

능동적 사고 방식의

java

강사 박주병

Park Ju Byeong

Park Ju Byeong



Part09 객체지향2



01 상속

02 오버라이딩

03 super 생성자

04 실습 문제

01 —

상속

상속의 필요성

- 중복된 멤버변수 선언

Marine

```
int hp=40;  
static int power=4;  
static int armor=0;
```

Zergling

```
int hp=40;  
static int power=4;  
static int armor=0;
```

Zealot

```
int hp=40;  
static int power=4;  
static int armor=0;
```

상속의 필요성

- 중복된 멤버메서드 선언

Marine

```
boolean attack(Zergling target)
{
    target.hp -= (power - target.armor);

    return target.hp <= 0;
}

void showState()
{
    System.out.println("체력: " + hp + "\t 공격력: " + power + "\t 방어력: " + armor);
}

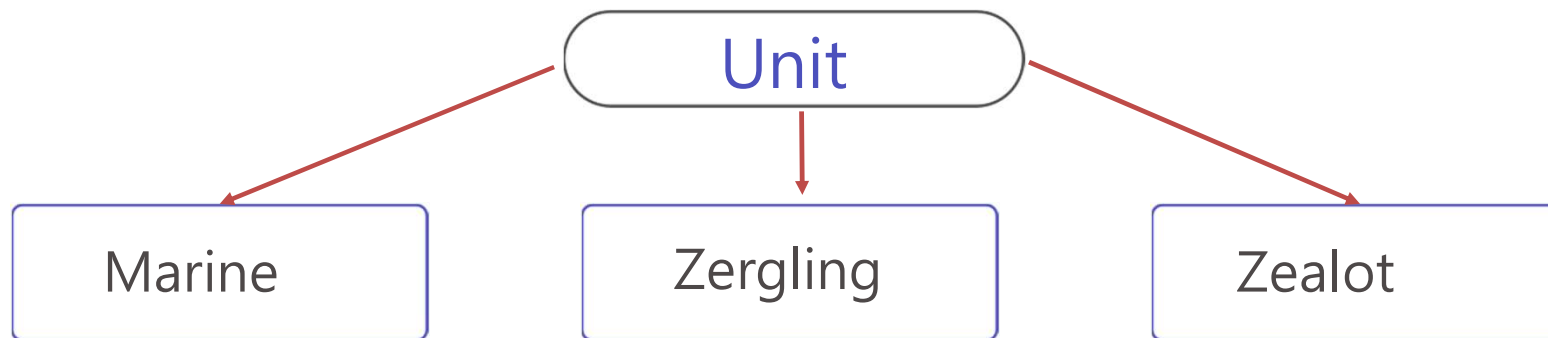
void powerUp()
{
    power++;
}
```

```
void attack(Marine target)
{
    target.hp -= (power - target.armor);
}
```

그외 유닛들 ...

```
int hp=40;  
static int power=4;  
static int armor=0;
```

```
public void attack(Unit target)  
{  
    target.hp -= (power-target.armor);  
}
```



상속

- 부모의 멤버변수와 멤버메서드를 자식에게 상속한다.

부모

```
public class Unit {  
  
    int hp=40;  
    static int power=4;  
    static int armor=0;  
  
}
```



자식

```
2  
3 public class Marine extends Unit {  
4  
5     String name;  
6  
7  
8  
9 Marine()  
0 {  
1     hp = 50;  
2 }  
3
```

선언하지 않아도 사용가능

```

public class Parent {
    String name;
    int age;

    Parent()
    {
    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    void showState()
    {
        System.out.println("이름: " + name + "나이: " + age);
    }
}

```

```

public class Child extends Parent{
}

```

```

public static void main(String[] args) {

    Child child = new Child();

    child.name = "자식1";
    child.age = 10;
    child.showState();

}

```

→ 상속받은 멤버변수와 메서드

자식에서 만들지 않아도 상속받아 마치 선언 해놓은 것처럼 사용한다.


```

public static void main(String[] args) {

    Parent parent = new Parent("부모1",45);

    parent.showState();

    Child child = new Child();

    child.showState();
    |

}

```

Parent
name
age



Child
name
age

```

<terminated> main [Java Application]

