

능동적 사고 방식의

java

강사 박주병



Part09 객체지향2 .

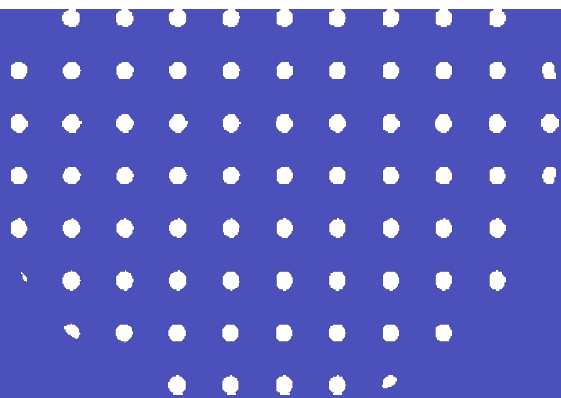
01 상속

02 오버라이딩

03 super 생성자

04 실습 문제





ParkJuByeong

01

상속



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;
```

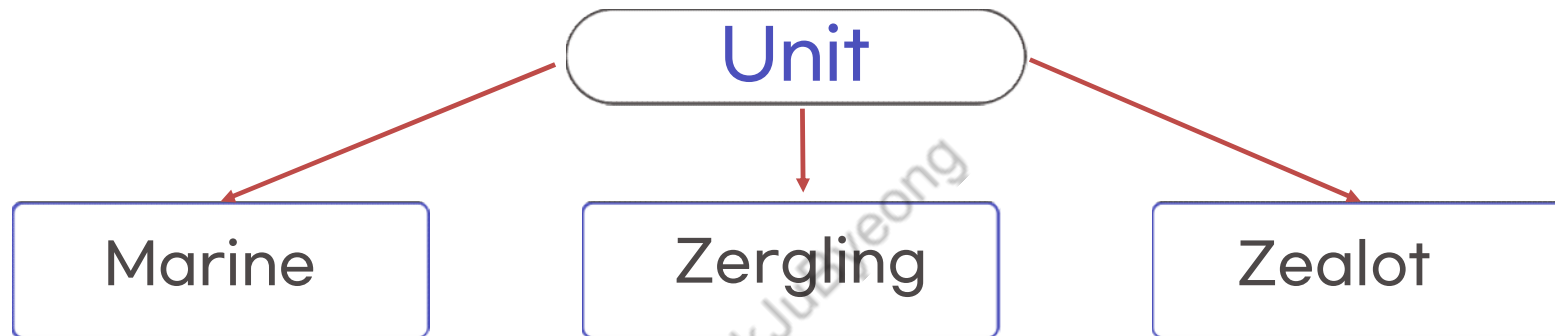
```
boolean attack(Zergling target)  
{  
    target.hp -= (power - target.armor);  
    return target.hp <= 0;  
}  
  
void showState()  
{  
    System.out.println("체력: "+hp+"\t 공격력: "+power + "\t 방어력: "+armor);  
}  
  
void powerUp()  
{  
    power++;  
}
```

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

그외 유닛들 ...

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

```
boolean attack(Character target)  
{  
    target.hp -= (power - target.armor);  
    return target.hp <= 0;  
}
```



상속

부모

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



자식

```
public class Marine extends Unit{  
  
    String name;  
  
    Marine()  
    {  
  
    }  
}
```

1. 초기화 블록은 상속되지 않는다.
2. 멤버변수, 멤버메서드가 상속된다.

```
public class Marine extends Unit{  
  
    String name;  
  
    Marine()  
    {  
        hp = 50;  
    }  
}
```

```

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();

}

```

```

<terminated> main [Java Application]
이름: 자식1나이: 10

```

자식에서 만들지 않아도 상속받아 마치 선언해놓은 것처럼 사용한다.

