06_2 Polymorphism

Object-Oriented Programming

Automatic Type Conversion

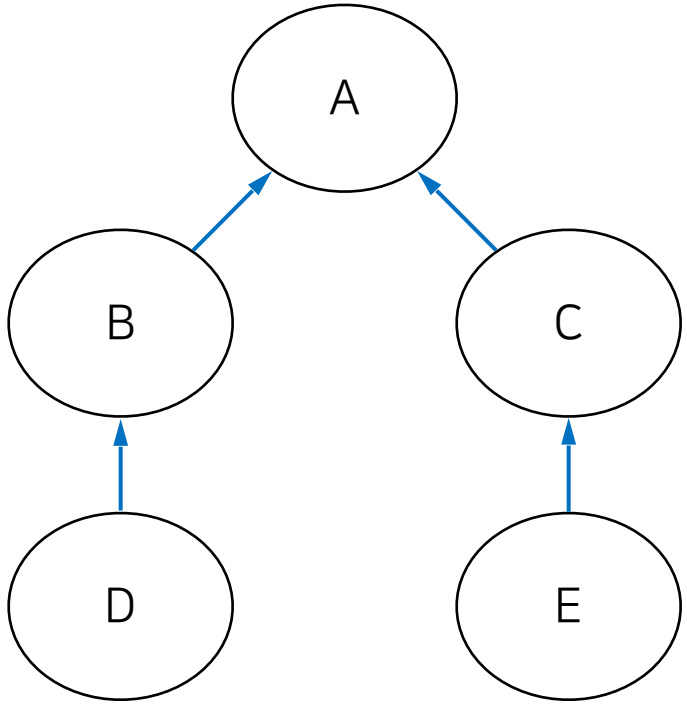
- The automatic type conversion of a Class:
  - Descendant to the Ancestor type.
  - **Upcasting**이라 부름

- Example)

```
class Animal { ... }
class Cat extends Animal { ... }

Cat cat = new Cat();
Animal animal1 = cat;
Animal animal2 = new Cat();
```

Inheritance Tree and Auto Type Conversion



class hierarchy
(child → parent)

```
B b = new B();
C c = new C();
D d = new D();
E e = new E();
```

```
A a1 = b; //ok
A a2 = c; //ok
A a3 = d; //ok
A a4 = e; //ok

B b1 = d; //ok
C c1 = e; //ok

B b3 = e; //error
C c2 = d; //error
```

Polymorphism in Method Call

```
class Parent {
    void method1() { ... }
    void method2() { ... }
}
```

```
class Child extends Parent {
    @override
    void method2() { ... }
    void method3() { ... }
}
```

```
class ParentChildDemo {
    public static void main(String[] args) {
        Child child = new Child();
        Parent parent = child;

        parent.method1();

        // Child's overridden method2()
        parent.method2(); // by Polymorphism

        // ERROR: no method3() in Parent
        parent.method3();
    }
}
```

Polymorphism

- In the previous example, when a parent class object behaves as if it were a child class object in some situations, this is called "polymorphism (다형성)".
- In other words, when a child class object is assigned to the parent class object, the parent object will call the overridden method in the child class.
- Polymorphism is implemented by Dynamic Binding (Late Binding).
- This means that we don't decide which 'method2' to run at compile time, but we decide which method2 to run at runtime.

- Advantages of Polymorphism
  - Flexibility in code: the same code can work for many different objects
  - Maintainability: New classes can be added without changing the code
  - Extensibility: Can extend functionality without code modifications

Example: AnimalCoversionTest.java

```
public class AnimalConversionTest {

    public static void main(String[] args) {
        Dog dog = new Dog("Jane", 8, "bulldog");
        Cat cat = new Cat("Kitti", 5, "white");

        Animal animal = new Animal("Tom", 3);
        animal.makeSound();  // original Animal's makeSound()
                             // “Some generic animal sound"
        animal = dog;
        animal.makeSound();  // dog's overridden makeSound()
                             // “Bark"
        animal = cat;
        animal.makeSound();  // cat's overridden makeSound()
                             // “Meow”
    }
}
```