```

```

이름: 부모 나이: 45

```

```

이름: null 나이: 0

```

부모의 멤버를 가져오는것이 아니라 부모와 별도로 멤버를 생성하는것이다.

```

public class Parent {

    String name;
    int age;

    Parent()
    {

    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    void showState()
    {
        System.out.println("이름: " + name + " 나이: " + age);
    }

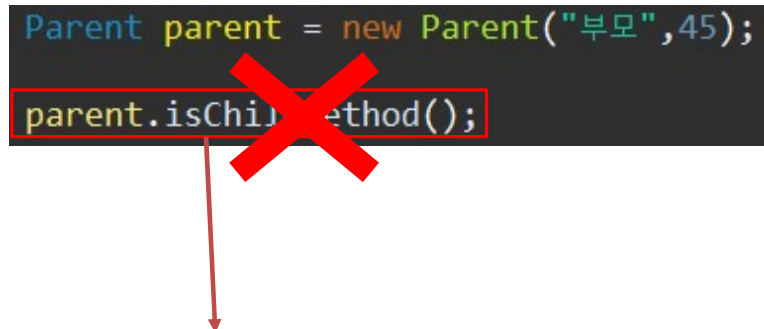
}

```

```

Parent parent = new Parent("부모", 45);
parent.isChildMethod();

```



자식에서 선언한 멤버들은
부모가 사용 할 수 없다.

```

3 public class Child extends Parent{
4
5
6 void isChildMethod()
7 {
8     |
9 }
10
11 }

```

자식에서 만든 멤버 메서드

```

public class Parent {

    String name;
    int age;

    Parent()
    {

    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    void showState()
    {
        System.out.println("이름: " + name + " 나이: " + age);
    }

}

```


```

public static void main(String[] args) {

    Child child = new Child("자식", 10);

}

```



생성자는 상속되지 않는다.

```

3 public class Child extends Parent{
4
5
6     void isChildMethod()
7     {
8
9     }
10
11 }

```

부모의 생성자

```
public class Parent {  
  
    String name;  
    int age;  
  
    Parent(String name, int age)  
    {  
        this.name = name;  
        this.age = age;  
    }  
  
    void showState()  
    {  
        System.out.println("이름: " + name + " 나이: " + age);  
    }  
}
```

기본생성자(디폴트생성자)가 없다.

```
public class Child extends Parent{  
  
    void isChildMethod()  
    {  
    }  
}
```

자식 또한 디폴트 생성자를 만들어주지 않는다.

부모의 생성자

```
3 public class Child extends Parent{  
4  
5  
6  
7     Child()  
8     {  
9  
10    }  
11  
12    void isChildMethod()  
13    {  
14  
15    }  
16 }  
17 }
```

명시적으로 선언하여 쓸 수도 없다.

자세한 이유는 super에서 학습

```
public class GrandParent {

    String name;
    int age;

    void showState()
    {
        System.out.println("이름: " + name + " 나이: " + age);
    }
}
```

```
public class Child extends Parent{

    void ageUp()
    {
        age++;
    }
}
```

```
public class Parent extends GrandParent{

    Parent()
    {
    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

상속은 무한히 내려갈수 있다.

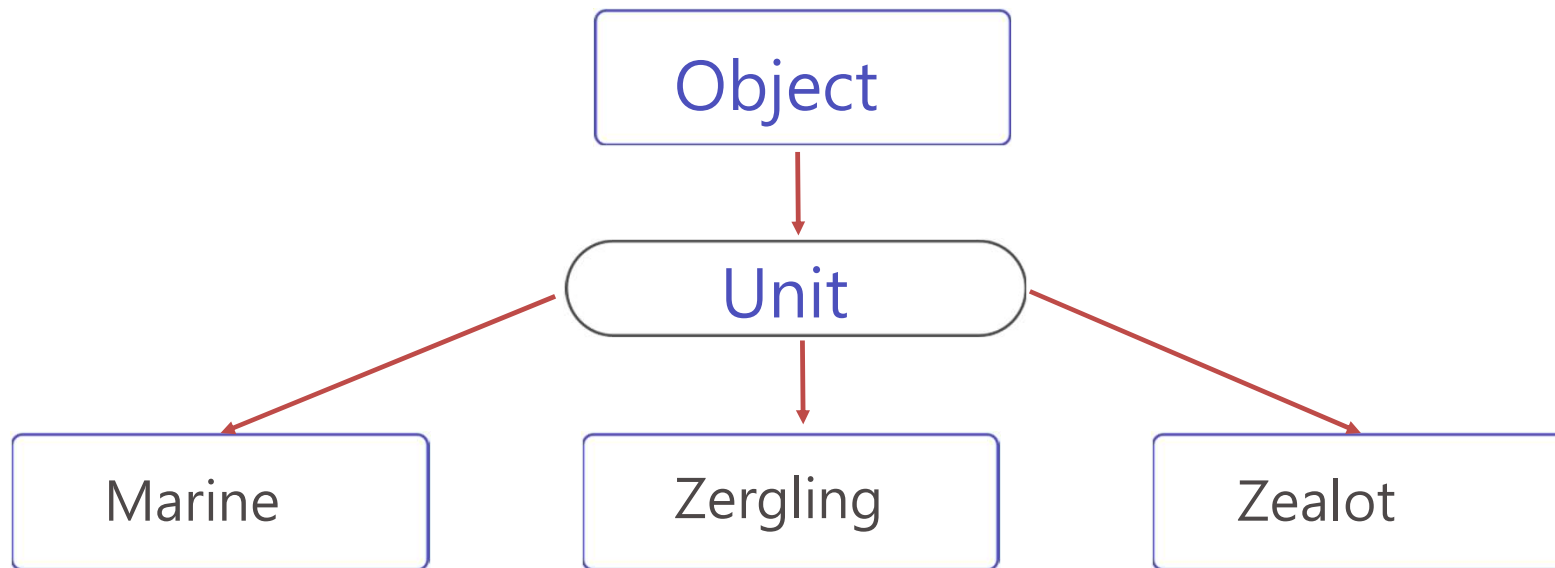
단일 상속

