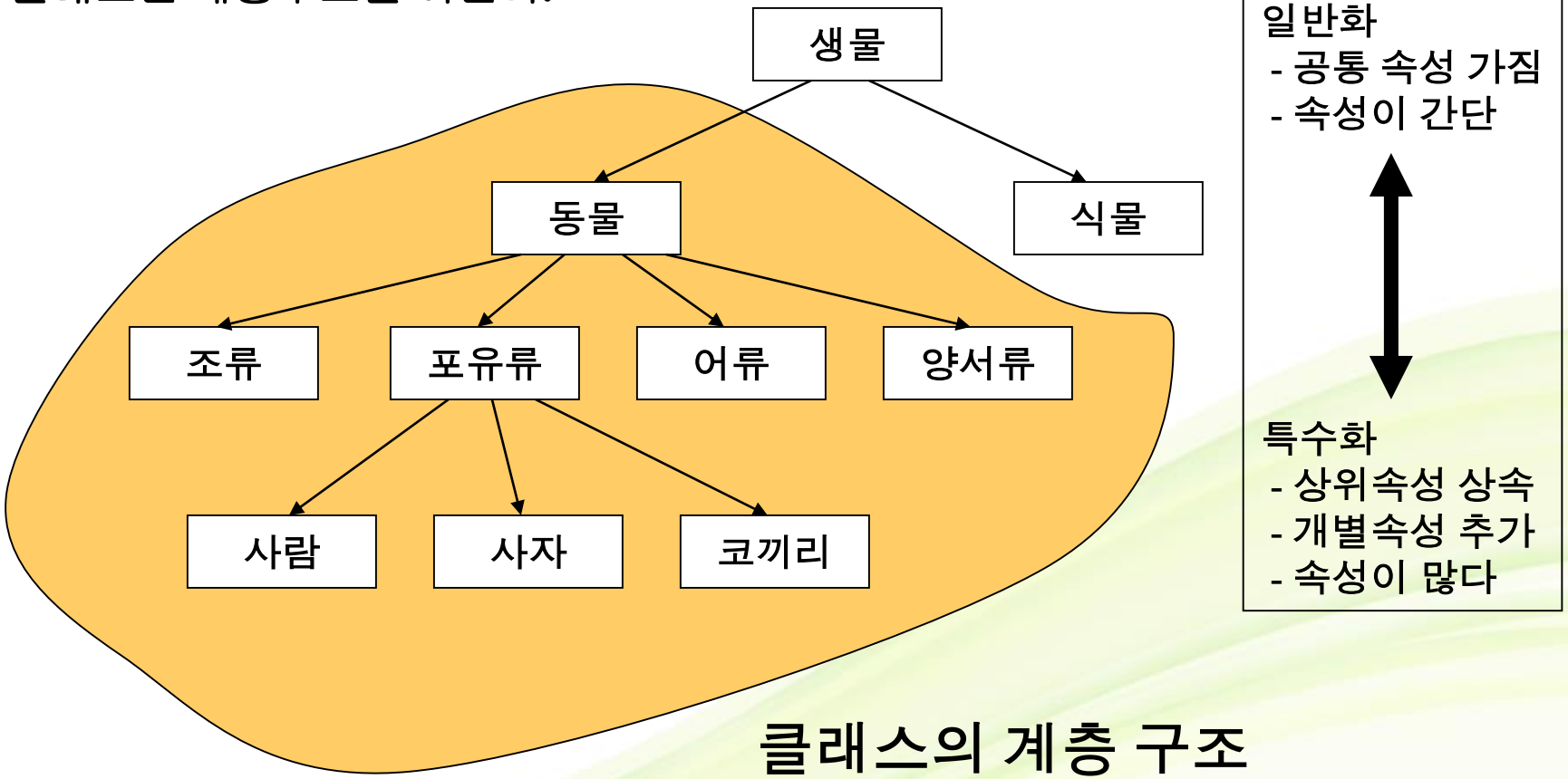
상속
Inner Class

강사 : 강병준

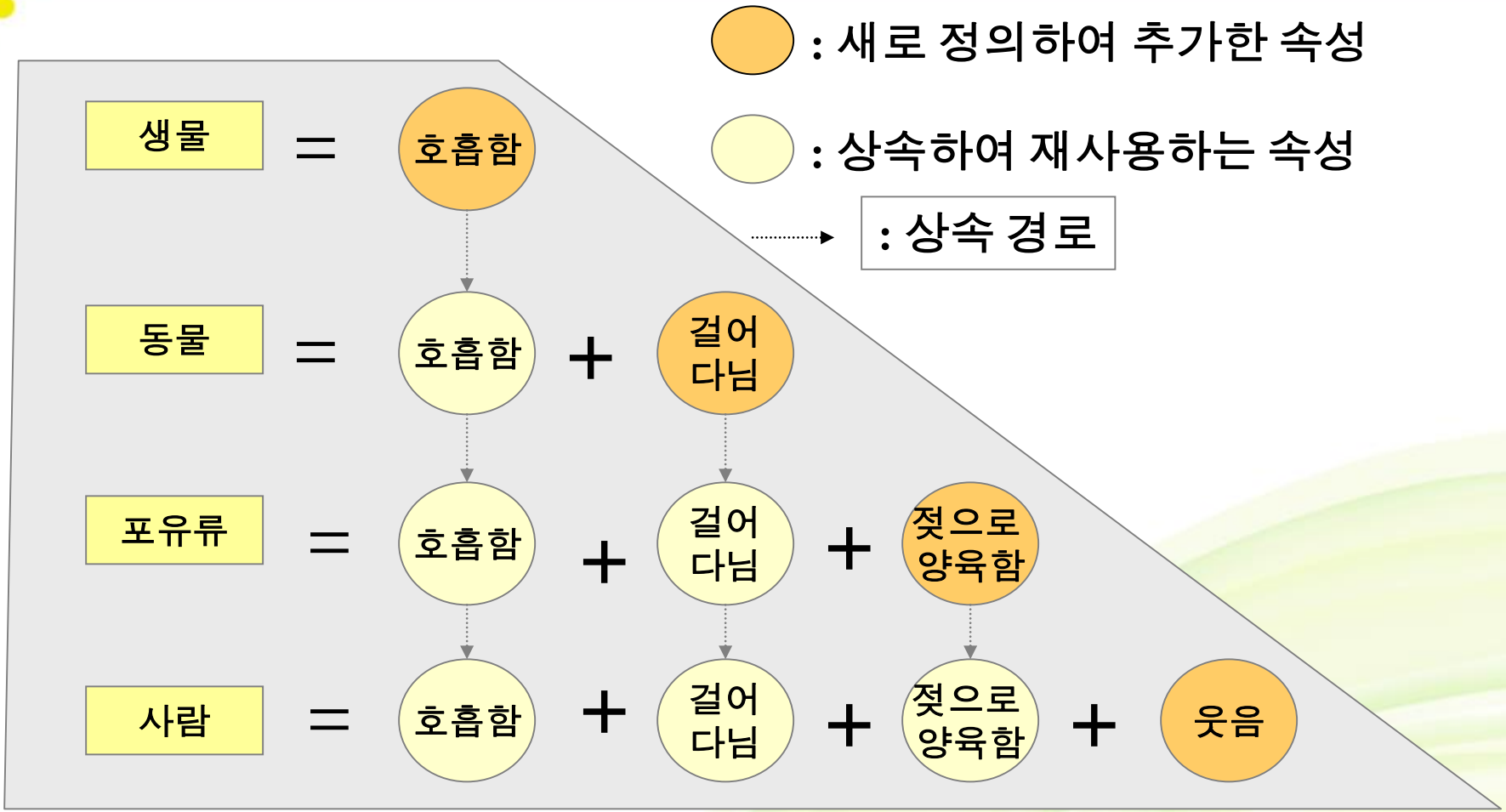
클래스의 계층구조

새로운 클래스를 생성할 때 기존 클래스의 하위 클래스로 선언할 수 있다.
새로운 클래스에 속성이나 메소드를 추가하여 기존 클래스를 확장할 수 있다.
클래스는 계층구조를 이룬다.