Polymorphism in Parameters

- Take a parent (ancestor) class type as a parameter in a method
- Then, any child (descendant) class of the parameter type can be passed as the parameter
- Ex)

```
class Parent { }
class Child extends Parent { }
...
Child c = new Child();
Parent p = new Parent();
...
void someMethod(Parent p) { }
...
someMethod(p);
someMethod(c);  // OK! because of the polymorphism
```

Example) ShapeDemo (1/2)

```
class Shape {
    void draw() {
        System.out.println("Drawing Something");
    }
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a Circle");
    }
}

class Rectangle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a Rectangle");
    }
}
```

Example) ShapeDemo (2/2)

```
class ShapeDrawer {
    // Use polymorphism to take a Shape (Parent, Ancestor) type as a parameter
    public void drawShape(Shape shape) {
        shape.draw(); // At runtime, appropriate class' draw method is called
    }
}

public class ShapeDemo {
    public static void main(String[] args) {
        ShapeDrawer shapeDrawer = new ShapeDrawer();

        Shape myShape = new Shape();
        Shape myCircle = new Circle();
        Shape myRectangle = new Rectangle();

        // Using polymorphism for method parameter
        shapeDrawer.drawShape(myShape);  // Output: Drawing Something
        shapeDrawer.drawShape(myCircle); // Output: Drawing a Circle
        shapeDrawer.drawShape(myRectangle); // Output: Drawing a Rectangle
    }
}
```

Upcasting vs Downcasting

- Upcasting
  - Automatic type conversion from descendant class to ancestor class
  - No explicit casting
  - ex) Child c = new Child();
    Parent p = c; // upcasting
- Downcasting
  - Type conversion from ancestor class to descendant class
  - Explicit casting needed
  - ex) Parent p1 = new Child(); // upcasting
    Child c = (Child) p1; // downcasting OK
    Parent p2 = new Parent();
    Child c = (Child) p2; // ERROR!! Runtime error

Example) EmployeeDemo (1/3)

```
class Employee {
    void work() {
        System.out.println("Employee is working");
    }
}

class Manager extends Employee {
    @Override
    void work() {
        System.out.println("Manager is managing");
    }

    void plan() {
        System.out.println("Manager is planning");
    }
}
```

Example) EmployeeDemo (2/3)

```
class Engineer extends Employee {
    @Override
    void work() {
        System.out.println("Engineer is engineering");
    }
    void design() {
        System.out.println("Engineer is designing");
    }
}

public class EmployeeDemo {
    public static void main(String[] args) {
        Employee employee1 = new Manager(); // Upcasting
        employee1.work(); // Output: Manager is managing

        if (employee1 instanceof Manager) { // original class = Manager?
            Manager manager = (Manager) employee1; // Downcasting
            manager.plan(); // Output: Manager is planning
        }
```

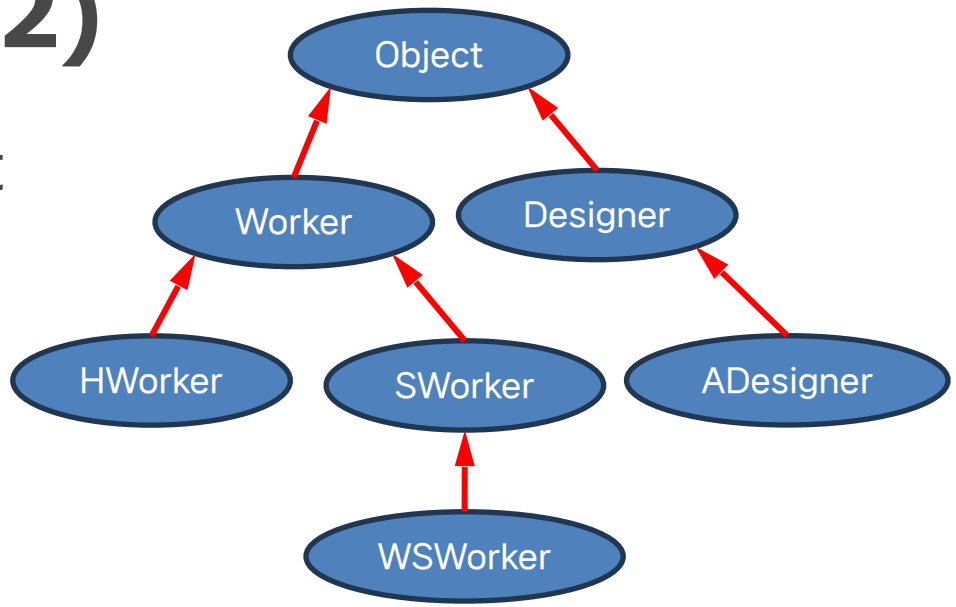
Example) EmployeeDemo (3/3)

```
        Employee employee2 = new Engineer(); // Upcasting
        employee2.work(); // Output: Engineer is engineering


        if (employee2 instanceof Engineer) { // original class = Engineer?
            Engineer engineer = (Engineer) employee2; // Downcasting
            engineer.design(); // Output: Engineer is designing
        }
    }
}
```