```
public class Child extends Parent,Marine{  
    void ageUp()  
    {  
        age++;  
    }  
}
```

자바는 복잡한 클래스관계를 막기 위해 다중 상속 안됨

Object 클래스

1. 모든 클래스의 부모
2. toString, equal과 같이 클래스에 기본적으로 필요한 메서드의 틀을 가지고 있다.
3. 모든 객체는 Object로 형변환이 가능하다.



Object.toString() 메서드

- 객체의 상태값을 String 타입으로 반환한다.
- 기본적으로는 "패키지.클래스명@객체주소" 형태로 반환한다.



기본적으로 가지고 있다.

```
GrandParent gp = new GrandParent();  
System.out.println(gp.toString());
```

```
<terminated> main [Java Application] C:\#User  
joo.GrandParent@27d415d9
```

패키지

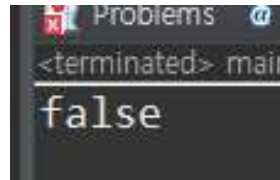
클래스명

객체주소

Object.equals() 메서드

- 매개변수로 객체를 받으며 equals를 호출하는 객체와 매개변수로 들어온 객체가 같으면 true를 반환한다.

```
public static void main(String[] args) {  
  
    GrandParent gp = new GrandParent();  
    GrandParent gp2 = new GrandParent();  
  
    gp.name = "할아버지";  
    gp2.name = "할아버지";  
  
    System.out.println(gp.equals(gp2));  
    |  
}
```



Problems @
<terminated> main
false

왜 false 가 나올까?

gp

0x000A

객체주소	이름
name	할아버지

equals()

gp2

0x300B

객체주소	이름
name	할아버지

```

GrandParent gp = new GrandParent();
GrandParent gp2 = new GrandParent();

gp.name = "할아버지";
gp2.name = "할아버지";

System.out.println(gp);
System.out.println(gp.toString());

System.out.println(gp2);
System.out.println(gp2.toString());

```

```

joo.GrandParent@27d415d9
joo.GrandParent@27d415d9
joo.GrandParent@5c18298f
joo.GrandParent@5c18298f

```

```

public static void main(String[] args) {

    GrandParent gp = new GrandParent();
    GrandParent gp2 = new GrandParent();

    gp.name = "할아버지";
    gp2.name = "할아버지2";

    gp2 = gp;
    System.out.println(gp.equals(gp2));

}

```

true

object.equals은 주소를 비교한다.

포함관계

- A클래스에서 B클래스타입의 멤버변수를 가지고 있으면 A가 B를 포함한다.

```
public class Parent {  
  
    GrandParent gp;  
    Parent()  
    {  
  
    }  
  
    Parent(String name, int age)  
    {  
        gp.name = name;  
        gp.age = age;  
    }  
}
```

```
Parent pt = new Parent();  
  
pt.gp = new GrandParent();  
  
pt.gp.name = "할아버지";  
pt.gp.age = 70;
```

상속과 마찬가지로 멤버변수, 메서드 등을 사용 할 수 있는데 똑같은거 아닌가?



is a(상속)

- 상속으로 표현한다.
- 같은 범주에 속한다.
- 차, 전기차 와의 관계

has a(포함)

- 멤버변수로 표현한다.
- 소유나 일부분을 나타낸다
- 차, 핸들,문 과의 관계

포함관계(has a)

```
public class Car {  
  
    int speed;  
  
    public Door doors[] = new Door[4];  
}
```



차와 문은 같은가? X

차는 문을 소유 하는가? O -> 포함관계로 표현한다.

상속관계(is a)

```
public class OilCar extends Car{  
    public int Oil;  
  
    public OilCar()  
    {  
          
    }  
}
```

차와 기름차는 같은가? O -> 상속관계
차는 기름차를 소유 하는가? X

1-1 실습 문제 (normal)

Car, OilCar 클래스를 만들어 상속관계를 만들어 보자.

- 멤버변수 : int Oil , int speed
(Oil은 모든 차량이 다 있어야 하는 변수인가?.. 전기차라면..?)