클래스의 계층 구조

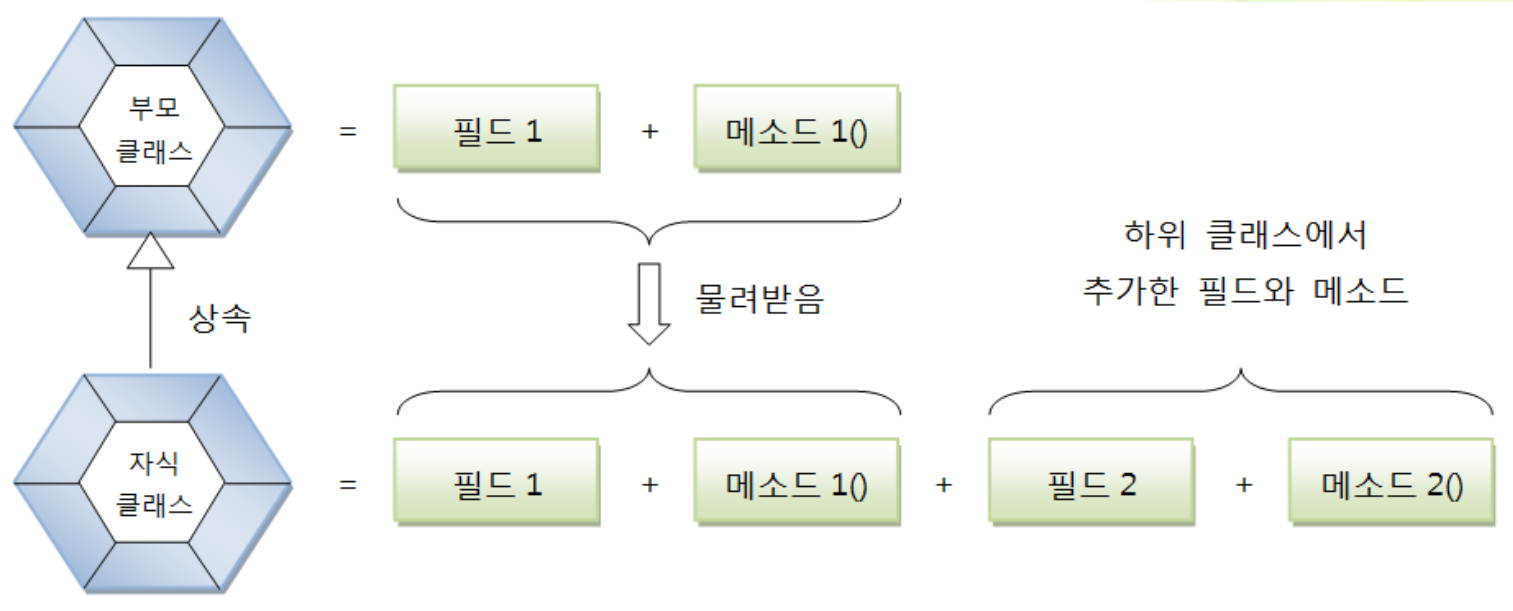
클래스계층구조에서의 상속관계



속성을 상속하여 새로운 클래스를 생성

상속 개념

- 상속(Inheritance)이란?
  - 현실 세계:
    - 부모가 자식에게 물려주는 행위
    - 부모가 자식을 선택해서 물려줌
  - 객체 지향 프로그램:
    - 자식(하위, 파생) 클래스가 부모(상위) 클래스의 멤버를 물려받는 것
    - 자식이 부모를 선택해 물려받음
    - 상속 대상: 부모의 필드와 메소드



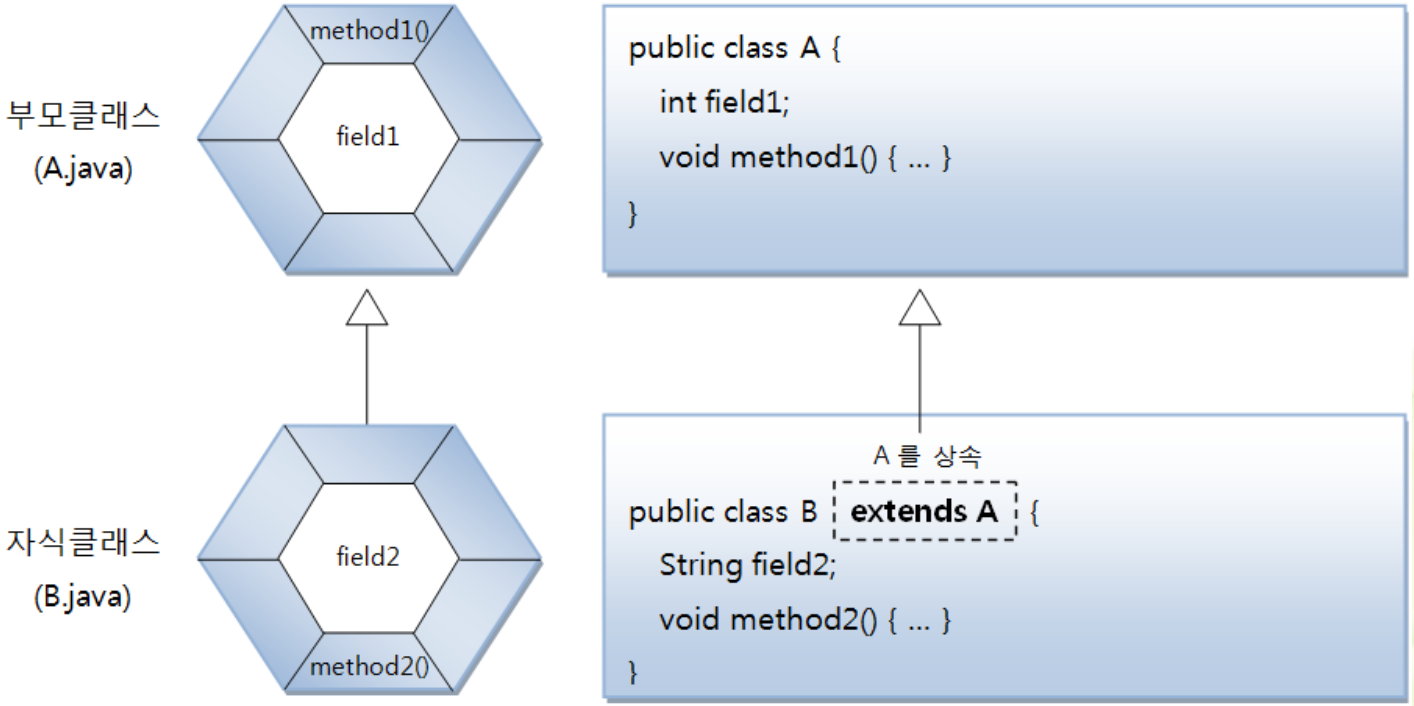
상속 개념

- 상속(Inheritance) 개념의 활용
  - 상속의 효과
    - 부모 클래스 재사용해 자식 클래스 빨리 개발 가능
    - 반복된 코드 중복 줄임
    - 유지 보수 편리성 제공
    - 객체 다형성 구현 가능
  - 상속 대상 제한
    - 부모 클래스의 private 접근 갖는 필드와 메소드 제외
    - 부모 클래스가 다른 패키지에 있을 경우, default 접근 갖는 필드와 메소드도 제외