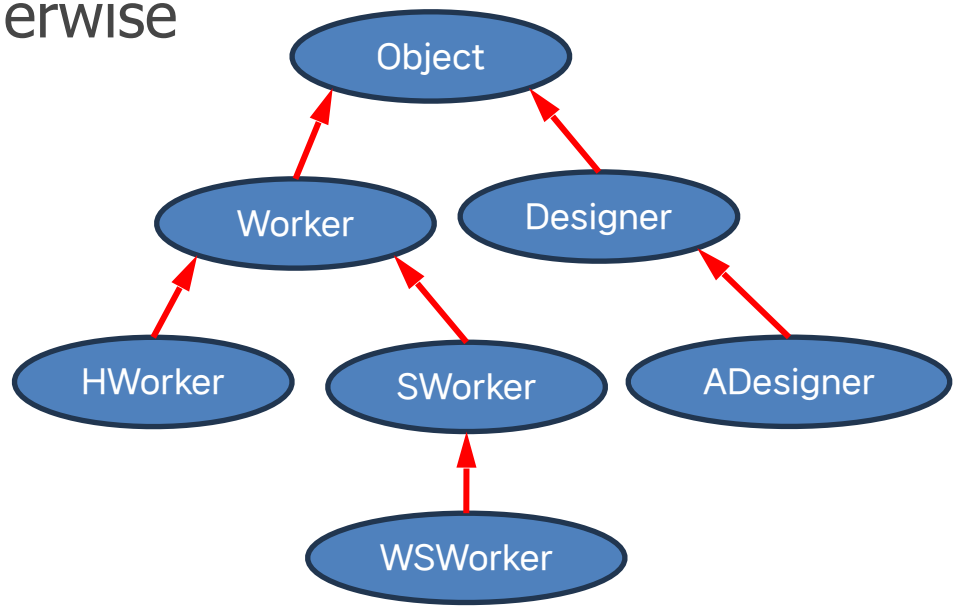
Instanceof and getClass() (1/2)

- Both can be used to check the class of an object
- instanceof
  - anObject instanceof SomeClass
    - returns true if anObject is of type SomeClass (or SomeClass's descendant)
  - ex) (other instanceof Worker) is true
    - when other is Worker, HWorker, SWorker, WSWorker



Instanceof and getClass() (2/2)