```
public class Main {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        OilCar car = new OilCar();  
        car.speed = 100;  
        car.Oil = 100;  
        System.out.println(car.speed);  
        System.out.println(car.Oil);  
    }  
}
```

Problems

<terminated>

100

100

1-1 문제 해설

Park Ju Byeong

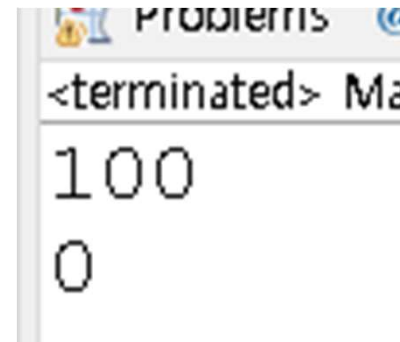
Park Ju Byeong

1-2 실습 문제(normal)

1-1문제에서 만든 Car, OilCar 클래스에 기능을 추가해보자.

- 멤버메서드 : void go(int speed) 매개변수로 받은 속도를 멤버변수에 저장한다.
void stop() 속도를 0으로 만든다.
- go,stop() 메서드는 어느 클래스에서 만들어야 할까?

```
public class Main {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        OilCar car = new OilCar();  
  
        car.go(100);  
        System.out.println(car.speed);  
  
        car.stop();  
        System.out.println(car.speed);  
  
    }  
}
```



1-2 문제 해설

Pa.

yeong

Park Ju b

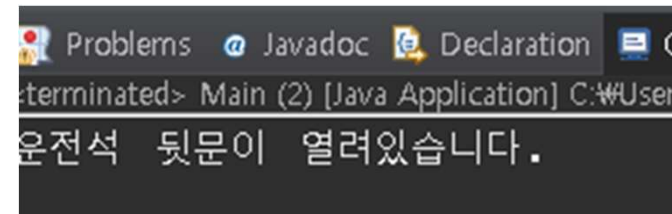
1-3 실습 문제(hard)

Door 클래스를 만들어 Car 클래스와 포함관계를 만들어보자.

- 문 개수는 4개이다.(Car는 여러 개의 문을 가지므로 객체배열을 이용하자)

```
OilCar car = new OilCar();  
  
car.doors[0].name = "운전석";  
car.doors[1].name = "조수석";  
car.doors[2].name = "운전석 뒷문";  
car.doors[3].name = "조수석 뒷문";  
  
car.doors[2].open();
```

클래스명	Door	
멤버변수	bool isOpen	문 열림 여부
	String name	ex)운전석, 조수석, 운전석 뒷문, 조수석 뒷문
메서드	void open()	문을 연다
	void close()	문을 닫는다.



Problems Javadoc Declaration C
terminated> Main (2) [Java Application] C:\#User
운전석 뒷문이 열려있습니다.

1-3 문제 해설

1-3 문제 해설

1-4 실습 문제(hard)

ElectricCar, HibrideCar 클래스를 만들고 둘 다 int battery 를 가지도록 하자

- battery를 전기,하이브리드 둘 다 선언하면 코드 중복이다.
- Car 클래스에 선언하면 OilCar 역시 배터리를 가지게 된다
(기름차 역시 현실에선 배터리가 있지만 없다고 가정하자)
- 둘 다 void Charge(int power) 메서드를 가지고 충전 할 수 있어야 한다.

```
ElectricCar car = new ElectricCar();
HibrideCar car2 = new HibrideCar();

car.battery= 50;
car.Charge(30);

car2.battery = 20;
car2.Charge(50);

System.out.println("현재 배터리량:"+car.battery);
System.out.println("현재 배터리량:"+car2.battery);
```


1-4 문제 해설

1-4 문제 해설

1-4 문제 해설

— 02

오버라이딩

메서드 오버라이딩

- 부모로부터 상속받은 메서드를 재정의 한다.