클래스 상속(extends)

❖ extends 키워드

- 자식 클래스가 상속할 부모 클래스를 지정하는 키워드



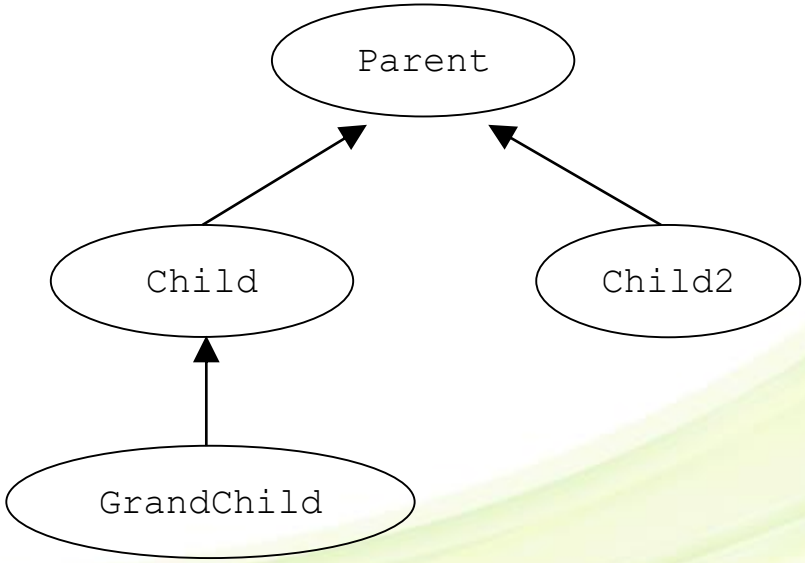

```
class 자식클래스 extends 부모클래스1, 부모클래스2 {
}
```

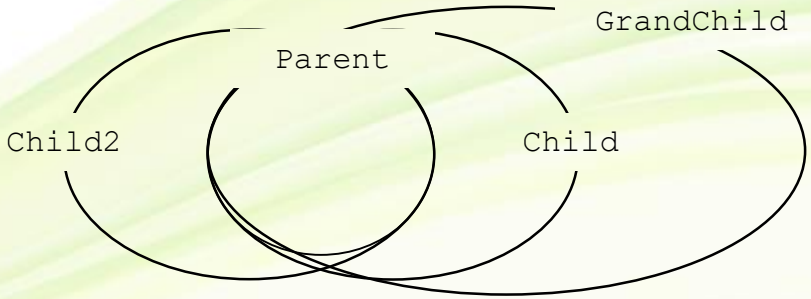
- 자바는 단일 상속 - 부모 클래스 나열 불가

클래스간의 관계 – 상속관계(inheritance)

- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리한다.
- 조상의 변경은 자손에 영향을 미치지만, 자손의 변경은 조상에 아무런 영향을 미치지 않는다.

```
class Parent {}
class Child extends Parent {}
class Child2 extends Parent {}
class GrandChild extends Child {}
```





상속의 선언

▶ 상속이 포함된 클래스 선언 형식

[접근제어자/final/abstract] class 클래스이름 **extends** 상위클래스이름 **{**

..... // 멤버변수선언

..... // 생성자

..... // 메소드선언

}

▶ 상속에서 제외되는것

☞ **private** 멤버들은 상속에서 제외된다.

☞ 생성자 함수도 상속에서 제외된다.

상속의 예(생성자 함수 상속에서 제외)

```
class A{
   int k=5;
   A(){
      System.out.println("A class");
   }
}
class B extends A{
   B(){
      System.out.println("B Class");
   }
   public static void main(String[] a){
      B b = new B();
      System.out.println("k=>"+b.k);
   }
}
```

상속 예제

```
class A{
        int a=1;
         public void a(){
                System.out.println("나 상속됐어...");
        }
}

class B extends A{
        int b=2;
         public void b(){
                System.out.println("나두 상속이 됐어...");
        }
}

class InhTest extends B{
        public static void main(String[] arg){
                InhTest t = new InhTest();
                System.out.println("a= "+t.a);
                t.a();
                System.out.println("b= "+t.b);
                t.b();
        }
}
```

```
/*
 * 상속(클래스의 관계):
 *          부모클래스(객체)의 멤버들을 자식클래스(객체)가 물려받는것
 *  1. 상속을 사용하는 이유--> 기존에 만들어놓은 클래스의 재사용,
 *                                                  확장을 위해 사용한다.
 *  2. 자바에서는 단일상속만이 가능하다(부모클래스가 한 개 만 가능)
 *  3. 부모클래스(super)와 자식클래스(sub)가 존재한다.
 *  4. 자바에서 제공되어지는 모든 클래스들은 Object 라고 하는
 *     최상위 클래스로부터 상속되어진다.
 *  5. 사용자정의 클래스들도 Object 클래스라는 최상위클래스를
 *     상속  받아야 한다.
 *
 */
public class Parent {
    public String member1="난 부모에서 정의한 멤버변수";
    public void print(){
        System.out.println("난 부모에서 정의한 멤버 메쏘드 member1:"+member1);
    }
}
```

```
public class Child extends Parent {
        public String member2="난 자식에서 정의한 멤버변수";
         public void childPrint(){
                System.out.println("난 자식에서 정의한  메쏘드");
                System.out.println("member1:"+this.member1);
                System.out.println("member2:"+this.member2);
         }
}

public class ParentChildMain {
        public static void main(String[] args) {
                Child c1=new Child();
                System.out.println("c1.member1:"+c1.member1);
                c1.print();
                System.out.println("c1.member2:"+c1.member2);
                c1.childPrint();
        }
}
```

상속(inheritance)

Object클래스 – 모든 클래스의 최고조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.
- 상속계층도의 최상위에는 **Object**클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다.

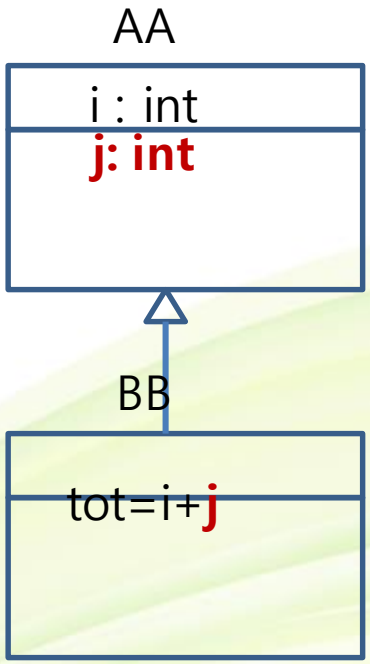
  toString(), equals(Object obj), hashCode(), ...

