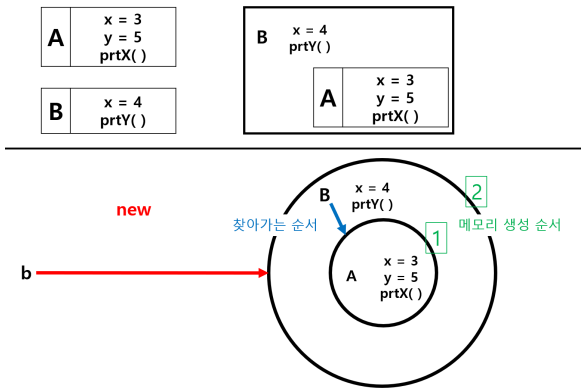
01.10 火_ 상속 + 다형성

```
package test;

class A {
  int x = 3;
  int y = 5;

  void prtX() {
    System.out.println(x);
  }
}

class B extends A {
  int x = 4;

  void prtY() {
    System.out.println(y);
  }
}


public class TEST_1 {

  public static void main(String[] args) {
    B b = new B();

    System.out.println(b.y);   // 5
    System.out.println(b.x);   // 4
    b.prtX();                  // 3
    b.prtY();                  // 5
  }

}
```



지한테 없는 걸 부모한테서 찾기 위해 안으로 들어간다.

❓ 상속을 받으면❓

⇒ 메모리상에 부모가 먼저 찍히고 그 밖에 자신을 찍는다.

1. 상속을 받으면 부모 class가 제일 안쪽으로 들어간다
2. 상속을 받은 자식 class 의 객체를 찍으면 자식 클래스의 객체만 찍히는게 아니라 그 부모 클래스의 객체도 같이 합해져서 찍힌다.
3. 참조 변수를 가지고 들어가면 **제일 첫 번째 만나는 놈을 잡는다** ,
4. 없으면 계속 안쪽으로 들어간다
5. 없으면 error 발생

※ 메서드를 호출하면 해당 메서드를 호출한 지점의 영역에서 변수 값들을 찾기 시작한다 ※

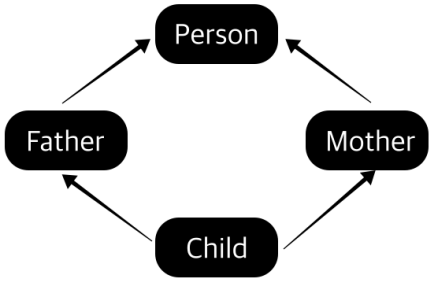
상속을 받은 객체의 class를 찍는다

상속받은 class의 객체만 찍히는게 아니라 부모 class까지 전부다 찍힌다.

```
package test;

class A {
  A(){System.out.println("A생성자 호출! ");}
  int x = 3;
  int y = 5;

  void prtX() {
    System.out.println(x);
  }
}

class B extends A {
  B(){System.out.println("B생성자 호출! ");}
  int x = 4;

  void prtY() {
    System.out.println(y);
  }

  void setY(int argvalue) {
    y = argvalue ;
  }ㄴ
}


public class TEST_1 {

  public static void main(String[] args) {
    B b = new B();
    b.setY(100);
    System.out.println(b.y);
    System.out.println(b.x);
    b.prtX();
    b.prtY();
  }
```

```
}
```

Diamond Problem



상속의 종류 2가지

- `단일 상속` - 부모가 1개 (JAVA에서는 단일 상속만 가능)
- `다중 상속` - 부모가 2개 이상 가능
  - Diamond Problem (어떤 부모의 멤버를 땡겨와야 하는지 모름)
  - 모호성의 문제가 생긴다 (부모가 같은 변수를 갖고 있는 경우)

내가 만든 상속 코드!

```
package test;

// 클래스. 1
class ChunganLee {

  // 멤버 변수
  String family = "청안";
  /*static*/ String firstName = "이";  /*static을 붙이니 전부 박씨가 되어버립니다... ;;; */
  String name;
  int num ;
  String num_char = "대손 (장손)";


  // 멤버 메소드
  String info_of_lee() {
    return family +" " +firstName + name +" "+ num +num_char;
  }
}

//클래스. 2      -> 클래스 1참조
class Sangyeul extends ChunganLee {

//  초기화 블록
  {

    name= "상열";
```

```
    num = 23 ;
  }

  // 멤버 메소드
  static String lastName() {
    String lastName="상열";
    return lastName;
  }

}

//클래스. 3      -> 클래스 2참조
class Eunyong extends Sangyeul {

//    firstName ;
//  초기화 블록
  {
    firstName ="박";
    family ="밀양";
    name= "은영";
    num_char ="처";
  }

  // 멤버 메소드
  static String husband() {
    String last_Name="상열";
    return last_Name;
  }

  String info_of_lee() {
    return family +" " +firstName + name +" "+  "/ 夫 :"+firstName+husband();
  }

}

//클래스. 4      -> 클래스 2참조
class Jaeil extends Sangyeul {

  // 멤버 변수
  String father ;

  // 초기화 블록
  {
     name= "재일";
     num = 24 ;
     father = Sangyeul.lastName();
  }

  // 멤버 메소드
    static String last_Name() {
      String last_Name="재일";
      return last_Name;
    }

  // 멤버 메소드
  String info_of_lee() {
    return family +" " +firstName + name +" "+ num +num_char+ "/ 父 :"+firstName+father;
  }
}

//클래스. 5      -> 클래스 2참조
class Jaesung extends Sangyeul {

  // 멤버 변수
  String father ;
  String brother ;


  // 초기화 블록
```

```
  {
     name= "재성";
     brother = Jaeil.last_Name();
     num_char = "";
     father = Sangyeul.lastName();
  }

  // 멤버 메소드
  String info_of_lee() {
    return family +" " +firstName + name +" "+"/ 父 :"+firstName+father +"/ 兄 :" +firstName+brother;
  }

}


public class Pedigree_Of_Lee {

  public static void main(String[] args) {


    Sangyeul tsitsi = new Sangyeul();
    System.out.println(tsitsi.info_of_lee());

    Eunyong haha = new Eunyong();
    System.out.println(haha.info_of_lee());


    Jaeil ore = new Jaeil();
    System.out.println(ore.info_of_lee());

    Jaesung otouto = new Jaesung();
    System.out.println(otouto.info_of_lee());
  }

}
```

```
package test;

class A {
  int x = 3;
}

class B extends A {

  int x = 4;

  void prtK() {
    System.out.println(x); // 일단은 지 클래스 지 영역안에서 찾는다
  }
}

class C extends B {

  int x = 5;

  void prtX() {
    prtK();
  }
}


public class TEST_1 {

  public static void main(String[] args) {
```

```
    C obj = new C();
    obj.prtX();
  }
}
```