```
2  
3 public class Parent extends GrandParent{  
4  
5     String name = "부모";  
6  
7  
8  
9     void parentMethod()  
0     {  
1         System.out.println("부모 메서드");  
2  
3     }  
4
```



```
public class Child extends Parent{  
  
    String name = "자식";  
  
    void parentMethod()  
    {  
        System.out.println("자식 메서드");  
    }  
}
```

부모 메서드의 시그니처 및
반환타입까지 모두 일치해야
오버라이딩이 가능하다.

메서드 오버라이딩의 필요성

```
2
3 public class GrandParent {
4
5     String name;
6     int age;
7
8     void showState()
9     {
10         System.out.println("이름: " + name + " 나이: " + age);
11     }
12
13 }
```

자식에서 영문으로 바꾸고 싶다면?

```
public class Parent extends GrandParent{

    String name;

    Parent()
    {

    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    void showState() {

        System.out.println("name: " + name + " age: " + age);

    }

}
```

메서드 오버라이딩 - 공변반환타입 (심화)

- 오버라이딩은 반환타입까지 일치해야한다.
- 반환타입이 부모에서 자식으로 변경되는경우 공변반환타입이라고 하며 오버라이딩으로 인정해준다.

```
public class GrandParent {  
  
    String name;  
    int age;  
  
    GrandParent getInstance()  
    {  
        return this;  
    }  
}
```



```
public class Parent extends GrandParent{  
  
    String name = "부모";  
  
    Parent getInstance()  
    {  
        return this;  
    }  
}
```

static 과 instance 메서드간의 변환

- 상속받을시 static을 instance로 혹은 그반대로 바뀌서 오버라이딩은 불가능하다.

```
public class GrandParent {  
  
    String name;  
    int age;  
  
    void showState()  
    {  
        System.out.println("이름: " + name + " 나이: " + age);  
    }  
  
    static void print()  
    {  
        System.out.println("test");  
    }  
  
    @Override  
    public String toString() {  
        return "name: " + name + "age: " + age ;  
    }  
}
```

```
public class Parent extends GrandParent {  
  
    String name;  
  
    Parent()  
    {  
    }  
  
    Parent(String name, int age)  
    {  
        this.name = name;  
        this.age = age;  
    }  
  
    void print()  
    {  
        System.out.println("test");  
    }  
}
```


Object.toString() 메서드의 오버라이딩

- Object 클래스는 따로 상속받지 않아도 기본적으로 모든 클래스가 상속받는다.
- Object로 부터 상속받은 메서드들을 오버라이딩 할 수 있다.

```
2
3 public class GrandParent {
4
5
6     String name;
7     int age;
8
9     void showState()
10    {
11        System.out.println("이름: " + name + " 나이: " + age);
12    }
13
14    |
15    @Override
16    public String toString() {
17
18        return "name: " + name + "age: " + age ;
19    }
20 }
```

일반적으로 그대로 쓰기보단
오버라이드하여 멤버변수의 값을 보여준다.

상속시 부모의 메서드를
덮어쓴다.

메서드를 **증설한다.**

시그니처가 동일하다

시그니처가 다르다.

오버라이딩

오버로딩

퀴즈

```
public class Parent extends GrandParent{
    String name;

    Parent()
    {
    }

    Parent(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    Parent parentMethod()
    {
        return new Parent();
    }
}
```

```
Parent parentMethod(int a)
{
    return new Parent();
}
```

→ 오버로딩

```
void parentMethod()
{
}
```

→ ERROR!

```
public class Child extends Parent{

    Parent ParentMethod()
    {
        return new Parent();
    }
}
```

→ 오버라이딩

```
Child ParentMethod()
{
    return this;
}
```

→ 공변반환타입 OK!
(오버라이딩)

```
void ParentMethod(int a)
{
}
```

→ 오버로딩

```
int parentMethod()
{
    return 0;
}
```

→ 상속받은 메서드와 중복
ERROR!

멤버변수 오버라이딩

- 부모가 상속해준 멤버변수 역시 오버라이딩 가능하다.
- 이미 가지고 있는 변수인데 오버라이딩 하는게 무슨 의미가 있을까?(super에서 자세히 보도록 한다)

```
public class GrandParent {  
  
    String name;  
    int age;  
  
    void showState()  
    {  
        System.out.println("이름: " + name + " 나이: " + age);  
    }  
  
    @Override  
    public String toString() {  
        return "name: " + name + "age: " + age ;  
    }  
}
```

```
GrandParent gp = new GrandParent();  
Parent parent = new Parent();  
gp.name = "할아버지";  
parent.name = "아버지";
```