상속(inheritance)

- 예제 private 객체변수는 상속 안됨

InheritanceTest2.java

```
class AA {
   int i;
   private int j;          객체 변수 j를 private로 선언
   void setij(int x, int y) {
      i = x;
      j = y;
   }
}
class BB extends AA {
   int total;
   void sum() {            오류 발생
      total = i + j;
   }
}
public class InheritanceTest2 {
   public static void main(String args[]) {
      BB subOb = new BB(); subOb.sum();
   }
}
```



상속(inheritance)

```
class A1 {
        int a1 = 10;
        void a1() {         System.out.println("난 부모 메소드");        }
}
class A2 extends A1 {
        int a2 = 20;
        void a2() {         System.out.println("난 자식 메소드");        }
}
public class InhetEx {
        public static void main(String[] args) {
                A2 a2 = new A2();
                a2.a1();  a2.a2();
                System.out.println("a1 = " + a2.a1);
                System.out.println("a2 = " + a2.a2);
                A1 a1 = new A1();
                a1.a1(); // a1.a2();
        }
}
```

상속(inheritance)

```
class B1 {
        // 생성자는 부모것이 먼저 실행(생성)
        B1() {    System.out.println("난 부모 생성자");        }
        int a1 = 10;
        void a1() {         System.out.println("난 부모 메소드");        }
}
class B2 extends B1 {
        B2() {    System.out.println("난 자식 생성자");        }
        int a2 = 20;
        void a2() {         System.out.println("난 자식 메소드");        }
}
public class InhetEx2 {
        public static void main(String[] args) {
                B2 a2 = new B2();
                a2.a1();  a2.a2();
                System.out.println("a1 = " + a2.a1);
                System.out.println("a2 = " + a2.a2);
        }
}
```

상속(inheritance)

```
class C3 extends B2 {
        void a3() {         System.out.println("대박 ! 금요일이야");    }
}
class C4 extends A2 {
        void c4() {         System.out.println("헐 ~ ");          }
}
public class InhetEx3 {
        public static void main(String[] args) {
                C3 c = new C3();
                c.a1();  c.a2(); c.a3();
                System.out.println("a1 = " + c.a1);
                System.out.println("a2 = " + c.a2);
                System.out.println("==============");
                C4 c4 = new C4();
                c4.a1();  c4.a2();  c4.c4();
        }
}
```

상속(inheritance)

```
class D1 {
        int d1 = 6;
        D1() {   System.out.println("난 할아버지야"); }
        void d11() {  System.out.println("D1메소드"); }
}
class D2 extends D1 {
        int d2 = 7;
        void print() {  System.out.println("d2메소드"); }
}
class D3 extends D2 {
        D3() {  System.out.println("ㅠㅠㅠ");  }
        void d3() {  System.out.println("안녕 ! 컴 동지들"); }
}
public class Inher4 {
        public static void main(String[] args) {
                D3 d = new D3();
                d.d11(); d.print(); d.d3();
                System.out.println("d1 = "+d.d1);
                System.out.println("d2 = "+d.d2);
        }
}
```

상속(inheritance)

```
public class Person {
        private String name;
        private int age;
        Person() {}
        Person(String name, int age) {
                this.name = name; this.age = age;
        }
        public String getName() {  return name;        }
        public void setName(String name) {          this.name = name;          }
        public int getAge() {          return age;          }
        public void setAge(int age) {                  this.age = age;      }
        public void print() {
                System.out.println("---------------");
                System.out.println("이름 : "+name);
                System.out.println("나이 : "+age);
        }
}
```

상속(inheritance)

```
public class Teacher extends Person {
        private String subject;
        public Teacher(String name, int age, String subject) {
                setName(name); setAge(age); this.subject = subject;
        }
        public String getSubject() {
                return subject;
        }
        public void setSubject(String subject) {
                this.subject = subject;
        }
        void printTh() {
                print();
                System.out.println("과목 : " + subject);
        }
}
```

상속(inheritance)

```
public class Student extends Person {
        private String ban;
        Student(String name, int age, String ban) {
                setName(name); setAge(age); this.ban = ban;
        }
        public String getBan() {
                return ban;
        }
        public void setBan(String ban) {
                this.ban = ban;
        }
        void printSt() {
                print();
                System.out.println("반 : "+ban);
        }
}
```

상속(inheritance)

```
public class Manager extends Person {
        private String job;
        Manager(String name, int age, String job) {
                setName(name); setAge(age); this.job = job;
        }
        public String getJob() {
                return job;
        }
        public void setJob(String job) {
                this.job = job;
        }
        public void printMg() {
                print();
                System.out.println("업무 : "+job);
        }
}
```

상속(inheritance)

```
public class PersonEx {
        public static void main(String[] args) {
                Student st1 = new Student("홍길동", 23, "1반");
                Student st2 = new Student("설현", 21,"2반");
                Teacher th1 = new Teacher("박명수", 42, "Java");
                Teacher th2 = new Teacher("전현무", 37, "떼이터");
                Manager mg1 = new Manager("정준하", 41,"화장실청소");
                st1.printSt();  st2.printSt();
                th1.printTh();  th2.printTh();
                mg1.printMg();
        }
}
```

super

▶ super의 사용

☞ 하위 클래스에 의해 가려진 상위 클래스의 멤버 변수나 메소드에 접근할 때

▷ **super.객체변수**

▷ **super.메소드이름(매개변수)**

☞상위 클래스의 생성자를 호출할 때

▷ **super(매개변수)**

☞ **super문장은 반드시 첫 번째 라인에 와야 한다.**

상속 관계에서의 생성자 문제와 해결책

- 디폴트 생성자는 JVM이 제공해주지만, 클래스 내의 매개변수가 있는 생성자가 하나라도 존재하게 되면 JVM은 더 이상 디폴트 생성자를 제공해 주지 않게 된다.
- 만일 수퍼 클래스에 매개 변수가 있는 생성자를 정의하면서 매개 변수 없는 디폴트 생성자를 정의하지 않으면 수퍼 클래스에는 매개 변수 없는 생성자가 존재하지 않게 된다.
- 이러한 상태에서 서브 클래스의 생성자는 수퍼 클래스의 매개 변수 없는 디폴트 생성자를 여전히 호출하고 있기에 존재하지 않는 생성자를 호출하는 셈이 되어 문제가 발생하게 된다.

부모 생성자 호출(super(…))

- 명시적인 부모 생성자 호출
  - 부모 객체 생성할 때, 부모 생성자 선택해 호출

```
자식클래스( 매개변수선언, ... ) {
   super( 매개값, ... );
   ...
}
```

- super(매개값,…)
  - ⇨매개값과 동일한 타입, 개수, 순서 맞는 부모 생성자 호출
- 부모 생성자 없다면 컴파일 오류 발생
- 반드시 자식 생성자의 첫 줄에 위치
- 부모 클래스에 기본(매개변수 없는) 생성자가 없다면 필수 작성

super 의 예제

```
class A{
  int k=5;
  A(){
    System.out.println("A class");
  }
  A(int i){
    System.out.println("A(int) class");
  }
}

class SuperTest1 extends A{
  SuperTest1(){
    super(3);
    System.out.println("Extends01 Class");
  }
  public static void main(String[] a){
    Extends01 b = new Extends01();
    System.out.println("k=>"+b.k);
  }
}
```

super, this 의 예제

```
class D1 {
    int x = 1000;
    void display() {
        System.out.println("상위클래스 D1의 display() 메소드 입니다");
    }
}
class D2 extends D1 {
    int x = 2000;
    void display() {
        System.out.println("하위클래스 D2의 display() 메소드 입니다");
    }
    void write() {
        this.display();
        super.display();
        System.out.println("D2 클래스 객체의 x 값은 : " + x);
        System.out.println("D1 클래스 객체의 x 값은 : " + super.x);
    }
}
class InheritenceSuper {
  public static void main(String args[]) {
    D2 d = new D2();
    d.write();
  }
}
```