```

public static void main(String[] args) {

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

    parent.showState();

    Child child = new Child();

    child.showState();
    |
}

```

```

<terminated> main [Java Application]

```

```

이름: 부모 나이: 45

```

```

이름: null 나이: 0

```

Parent
name
age



Child
name
age

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);
    }

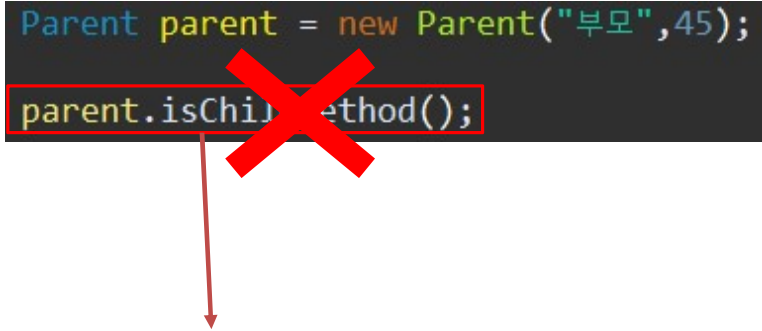
}

```

```

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

```



자식에서 생성된 멤버들은
부모에 영향을 주지 않는다.

```

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

```

```

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 }

```

```
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);  
    }  
}
```

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

부모가 디폴트생성자가 없다면 자식또한 디폴트
생성자를 만들어주지 않는다.

```
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 쓸수도 없다.

부모가 디폴트 생성자를 만들지 않았다면
자식또한 디폴트 생성자를 허용하지 않는다.

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

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

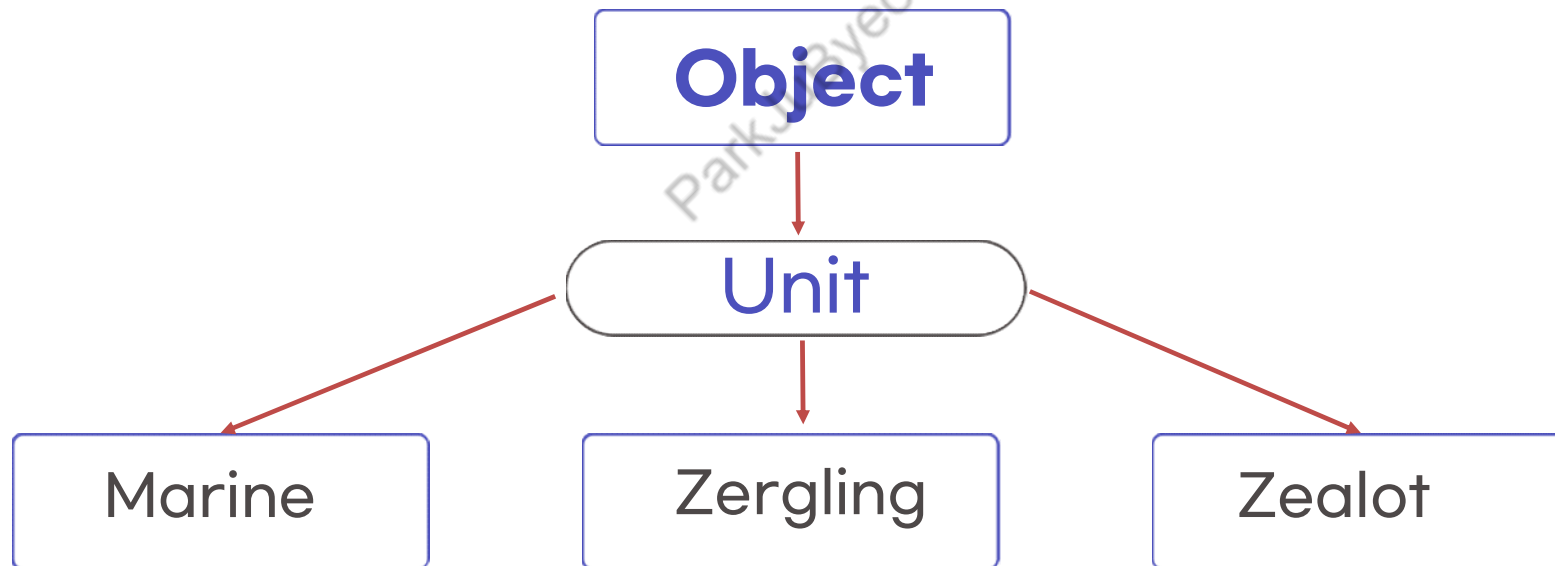
```
public class Parent extends GrandParent{  
    Parent()  
    {  
    }  
  