```
public class Parent extends GrandParent {  
  
    String name;  
    Parent()  
    {  
    }  
  
    Parent(String name, int age)  
    {  
        this.name = name;  
        this.age = age;  
    }  
}
```

이미 있지만 다시 재정의



03

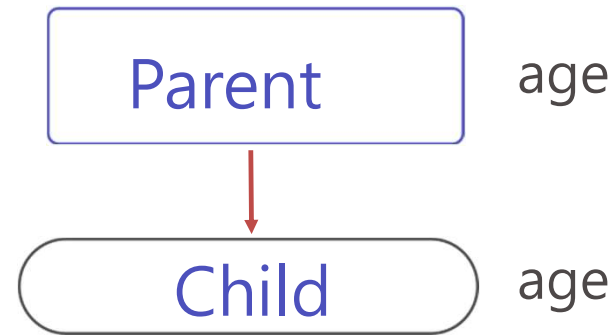
super

super

부모로부터 물려 받은 멤버변수, 메서드를 가리킬때 사용 한다.

절대 부모객체를 가리키는것이 아니다

super



super

```
class Parent1
{
    int age = 50;
}
```

```
public static void main(String[] args) {
    // TODO Auto-generated method stub

    Child1 child = new Child1();
}
```

```
class Child1 extends Parent1
{
    Child1()
    {
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```

```
<terminated> Main (2) [
50
50
50
```

부모로부터 상속받은 age 변수를 가리킨다.

super

- 일반적인 상황에서 super는 그냥 멤버변수를 가리킨다.

```
class Child1 extends Parent1
{

    Child1()
    {
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);

        System.out.println(System.identityHashCode(age));
        System.out.println(System.identityHashCode(this.age));
        System.out.println(System.identityHashCode(super.age));
    }
}
```

```
<terminated> Main (2) [Java Applica
50
50
50
1586600255
1586600255
1586600255
```

super

- 부모의 멤버변수와 상속받은 자식의 멤버변수는 다르다.

```
Parent1 parent = new Parent1();  
parent.age = 80;
```

```
Child1 child = new Child1();
```

Parent

age = 80

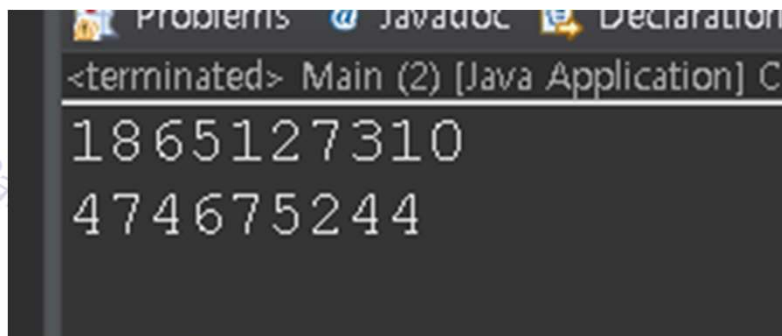
Child

age = 50

super

- 주소값이 다른걸 확인 할수 있다.

```
Parent1 parent = new Parent1();  
parent.age = 80;  
  
System.out.println(System.identityHashCode(parent.age));  
Child1 child = new Child1();  
  
System.out.println(System.identityHashCode(child.age));
```



```
<terminated> Main (2) [Java Application] C  
1865127310  
474675244
```

this는 지역변수와 멤버변수를 구분하는 용도로 쓸 수 있는데

super는 왜 필요하지..?

그냥 멤버 변수 쓰거나 this.멤버변수 쓰면 똑같은거 아닌가?

super - 멤버변수 오버라이딩

- 오버라이딩을 하게 되면 일반적인 멤버변수와 super를 통해 접근하는게 주소가 다른걸 확인 할 수 있다.
- super는 부모가 **상속해준 원본**을 가리킬 수 있다.

```
1
2
3 class Parent1
4 {
5     int age = 50;
6 }
7
8 class Child1 extends Parent1
9 {
10     int age;
11
12     Child1()
13     {
```

오버라이딩

```
class Child1 extends Parent1
{
    Child1()
    {
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);

        System.out.println(System.identityHashCode(age));
        System.out.println(System.identityHashCode(this.age));
        System.out.println(System.identityHashCode(super.age));
    }
}
```

<terminated> Main (10) [Java]