super

```
class F1 {
		// F1() {}
		F1(String str) { System.out.println("매개변수가 있는 생성자"+str); }
		void print() {		System.out.println("대박 ");  }
}
class F2 extends F1 {
		F2(String str) {
				super(str); // super() 부모 생성자 호출
				System.out.println("난 자식 생성자");
		}
		void disp() {		System.out.println("사건");   }
}
public class SuperEx {
		public static void main(String[] args) {
				F2 f= new F2("헐 ~");
				f.disp();
				f.print();
		}
}
```

super

```
class G1 {
		int x = 10;
		void print() {
			System.out.println("난 부모 메소드");
		}
}
class G2 extends G1 {
		int x = 20;
		void print() {
			super.print();
			System.out.println("난 자식 메소드 x = " +x);
			System.out.println("난 부모 x = " +super.x);
		}
}
public class SuperEx2 {
		public static void main(String[] args) {
			G2 g = new G2();
			g.print();
		}
}
```

super

```
class H1 {
		int x;
		H1(int x) {		this.x = x; System.out.println("부모 매개 1개");}
		H1(int x, int y) {
			this(x); System.out.println("부모 매개 2개");
		}
		void print() {		System.out.println("x = " + x); }
}
class H2 extends H1{
		H2(int x, int y) { super(x,y); System.out.println("자식 2개");}
		H2(int x, int y, int z) {
			this(x,y); System.out.println("자식 매개변수 3개");		}
}
public class SuperEx3 {
		public static void main(String[] args) {
			H2 h = new H2(34, 45, 67);
			h.print();
		}
}
```

내부 클래스(inner class)란?

- 클래스 안에 선언된 클래스
- 특정 클래스 내에서만 주로 사용되는 클래스를 내부 클래스로 선언한다.
- GUI어플리케이션(AWT, Swing)의 이벤트처리에 주로 사용된다.

```
class A {
    //...
}

class B {
    //...
}
```



```
class A {  // 외부 클래스
    //...
    class B { // 내부 클래스
        //...
    }
    //...
}
```

▶ 내부 클래스의 장점

- 내부 클래스에서 외부 클래스의 멤버들을 쉽게 접근할 수 있다.
- 코드의 복잡성을 줄일 수 있다.(캡슐화)

내부 클래스의 종류와 특징

- 내부 클래스의 종류는 변수의 선언위치에 따른 종류와 동일하다.
- 유효범위와 성질도 변수와 유사하므로 비교해보면 이해하기 쉽다.

내부 클래스	특 징
인스턴스 클래스 (instance class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 인스턴스멤버처럼 다루어진다. 주로 외부 클래스의 인스턴스멤버들과 관련된 작업에 사용될 목적으로 선언된다.
스태틱 클래스 (static class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 static멤버처럼 다루어진다. 주로 외부 클래스의 static멤버, 특히 static메서드에서 사용될 목적으로 선언된다.
지역 클래스 (local class)	외부 클래스의 메서드나 초기화블럭 안에 선언하며, 선언된 영역 내부에서만 사용될 수 있다.
익명 클래스 (anonymous class)	클래스의 선언과 객체의 생성을 동시에 하는 이름없는 클래스(일회용)

```
class Outer {
    int iv=0;
    static int cv=0;

    void myMethod() {
        int lv=0;
    }
}
```

```
class Outer {
    class InstanceInner {}
    static class StaticInner {}

    void myMethod() {
        class LocalInner {}
    }
}
```

Inner 클래스

- **중첩(내부) 클래스 (Inner Class)**
  - ⇨클래스 내부에 또 다른 클래스를 가짐으로 클래스 관리의 효율을 높인 것(static 포함불가)
- **중첩 클래스의 형식과 생성파일**
  - ⇨형식) class Outer { class Inner { … } }
  - ⇨생성파일) Outer.class, Outer$Inner.class
- **중첩 클래스 객체 생성**
  - ⇨Outer.Inner oi = new Outer().new Inner();

```
class Outer {
    private int height;
    private int           width;
    public Outer(int h, int w)  {
              height = h;
              width = w;
     }
    public Inner getInner()    {
    return new Inner();
    }

    class Inner    {
        private float rate = 0.5f;
        public float capacity()  {
              return rate * height * width;
        }
    }
}
```

```
class TestOuter  {
  public static void main(String args[])  {
    Outer outRef = new Outer(100,200);
    Outer.Inner myInner = outRef.getInner();
    // Inner myInner = outRef.getInner();
    System.out.println("Inner value is " +
myInner.capacity());

    Outer.Inner yourInner = outRef.new Inner();
    System.out.println("Inner value is " +
yourInner.capacity());
  }
}
```

정적 중첩 클래스

- **정적 중첩 클래스(Static Inner Class)**
  - ⇨중첩 클래스 내부에 static 멤버를 포함할 수 있는 형태(Outer의 non-static 멤버 포함 불가)
- **정적 중첩 클래스의 형식과 생성파일**
  - ⇨형식) class Outer { static class Inner {…} }
  - ⇨생성파일) Outer.class, Outer$Inner.class
- **정적 중첩 클래스 객체 생성**
  - ⇨Outer.Inner oi = new Outer.Inner();

내부 클래스의 제어자와 접근성(1/5)

- 내부 클래스의 접근제어자는 변수에 사용할 수 있는 접근제어자와 동일하다.

```
class Outer {
    private int iv=0;
    protected static int cv=0;

    void myMethod() {
        int lv=0;
    }
}
```



```
class Outer {
    private class InstanceInner {}
    protected static class StaticInner {}

    void myMethod() {
        class LocalInner {}
    }
}
```

- static클래스만 static멤버를 정의할 수 있다.

```
class InnerEx1 {
    class InstanceInner {
        int iv = 100;
//      static int cv = 100;            // 에러! static변수를 선언할 수 없다.
        final static int CONST = 100;   // final static은 상수이므로 허용한다.
    }

    static class StaticInner {
        int iv = 200;
        static int cv = 200;        //
    }

    void myMethod() {
        class LocalInner {
            int iv = 300;
//          static int cv = 300;            // 에러! static변수를 선언할 수 없다.
            final static int CONST = 300; // final static은 상수이므로 허용
        }
    } // void myMethod() {
}
```

```
class InnerTest {
    public static void main(String args[]) {
        System.out.println(InnerEx1.InstanceInner.CONST);
        System.out.println(InnerEx1.StaticInner.cv);
    }
}
```

```
class Outer1 {
   private int x = 100;
   private static int y = 200;
   public Outer1() {}
   public void disp() {
            System.out.println("x = " + x);
            System.out.println("y = " + y);
   }
   static class Inner1 {
            private int a = 10;
            static int b = 20;
            public Inner1() {}
            public void disp_in() {
                     System.out.println("a = " + a);
                     System.out.println("b = " + b);
                     //System.out.println("x = " + x);
                     System.out.println("y = " + y);
                     //disp();
            }
   }
}
```

```
public class Exam_02 {
   public static void main(String[] ar) {
        Outer1.Inner1 oi = new Outer1.Inner1();
        oi.disp_in();
       System.out.println("b = " +
Outer1.Inner1.b);
    }
}
```

지역 중첩 클래스

- **지역 중첩 클래스(Local Inner Class)**
  - ⇨메서드 실행 시에 사용되는 클래스를 정의한 형식으로 접근자나 지정어를 가질 수 없다.
- **지역 중첩 클래스의 형식 및 생성 파일**
  - ⇨형식) class Outer {
    method() { class Inner { … } } }
  - ⇨생성파일) Outer.class, Outer$숫자Inner.class
- **객체 생성은 외부에서 할 수 없다.**

```
class Outer3 {
    private int x = 100;
    private static int y = 200;
    public void disp() {
        class Inner3 {
            private int a = 10;
            public void disp_in() {
              System.out.println("a = " + a);
              System.out.println("x = " + x);
              System.out.println("y = " + y);
            }
        }
        Inner3 in = new Inner3();
        in.disp_in();
    }
}
```

```
class Exam_03 {
public static void main(String[] ar) {
        Outer3 ot = new Outer3();
        //Outer3.Inner3 oi = ot.new Inner3();
        ot.disp();
    }
}
```