    Parent(String name, int age)  
    {  
        this.name = name;  
        this.age = age;  
    }  
}
```

ParkJuByeong

상속은 무한히 내려갈수 있다.

Object 클래스

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



Object.toString()

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

```
7 public static void main(String[] args) {  
8  
9  
10 GrandParent gp = new GrandParent();  
11  
12 gp.  
13  
14  
15  
16  
17 }  
18  
19  
20  
21 }  
22
```

- age : int - GrandParent
- name : String - GrandParent
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- showState() : void - GrandParent
- toString() : String - Object
- wait() : void - Object
- wait(long timeoutMillis) : void - Object
- wait(long timeoutMillis, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

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

패키지

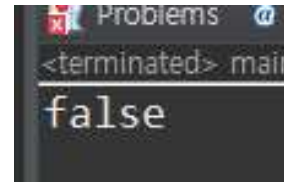
클래스명

객체주소

Object.equal()

```
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));  
  
    |  
  
}
```

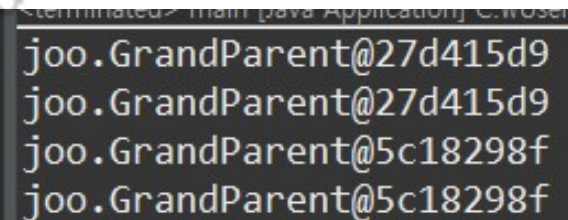
```
GrandParent gp = new GrandParent();  
GrandParent gp2 = new GrandParent();  
  
gp.name = "할아버지";  
gp2.name = "할아버지";  
  
System.out.println(gp);  
System.out.println(gp.toString());  
  
System.out.println(gp2);  
System.out.println(gp2.toString());
```



Problems

<terminated> main

false



<terminated> main [Java Application] C:\User

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

단일상속

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

자바는 복잡한 클래스관계를 막기 위해 다중 상속 안됨

포함관계

```
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 사용할 수 있는데 똑같은거 아닌가?



is a

- 상속으로 표현한다.
- 같은 범주에 속한다.
- 차, 전기차와의 관계

has a

- 멤버변수로 표현한다.
 - 소유나 일부분을 나타낸다
 - 차, 핸들,문과의 관계
- 

실습문제1

1. Car, OilCar 클래스를 만들어 상속관계를 만들어 보자.
OilCar는 기름량을 저장 할수 있어야 한다.

```
OilCar car = new OilCar();  
car.go(100);  
System.out.println(car.speed);  
car.stop();  
System.out.println(car.speed);
```

```
<terminate>  
100  
0
```

클래스명	Car	
멤버변수	int speed	속도
메서드	void go(int speed)	
	void stop()	속도를 0으로 만든다.

2. Door 클래스를 만들어 Car 클래스와 포함관계를 만들어보자.
문 개수는 4개이다.

```
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()	
	void close()	

3. ElectricCar, HibrideCar 클래스를 만들고 둘다 int battery 를
가지도록 하자

*battery를 전기,하이브리드 둘다 선언하면 코드 중복이다.
*Car 클래스에 선언하면 OilCar 역시 배터리를 가지게 된다
(기름차 역시 현실에선 배터리가 있지만 없다고 가정하자)

```
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);
```

02

오버라이딩

ParkJuByeong

오버라이딩

```
2  
3 public class Parent extends GrandParent{  
4  
5  
6     String name = "부모";  
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");  
    }  
}
```

부모로부터 물려 받은 멤버변수, 메서드를 자식이 새롭게 덮어쓰는것


```

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;
    }

}

```

멤버변수의 오버라이딩은 무슨 의미가 있나? 오버라이딩 안해도 있는데..?

super 변수를 배울때 자세히 알아보자!

메서드 오버라이딩

```
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);

    }

}
```

메서드의 이름, 매개변수, 리턴타입이 일치해야 한다

공변반환타입

```
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    GrandParent getInstance()
15    {
16        return this;
17    }
18 }
```

```
• Parent getInstance() {
    // TODO Auto-generated method stub
    return this;
}
```

리턴이 부모타입일때 자식의 타입으로 변환하여도 오버라이딩으로 인정

만약 오버라이딩으로 인정하지
않으면 어떻게 될까?