0
0
50
1586600255
1586600255
474675244

super

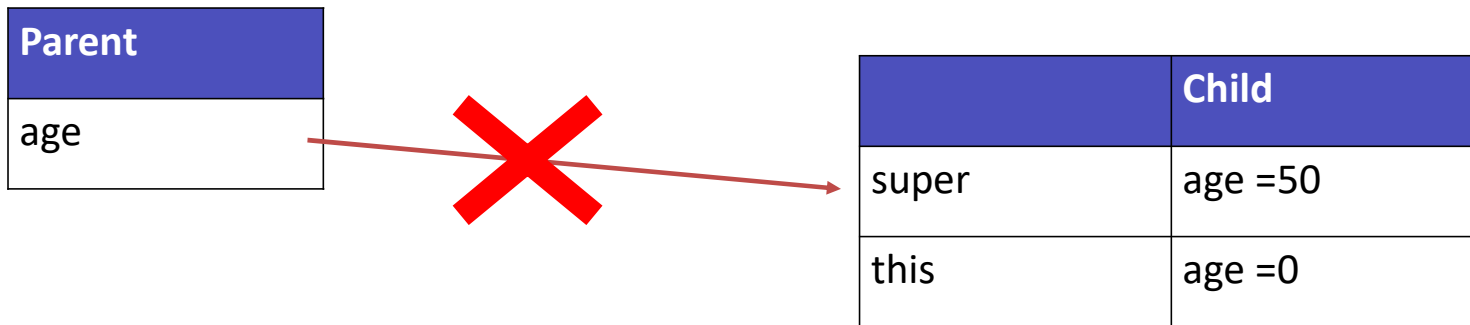
```
System.out.println(age);  
System.out.println(this.age);  
System.out.println(super.age);
```

```
System.out.println(System.identityHashCode(age));  
System.out.println(System.identityHashCode(this.age));  
System.out.println(System.identityHashCode(super.age));
```

```
System.out.println(this);  
System.out.println(super.toString());
```

```
<terminated> Main (10) [Java Application] C:\Use  
0  
0  
50  
1586600255  
1586600255  
474675244  
joo.강의9.Child1@379619aa  
joo.강의9.Child1@379619aa
```

super는 this와 똑같이 자기자신의 객체 주소값을 가지고 있다. 즉 부모의 객체를 가리키는것이 아니다. 부모객체는 생성조차 안했다.



부모 객체와는 아무 관계 없다.

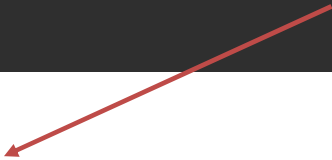
super.age => 상속받은 원본

age , this.age => 오버라이딩한 자식 고유의 age

super - 부모메서드 오버라이딩

```
2
3 public class Parent {
4
5     String name = "부모";
6
7
8     public String toString()
9     {
10         return "이름: " + name;
11     }
12
13 }
```

```
public class Child extends Parent{
    int age;
    public String toString()
    {
        return "이름: " + name + " 나이: " + age;
    }
}
```



자식에서 나이 출력만 추가 하고 싶은데 부모가 만들어놓은기능을 다시 만들어야 한다.

부모로부터 상속받은 메서드를 재사용할수 없을까?

super - 부모메서드 오버라이딩

- super는 부모가 상속시켜준 원본 멤버변수와 메서드를 가리킬수 있다.
- 오버라이딩 하여도 원본은 감춰져있을뿐 유지가된다

```
2
3 public class Parent {
4
5     String name = "부모";
6
7
8     public String toString()
9     {
10         return "이름: " + name;
11     }
12
13 }
```

```
public class Child extends Parent{
    int age;
    public String toString()
    {
        return super.toString() + " 나이: " + age;
    }
}
```

부모가 상속시켜준 원본 toString()

super생성자

```
public class Parent {  
  
    String name = "부모";  
  
    Parent()  
    {  
  
    }  
  
    Parent(String name)  
    {  
        this.name = name;  
    }  
}
```

```
3 public class Child extends Parent{  
4  
5  
6     int age;  
7  
8     Child()  
9     {  
10  
11     }  
12  
13  
14     Child(String name , int age)  
15     {  
16         this.name = name;  
17         this.age = age;  
18     }  
19
```

```
Child(String name , int age)  
{  
    super(name);  
    this.age = age;  
}
```

생성자는 상속은 안되지만 super를 이용해 부모의 생성자를 사용할수 있다.

super 안 쓰고 그냥 직접 초기화 하면 안되나?

1. 초기화 코드의 중복
2. class 라이브러리만 가져 올 경우 생성자내부를 확인 해볼 수 없다.
3. 어떤 필터링을 거치는지 확인 할 수 없다.
4. 부모의 생성자가 변경되면 같이 변경해줘야 한다.
5. 부모의 생성자가 길어지면 가독성이 떨어진다.