익명 중첩 클래스

- **익명 중첩 클래스(Anonymous Inner Class)**
  - ⇨기존 클래스의 특정 메서드를 오버라이딩 하여 원하는 형태로 재정의 하여 사용하는 방식
  - ⇨외부 멤버 중 final만 포함할 수 있다.
- **익명 중첩 클래스의 형식 및 생성파일**
  - ⇨형식) class Inner { … }

    class Outer { method() { new Inner() {…}}}
  - ⇨생성파일) Outer.class, Outer$숫자.class
- **new Inner() 자체가 객체 생성임.**

```
import java.awt.*;
import java.awt.event.*;

class InnerEx8  {
    public static void main(String[] args)        {
            Button b = new Button("Start");
            b.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            System.out.println("ActionEvent occurred!!!");
                    }
                } // 익명 클래스의 끝
            );
    } // main메서드의 끝
} // InnerEx8클래스의 끝
```

클래스간의 관계 – 포함관계(composite)

- 포함(composite)이란?
  - 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
  - 작은 단위의 클래스를 먼저 만들고, 이 들을 조합해서 하나의 커다란 클래스를 만든다.

```
class Circle {
    int x; // 원점의 x좌표
    int y; // 원점의 y좌표
    int r; // 반지름(radius)
}
```

```
class Circle {
    Point c = new Point();
    int r; // 반지름(radius)
}
```

```
class Point {
    int x;
    int y;
}
```

```
class Car {
    Engine e = new Engine(); // 엔진
    Door[] d = new Door[4];  // 문, 문의 개수를 넷으로 가정하고 배열로 처리했다.
    //...
}
```

클래스간의 관계결정하기 – 상속 vs. 포함

- 가능한 한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게 한다.

-'is-a'와 'has-a'를 가지고 문장을 만들어 본다.

원(Circle)은 점(Point)**이다**. - Circle **is a** Point.
원(Circle)은 점(Point)을 **가지고 있다**. - Circle **has a** Point.

상속관계 - '~은 ~이다.(is-a)'
포함관계 - '~은 ~을 가지고 있다.(has-a)'

```
class Point {
    int x;
    int y;
}
```

```
class Circle extends Point{
    int r; // 반지름(radius)
}
```

```
class Circle {
    Point c = new Point(); // 원점
    int r; // 반지름(radius)
}
```

포함관계(composite)

```
public class Engine {
	private int cc;
	private String type;
	Engine() {}
	Engine(int cc,String type) {
		this.cc = cc; this.type = type;
	}

	public int getCc() {		return cc;		}
	public void setCc(int cc) {		this.cc = cc;		}
	public String getType() {	return type;		}
	public void setType(String type) {	this.type = type;	}

	void print() {
		System.out.println("----------------");
		System.out.println("배기량 : "+cc);
		System.out.println("엔진타입 : "+type);
	}
}
```

포함관계(composite)

```
public class Car {
        private Engine eg;
        private String color;
        Car() {}
        Car(Engine eg, String color) {
                this.eg = eg; this.color = color;
        }
        public Engine getEg() {         return eg; }
        public void setEg(Engine eg) {          this.eg = eg; }
        public String getColor() {              return color;           }
        public void setColor(String color) {
                this.color = color;
        }
        void print() {
                eg.print();
                System.out.println("색깔 : "+color);
        }
}
```

포함관계(composite)

```
public class CarEx {
	public static void main(String[] args) {
		Engine eg1 = new Engine(2000, "GDI");
		Engine eg2 = new Engine(1500, "DOHC");
		Car c1 = new Car(eg1, "빨강");
		Car c2 = new Car(eg2, "노랑");
		Car c3 = new Car(eg1, "파랑");
		c1.print();		c2.print();		c3.print();
		System.out.println("==================");
		Car[] car = {c1, c2, c3};
		for (Car c : car) {
			c.print();
		}
		System.out.println("==================");
		for (int i =0; i < car.length;i++) {
			car[i].print();
		}
	}
}
```

포함관계(composite)

```
public class FruitSeller {
	int numOfApple = 20;
	int money = 0;
	final int PRICEPERAPPLE = 1000;

	int saleApple(int amt) {
		money+=amt;
		int num = amt/PRICEPERAPPLE;
		numOfApple-=num;
		return num;
	}

	void print() {
		System.out.println("판매자 사과갯수 : "+numOfApple);
		System.out.println("판매자 금전잔액 : "+money);
	}
}
```

포함관계(composite)

```
public class FruitBuyer {
	int numOfApple = 0;
	int money = 5000;
	void buyFruit(FruitSeller fs, int amt) {
		int num = fs.saleApple(amt);
		money-=amt;
		numOfApple+=num;
	}
	void print() {
		System.out.println("구매자 사과갯수 :
"+numOfApple);
		System.out.println("구매자 금전잔액 : "+money);
		System.out.println("----------------");
	}
}
```

포함관계(composite)

```
public class FruitEx {
        public static void main(String[] args) {
                FruitSeller fs1 = new FruitSeller();
                FruitSeller fs2 = new FruitSeller();
                FruitBuyer  fb1 = new FruitBuyer();
                FruitBuyer  fb2 = new FruitBuyer();
                fb1.buyFruit(fs1, 3000);
                fb1.buyFruit(fs2, 2000);
                fb2.buyFruit(fs1, 2000);
                fs1.print();
                fs2.print();
                fb1.print();
                fb2.print();
        }
}
```

포함관계(composite)

```
public class FruitSeller2 {
        int numOfApple = 20;       int money = 0; String name;
        final int PRICEPERAPPLE = 1000;
        FruitSeller2(String name,int money, int numOfApple) {
                this.name = name; this.money = money;
                this.numOfApple = numOfApple;
        }
        int saleApple(int amt) {
                money+=amt;
                int num = amt/PRICEPERAPPLE;
                numOfApple-=num;
                return num;
        }
        void print() {
                System.out.println(name+" 사과갯수 : "+numOfApple);
                System.out.println(name+" 금전잔액 : "+money);
                System.out.println("----------------");
        }
}
```

포함관계(composite)

```
public class FruitBuyer2 {
	int numOfApple = 0; int money = 5000; String name;
	FruitBuyer2(String name,int money, int numOfApple) {
		this.name = name; this.money = money;
		this.numOfApple = numOfApple;
	}
	void buyFruit(FruitSeller2 fs, int amt) {
		int num = fs.saleApple(amt);
		money-=amt;
		numOfApple+=num;
	}
	void print() {
		System.out.println(name+" 사과갯수 : "+numOfApple);
		System.out.println(name+" 금전잔액 : "+money);
		System.out.println("----------------");
	}
}
```

포함관계(composite)

```
public class FruitEx2 {
	public static void main(String[] args) {
		FruitSeller2 fs1 = new FruitSeller2("전지현",10000,30);
		FruitSeller2 fs2 = new FruitSeller2("최순실",0,20);
		FruitBuyer2  fb1 = new FruitBuyer2("길동",20000,0);
		FruitBuyer2  fb2 = new FruitBuyer2("하니",10000,5);
		fb1.buyFruit(fs1, 3000);
		fb1.buyFruit(fs2, 2000);
		fb2.buyFruit(fs1, 2000);
		fs1.print();
		fs2.print();
		fb1.print();
		fb2.print();
	}
}
```

