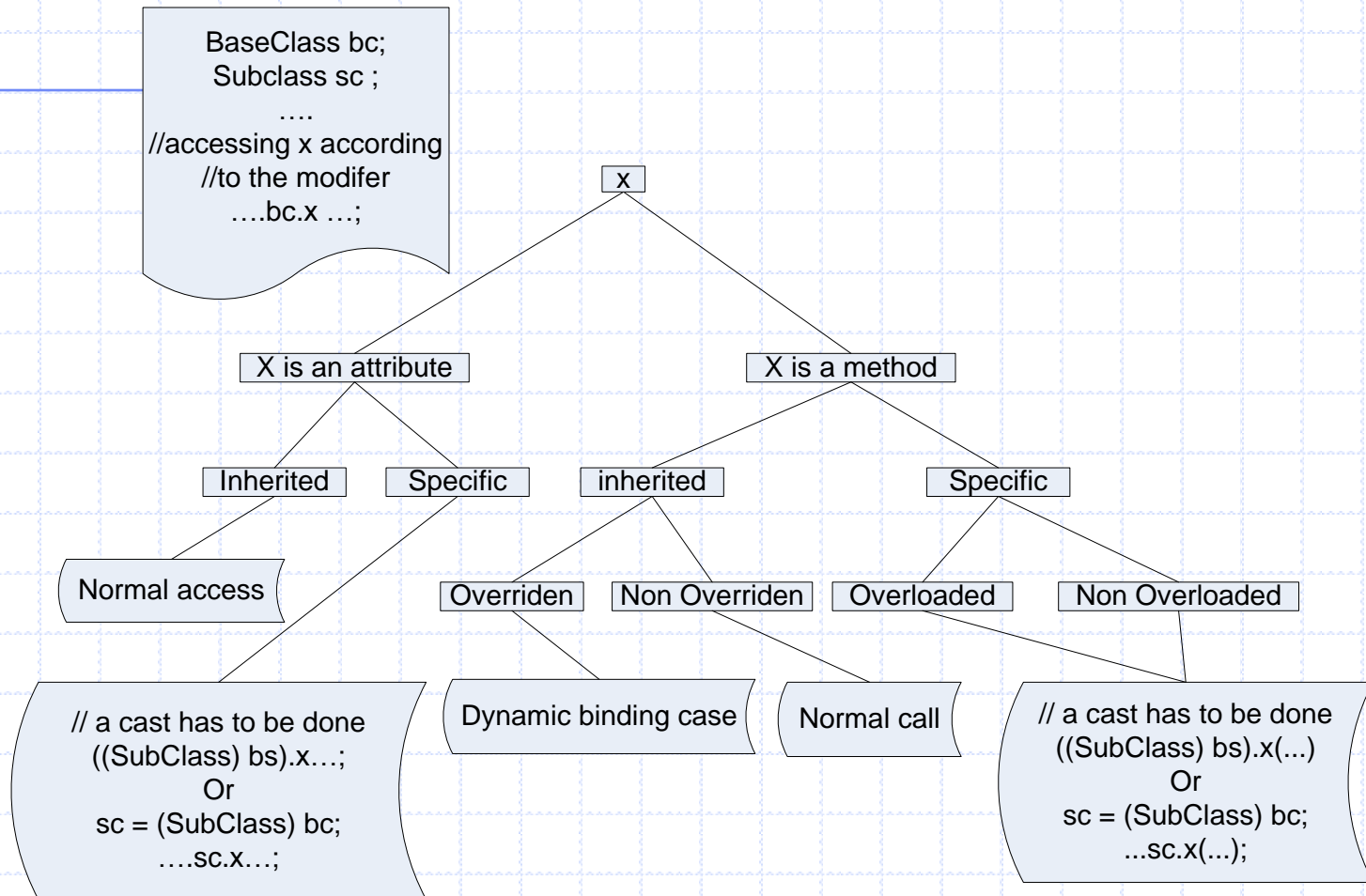
- Or we could cast with just one line of code:

```
((Manager) e).department = "Finance";
```
- The cast will fail at runtime if e does not refer to a Manager object.
- Before performing the cast, we should use the **instanceof** operator to test if e is referring to a Manager object.

Casting an Object to its Subclass Form

```
public class Employee {  
    public class Manager extends Employee {  
        private String name;  
        private String department;  
        ...  
        private String jobTitle;  
    }  
    ...  
}  
  
public void method(Employee e) {  
    if (e instanceof Manager) {  
        Manager m = (Manager) e;  
        System.out.println("Casting object as a manager");  
        m.department = "Finance";  
    }  
    // rest of operation  
}
```

Polymorphic cases



Interface

- An interface is a reference data type
- An interface includes only constants and methods headers (all public)
- An interface does not have instances
- An interface can inherit from other interfaces
- A class must implement an interface
- A class implements an interface if it provides the method body to all abstract methods defined in the interface

Example

```
public interface A {  
    public void x();  
    public double y();  
}  
  
public class B implements A {  
    public void x(){  
        //Implement the body of x  
    }  
    public double y(){  
        //Implements the body of y  
    }  
}
```

```

public interface Shape {
    public double getArea();
    public double getVolume();
    public String getName();
}

public class Point extends Object implements Shape {
    private int x;
    private int y;
    public Point() {
        // implicit call to Object constructor occurs here
    }
    public Point( int xValue, int yValue ) {
        x = xValue;
        y = yValue;
    }
    public void setX( int xValue ) {
        x = xValue;
    }
    ....
}

```