[상속 ,다형성 ,추상] 이 세가지는 묶여서 돌아간다

다형성

⭐ 다양한 형태의 성질을 가질 수 있다 ⇒ 대상은 `참조 변수` , `매개변수` , **`메소드의 반환형`**

⭐ 자바에서는 한 타입의 `참조 변수`로 여러 타입의 객체를 참조할 수 있도록 함
⇒ 조상클래스 타입의 `참조 변수`로 자손클래스의 인스턴스(객체)를 참조할 수 있도록 함

※ **반대로 자손 타입의 참조 변수로 조상 타입의 인스턴스를 참조할 수는 없다!!**

⭐ 참조변수의 타입에 따라 사용할 수 있는 멤버의 개수가 달라진다

- `참조 변수`가 사용되는 목적은 객체를 가르키기 위해서 사용된다
- 원래 `참조 변수`가 가질 수 있는 자료형은 1개!
- ⭐⭐⭐ `참조 변수`가 가르키는 객체의 자료형은 `참조 변수`와 같아야 한다 ⭐⭐⭐

다형성을 주입하면 `참조 변수`가 **자기 자신과 같은 자료형을 가지는 객체**
+ 🅰️ (나로부터 상속 받은 자식들) 를 가르킬 수 있다
⇒ 자기 자신과 다른 형태의 객체를 가르킬 수 있다

⭐ 다형성이 나오면 반드시 `참조 변수`가 따라 나와야 한다!

⭐다형성을 사용하는 이유⭐

> ⭐ 서로 다른 자료형임에도 불구하고 부모의 형을 이용해
> **하나의 같은 자료형으로 통합해서 관리** 가능!!

참조 변수 의 다형성

```
package Test;

class Terran {}
class Scv extends Terran {}
class Marine extends Terran {}
class Tank extends Terran {}

class bar {}  // 자동으로 생략 된 ( extends Object {} )

public class MyProject {
  public static void main(String args[]) {

    bar b = new bar();

    Scv [] scvList = new Scv[200];
    Marine [] marineList = new Marine[200];
    Tank [] tankList = new Tank[200];

    scvList[0] = new Scv();
    marineList[0] = new Marine();
    tankList[0] = new Tank();

  //===========================================

    Terran unitList[] new Terran[200]; // 다형성 미적용

  // ☆ 다형성 적용 ☆    =>
    Terran unitList[] = new Scv [200];  // 참조변수 : unitList
    Terran unitList[] = new Marine [200];
    Terran unitList[] = new Tank [200];


    unitList[0] = new Scv();
    unitList[0] = new Marine();
    unitList[0] = new Tank();
  }
}
```

1. 프로그램을 만들었는데, 클래스가 여러 개 있다.
2. 어? 공통적인 기능이 있네? ⇒ 상속으로 올린다
3. 이들을 하나로 묶어서 관리 해야 하는데 부모의 자료형으로 다형성을 적용 해서 돌리자!

Object의 생략

```
package test;

// 자바에서 Object 를 자동으로 생성한다.
// 만든 클래스들은 전부 Object로 부터 상속 받고 JAVA안에서 공통적으로 사용해야할 기능들을
// Method로 다 구현을 해놨다.
// jvm이 돌아갈 때 객체들을 하나의 자료형으로 관리해야한다 => 다형성을 이용해 Object로 관리!

class Bar /*extends Object 이 생략되어 있다.  */ {

}

public class TEST_4 {

  public static void main(String[] args) {

    Bar.b = new Bnd();


    System.out.println();

  }

}
```

매개변수 의 다형성

```
package Test;

class Terran { int hp;}
class Scv extends Terran {}
class Marine extends Terran {}
class Tank extends Terran {}

class bar extends Object {}

public class MyProject {

  int getScvHP(Scv argScv) {
    return argScv.hp;
  }

  int getMarineHP(Marine argMarine) {
    return argMarine.hp;
  }

  int getHP(Terran argScv) {
    return argScv.hp;
  }

  public static void main(String args[]) {

    getHp(new Scv());
    bar b = new bar();

    Scv scvList[] = new Scv[200];
    Marine marineList[] = new Marine[200];
    Tank tankList[] = new Tank[200];

    scvList[0] = new Scv();
    marineList[0] = new Marine();
    tankList[0] = new Tank();

    //======================
```

```
    Terran unitList[] = new Terran[200];

    unitList[0] = new Scv();
    unitList[0] = new Marine();
    unitList[0] = new Tank();
  }
}
```