포함관계(composite)

```
public class FruitSeller3 {
	int numOfApple = 20;int money = 0; String name;
	final int PRICEPERAPPLE = 1000;
	FruitSeller3(String name,int money, int numOfApple) {
		this.name = name; this.money = money;
		this.numOfApple = numOfApple;
	}
	int saleApple(int amt) {
		int num = amt/PRICEPERAPPLE;
		if (numOfApple >= num) {
			money+=amt;			numOfApple-=num;
			System.out.printf("%d개 판매, 수입%d원\n",num,amt);
		} else {
			System.out.println("사과 떨어졌어");
			num = 0;
		}
		return num;
	}
	void print() {
		System.out.println(name+" 사과갯수 : "+numOfApple);
		System.out.println(name+" 금전잔액 : "+money);
		System.out.println("----------------");
	}
}
```

포함관계(composite)

```
public class FruitBuyer3 {
        int numOfApple = 0; int money = 5000; String name;
        FruitBuyer3(String name,int money, int numOfApple) {
                this.name = name; this.money = money;
                this.numOfApple = numOfApple;
        }
        void buyFruit(FruitSeller3 fs, int amt) {
                if (money<amt) System.out.println("돈도 없는데 우찌 사노 ?");
                else {
                        int num = fs.saleApple(amt);
                        if (num != 0) {
                                money-=amt;
                                numOfApple+=num;
                                System.out.printf("사과 %d개 득템, 지출 %d원\n",
                                                num,amt);
                        }
                }
        }
        void print() {
                System.out.println(name+" 사과갯수 : "+numOfApple);
                System.out.println(name+" 금전잔액 : "+money);
                System.out.println("----------------");
        }
}
```

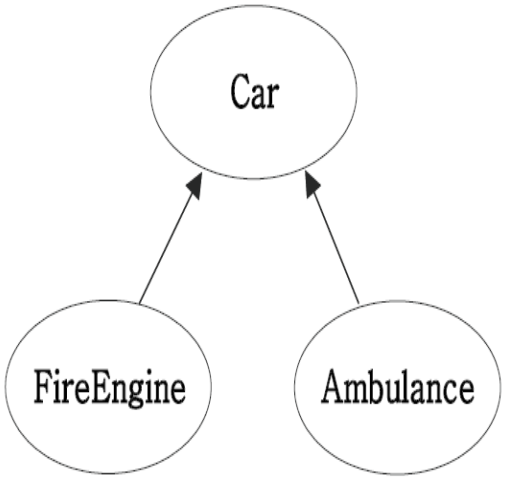
```
public class FruitEx3 {
	public static void main(String[] args) {
		FruitSeller3 fs1 = new FruitSeller3("전지현",10000,30);
		FruitSeller3 fs2 = new FruitSeller3("최순실",0,20);
		FruitBuyer3  fb1 = new FruitBuyer3("길동",20000,0);
		FruitBuyer3  fb2 = new FruitBuyer3("하니",10000,5);
		fb1.buyFruit(fs1, 32000);
		fb1.buyFruit(fs2, 2000);
		fb2.buyFruit(fs1, 12000);
		fs1.print();
		fs2.print();
		fb1.print();
		fb2.print();
	}
}
```

참조변수의 형변환

- 서로 상속관계에 있는 타입간의 형변환만 가능하다.
- 자손 타입에서 조상타입으로 형변환하는 경우, 형변환 생략가능

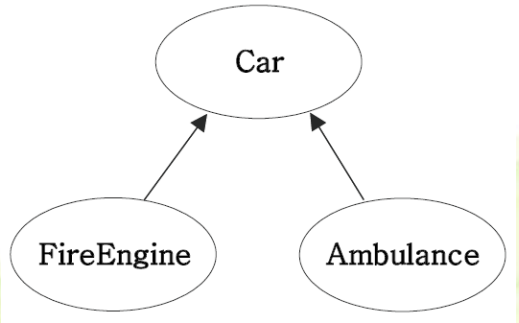
자손타입 → 조상타입 (Up-casting) : 형변환 생략가능
자손타입 ← 조상타입 (Down-casting) : 형변환 생략불가

```
class Car {
    String color;
    int door;

    void drive() {  // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }

    void stop() {   // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {  // 소방차
    void water() {  // 물뿌리는 기능
        System.out.println("water!!!");
    }
}

class Ambulance extends Car {   // 구급차
    void siren() {  // 사이렌을 울리는 기능
        System.out.println("siren~~~");
    }
}
```

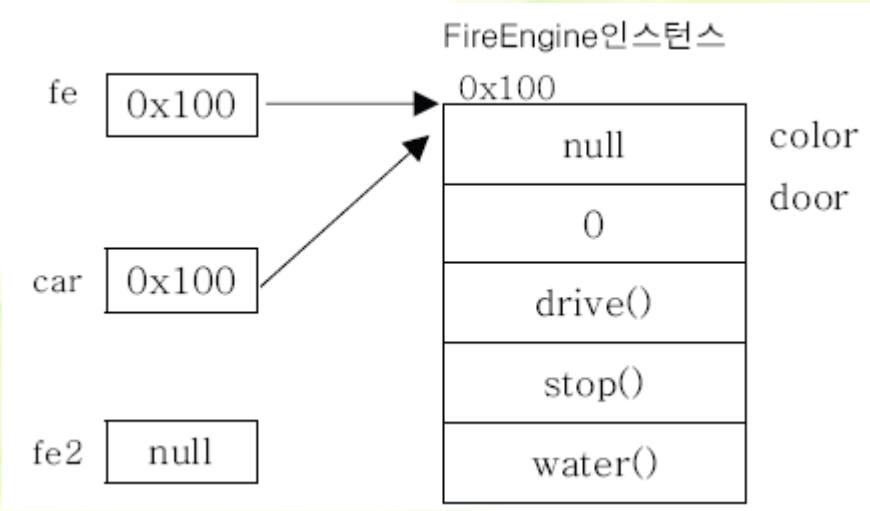



```
FireEngine f
Ambulance a;

a = (Ambulance)f;
f = (FireEngine)a;
```

```java
class Car {
    String color;
    int door;

    void drive() {  // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }

    void stop() {   // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {  // 소방차
    void water() {  // 물뿌리는 기능
        System.out.println("water!!!");
    }
}

class Ambulance extends Car {   // 구급차
    void siren() {  // 사이렌을 울리는 기능
        System.out.println("siren~~~");
    }
}
```




```java
public static void main(String args[]) {
    Car car = null;
    FireEngine fe = new FireEngine();
    FireEngine fe2 = null;

    fe.water();
    car = fe;   // car = (Car)fe; 조상 <- 자손
//  car.water();
    fe2 = (FireEngine)car; // 자손 <- 조상
    fe2.water();
}
```



참조변수의 형변환

```
public class Car2 {
	void drive() {
		System.out.println("자동차가 달린다");
	}
}
class FireEngine extends Car2 {
	void drive() {
		System.out.println("앵소리나면 달린다");
	}
	void job() {
		System.out.println("불을 끈다");
	}
}
class Ambulance extends Car2 {
	void drive() {
		System.out.println("환자를 싣고 달린다");
	}
}
```