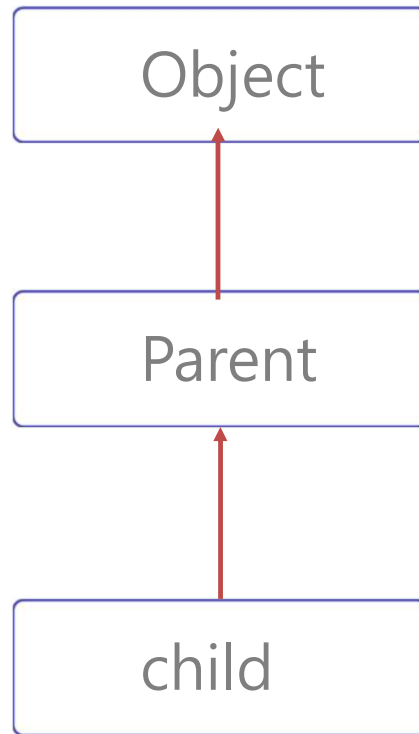
```
Child(String name , int age)
{
    this.age = age;
    super(name);
}
```

→ 항상 제일 먼저 수행 되어야 한다.

```
Child()
{
}

Child(String name , int age)
{
    this.age = age;
}
```

← 만약 super 생성자를 사용하지 않으면
컴파일러가 자동으로 끼워 넣는다.



모든 클래스들은 객체 생성시 Object 생성자를 호출한다.

퀴즈

- Child클래스의 생성자가 에러가 발생하는 이유는?

```
2  
3 public class Parent {  
4  
5  
6     String name = "부모";  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

```
public class Child extends Parent {  
    int age;  
  
    Child()  
    {  
    }  
  
    Child(String name , int age)  
    {  
        super(name);  
        this.age = age;  
    }  
}
```

super()가 자동삽입 되어야 하나 부모의 디폴트 생성자가 없다.

2-1 실습문제(normal)

1. 오버라이딩의 조건으로 옳지 않은것은?
 1. 부모의 메서드와 이름이 같아야 한다.
 2. 매개변수의 수와 타입이 모두 같아야 한다.
 3. 반환타입이 부모인 메서드를 자식의 타입으로 변경
 4. 반환 타입은 달라도 된다.

2-2 실습문제(normal)

- Tv클래스의 생성자가 오류가 발생하는 이유는?

```
class Product
{
    int price;
    Product(int price)
    {
        this.price = price;
    }
}

class Tv extends Product
{
    Tv()
    {
    }
}
```

2-3 실습문제(hard)

도형을 의미 하는 Shape 클래스와 Circle, Rectangle 클래스를 만드시오

클래스간의 상속관계와 멤버변수, 멤버 메서드를 적절한 클래스에 넣어 설계해보자

-Shape 도형 , Circle 원 , Rectangle 사각형

멤버변수 double r 반지름

double width 폭

double height 높이

멤버메서드 double getArea() 해당 도형의 면적을 반환한다.

삼각형 = $PI \times \text{반지름} \times \text{반지름}$ 사각형 = 가로 \times 세로

boolean isSquare() 정사각형이면 true를 반환한다.

```
Circle circle = new Circle(5);
Rectangle rectangle = new Rectangle(15, 15);

System.out.println("반지름 5의 원의 면적: "+circle.getArea());

System.out.println("네모 면적:"+rectangle.getArea());

if(rectangle.isSquare())
    System.out.println("정사각형 입니다.");
```

<terminated> Main (2) [Java Application] C:\Users\USER545\p2\pool\plugins

반지름 5의 원의 면적: 78.53981633974483

네모 면적: 225.0

정사각형 입니다.

Park Ju Byeong

2-3 문제 풀이

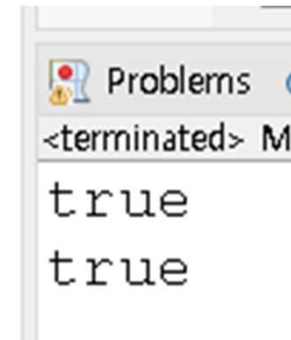
2-4 실습문제(hard)

equals 메서드를 오버라이딩 하여 면적이 같으면 true가 반환되도록 하자.

- 사각형도 면적이 같다면 true이다
- object 클래스의 equals 메서드의 원형은 아래와 같다.

```
public boolean equals(Object obj)
```

```
Circle circle = new Circle(5);  
Circle circle2 = new Circle(5);  
System.out.println(circle.equals(circle2));  
  
Rectangle rectangle = new Rectangle(15, 15);  
Rectangle rectangle2 = new Rectangle(15, 15);  
System.out.println(rectangle.equals(rectangle2));
```



2-4 문제풀이

Park

, k Ju Byeong



THANK YOU



강사 박주병