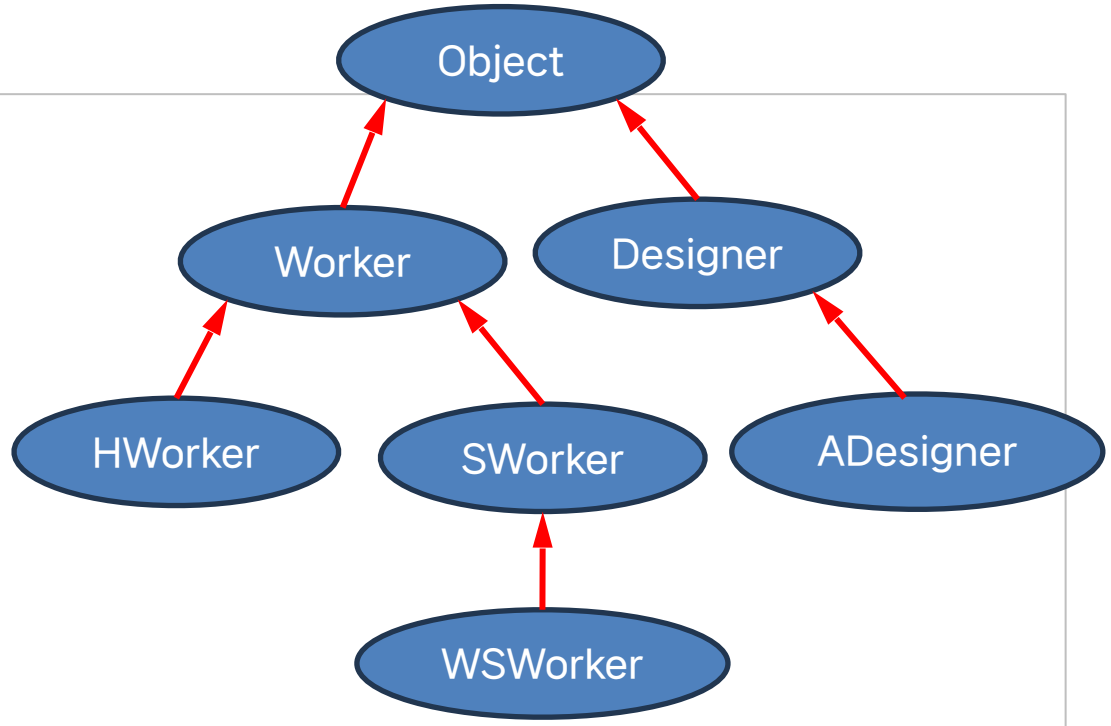
- getClass()
  - true when both classes are the same, false otherwise
  - ex) Designer d = new ADesigner();
    ADesigner ad = new ADesigner();
    if (d.getClass() == ad.getClass()) { }



Example (1/2)

```
class Worker { }
class Designer { }
class HWorker extends Worker { }
class SWorker extends Worker { }
class WSWorker extends SWorker { }
class ADesigner extends Designer { }


public class TestGetClass {
    public static void main(String[] args) {
        Worker worker = new Worker();
        HWorker hworker = new HWorker();
        SWorker sworker1 = new SWorker();
        SWorker sworker2 = new SWorker();
        WSWorker wsworker = new WSWorker();
        Designer designer = new Designer();
        ADesigner adesigner = new ADesigner();
```



Example (2/2)

```
        System.out.println(worker.getClass() == hworker.getClass()); // false
        System.out.println(sworker1.getClass() == sworker2.getClass()); // true
        System.out.println(sworker1.getClass() == wsworker.getClass()); // false
        //The following is ERROR! No ancestor-descendant relationship
        //System.out.println(hworker.getClass() == designer.getClass());


        System.out.println(wsworker instanceof Worker); // true
        System.out.println(sworker1 instanceof WSWorker); // false
        System.out.println(designer instanceof Object); // true
        // The following is ERROR: No ancestor-descendant relationship
        // System.out.println(worker instanceof Designer);
    }
}
```

Summary: Automatic Type Conversion (1/2)

- Automatic Type Conversion이 가능한 경우
  - Numbers 에 속하는 type들 중에 범위가 작은 쪽에서 큰 쪽으로
    - ex) int x = 5;
      double y = x;
  - Descendant class object를 Ancestor class object로 (Upcasting)
    - ex) class Animal { }
      class Cat extends Animal { }
      Cat c = new Cat();
      Animal a = c;

Summary: Automatic Type Conversion (2/2)

- Automatic Type Conversion이 불가능한 경우
  - Numbers type들 중 큰 범위의 type을 작은 범위로 assign
    - ex) int x; double y = 123.51;
      x = y; // x = (int)y; 는 가능, explicit type conversion
  - Ancestor class type을 Descendant class type으로 (Down-casting)
    - ex) class Animal { } class Cat extends Animal { }
      Animal a = new Animal(); Cat c = a;
      Animal b = new Cat(); Cat c = (Cat)b; // explicit down-casting
  - 아무 관련 없는 두 type들 간의 conversion
    - ex) boolean b = false; int x; x = b; // x = (int)b; 도 불가