참조변수의 형변환

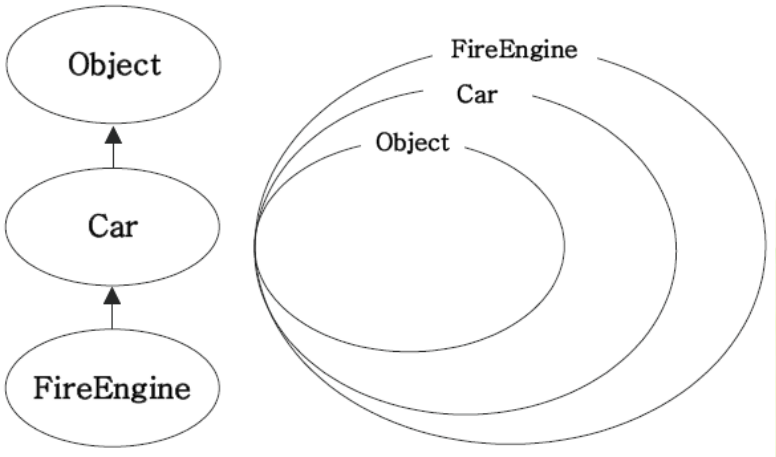
```
public class Car2Ex {
	public static void main(String[] args) {
		Car2 c1 = new Car2();
		FireEngine fe = new FireEngine();
		Ambulance  ab = new Ambulance();
		c1.drive(); fe.drive();  ab.drive();
		System.out.println("==================");
		Car2 c2 = fe;   Car2 c3 = ab;
		c2.drive();     c3.drive();
		System.out.println("==================");
		// FireEngine fe2 = (FireEngine)c1; // 실행 안됨
		FireEngine fe3 = (FireEngine)c2;
		fe3.drive();
		System.out.println("==================");
		Car2[] car = new Car2[2];
		car[0] = fe;   car[1] = ab;
		for (Car2 c : car) {
			c.drive();
		}
	}
}
```

instanceof연산자

- 참조변수가 참조하는 인스턴스의 실제 타입을 체크하는데 사용.
- 이항연산자이며 피연산자는 참조형 변수와 타입. 연산결과는 true, false.
- instanceof의 연산결과가 true이면, 해당 타입으로 형변환이 가능하다.

```
class InstanceofTest {
    public static void main(String args[]) {
        FireEngine fe = new FireEngine();

        if(fe instanceof FireEngine) {
            System.out.println("This is a FireEngine instance.");
        }

        if(fe instanceof Car) {
            System.out.println("This is a Car instance.");
        }

        if(fe instanceof Object) {
            System.out.println("This is an Object instance
        }
    }
}
```

```
---------- java ----------
This is a FireEngine instance.
This is a Car instance.
This is an Object instance.

출력 완료 (0초 경과)
```




```
void method(Object obj) {
    if(c instanceof Car) {
        Car c = (Car)obj;
        c.drive();
    } else if(c instanceof FireEngine) {
        FireEngine fe = (FireEngine)obj;
        fe.water();
    }
}
```

참조변수의 형변환

```
public class Car2Ex2 {
        public static void main(String[] args) {
                FireEngine fe = new FireEngine();
                Ambulance  ab = new Ambulance();
                Car2[] car = new Car2[2];
                car[0] = fe;   car[1] = ab;
                for (Car2 c : car) {
                        c.drive();  // 부모에 있는 메소드만 실행 가능
                                // 실행되는 메소드는 자식 것
                        // c.job();
                        if (c instanceof FireEngine) {
                                FireEngine fe2 = (FireEngine)c;
                                fe2.job();
                        }
                }
        }
}
```

참조변수와 인스턴스변수의 연결

- 멤버변수가 중복정의된 경우, 참조변수의 타입에 따라 연결되는 멤버변수가 달라진다. (참조변수타입에 영향받음)
- 메서드가 중복정의된 경우, 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입에 정의된 메서드가 호출된다.(참조변수타입에 영향받지 않음)

```
class Parent {
    int x = 100;

    void method() {
        System.out.println("Parent Method");
    }
}

class Child extends Parent {
    int x = 200;

    void method() {
        System.out.println("Child Method");
    }
}
```

```
p.x = 100
Child Method
c.x = 200
Child Method
```

```
class Parent {
    int x = 100;

    void method() {
        System.out.println("Parent Method");
    }
}

class Child extends Parent { }

public static void main(String[] args) {
    Parent p = new Child();
    Child c = new Child();

    System.out.println("p.x = " + p.x);
    p.method();

    System.out.println("c.x = " + c.x);
    c.method();
}
```

```
p.x = 100
Parent Method
c.x = 100
Parent Method
```

```
public class Person2 {
	private String name;
	private int age;
	Person2(String name, int age) {
		this.name = name; this.age = age;
	}
	public String getName() {	return name;		}
	public void setName(String name) {	this.name = name;	}
	public int getAge() {		return age;		}
	public void setAge(int age) {		this.age = age;		}
	void print() {
		System.out.println("---------------");
		System.out.println("이름 : "+name);
		System.out.println("나이 : "+age);
	}
}
```

```
public class Teacher2 extends Person2 {
        private String subject;
        public Teacher2(String name, int age, String subject) {
                super(name,age); this.subject = subject;
        }
        public String getSubject() {
                return subject;
        }
        public void setSubject(String subject) {
                this.subject = subject;
        }
        void print() {
                super.print();
                System.out.println("과목 : " + subject);
        }
}
```

```
public class Manager2 extends Person2 {
	private String job;
	Manager2(String name, int age, String job) {
		super(name,age); this.job = job;
	}
	public String getJob() {
		return job;
	}
	public void setJob(String job) {
		this.job = job;
	}
	public void print() {
		super.print();
		System.out.println("업무 : "+job);
	}
}
```

```
public class Student2 extends Person2 {
	private String ban;
	Student2(String name, int age, String ban) {
		super(name, age); this.ban = ban;
	}
	public String getBan() {
		return ban;
	}
	public void setBan(String ban) {
		this.ban = ban;
	}
	void print() {
		super.print();
		System.out.println("반 : "+ban);
	}
}
```

```
public class Person2Ex {
	public static void main(String[] args) {
		Person2[] p = new Person2[5];
		p[0] = new Student2("수지",21,"1반");
		p[1] = new Student2("강호동",43,"2반");
		p[2] = new Teacher2("효리", 23, "Java");
		p[3] = new Teacher2("유즈", 17, "스프링");
		p[4] = new Manager2("준하", 32,"변기청소");

		for (int i=0; i<p.length;i++) {
			p[i].print();
		}
	}
}
```

연습문제

- 1. 다음 문제의 에러를 찾아서 고쳐라

```
class Product  {
    int price; // 제품의 가격
     int bonusPoint; // 제품구매 시 제공하는 보너스점수
     Product(int price) {
          this.price = price;
          bonusPoint =(int)(price/10.0);
     }
}

class Tv extends Product {
    Tv() {}
    public String toString() {
           return "Tv";
    }
}
class Exercise7_5 {
    public static void main(String[] args) {
         Tv t = new Tv();
    }
}
```

2. 다음 코드의 실행했을 때 호출되는 생성자의 순서와 실행결과를 적으시오.

```
class Parent {
     int x=100;
      Parent() {
          this(200); // Parent(int x)를 호출
      }
      Parent(int x) {    this.x = x;   }
      int getX() {    return x;    }
}
class Child extends Parent {
    int x = 3000;
   Child() {
         this(1000); // Child(int x)를 호출
   }
   Child(int x) {    this.x = x;    }
}
class Exercise7_7 {
    public static void main(String[] args) {
         Child c = new Child();
         System.out.println("x="+c.getX());
    }
}
```