```
Parent getInstance() {  
    // TODO Auto-generated method stub  
    return this;  
}
```

```
Parent parent = new Parent();  
Parent temp = (Parent)parent.getInstance();
```

→ 사용시 형변환을 명시적으로 해줘야되서 불편하다.

오버라이딩이 아니라면 함수 이름과 매개변수가 같기에 오버로딩으로 분류된다. 그러나 오버로딩은 리턴타입을 고려하지 않기에 결국 메서드 중복정의로 해당메서드를 문법적으로 만들 방법이 없다.

```

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");
    }
}

```

static <-> 인스턴스 메서드 의 변환은 불가능하다.

toString() 오버라이딩

```
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.

Tech
01

상속시 부모의 메서드를
덮어쓴다.

Tech
02

메서드를 증설한다.

Tech
03

시그니처가 동일하다

Tech
04

시그니처가 다르다.

오버라이딩

오버로딩

ParkJuByeong

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

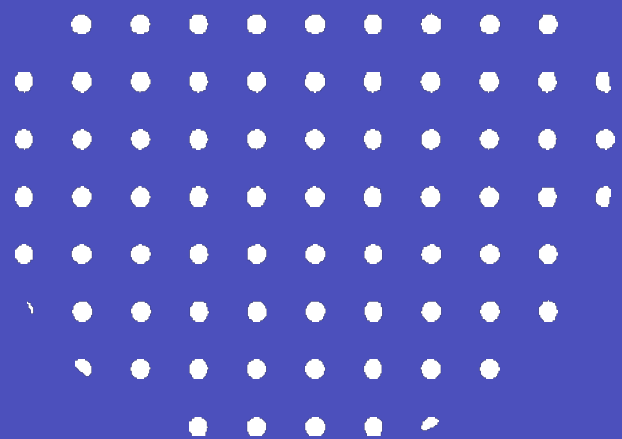
```
/*  
 * Parent parentMethod()  
 * {  
 *     return new Parent();  
 * }  
 */  
  
int parentMethod()  
{  
    return 0;  
}
```

→ 상속받은 메서드와 중복
ERROR!

실습문제2

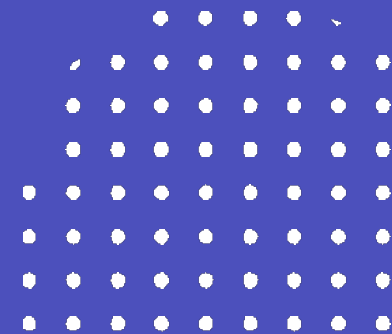
1. 상속시 부모의 메서드를 덮어쓴다.

ParkJuByeong



ParkJuByeong

03
super



부모로부터 물려 받은
멤버변수, 메서드를 가리킬때 사용 한다.

절대 부모객체를 가리키는것이 아니다

```

3 public class Parent extends GrandParent{
4
5
6     String name = "부모";
7
8     Parent()
9     {
10
11     }

```

```

1 public class Child extends Parent{
2
3     void test()
4     {
5
6         System.out.println(name);
7         System.out.println(this.name);
8         System.out.println(super.name);
9     }

```

```

1 public static void main(String[] args) {
2
3     Child child = new Child();
4
5     child.test();
6
7 }

```

Problems @

<terminated> main

부모
부모
부모

```
void test()
{
    System.out.println(name);
    System.out.println(this.name);
    System.out.println(super.name);

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

Problems JavaDoc
<terminated> main [Java App

부모
부모
부모
838411509
838411509
838411509

```

public static void main(String[] args) {

    Parent parent = new Parent();

    parent.name = "부모";

    Child child = new Child();
    child.name = "자식";

    child.test();
}

```

Parent
부모

Child
자식

```

void test() |
{

    System.out.println(name);
    System.out.println(this.name);
    System.out.println(super.name);

    System.out.println(System.identityHashCode(name));
    System.out.println(System.identityHashCode(this.name));
    System.out.println(System.identityHashCode(super.name));

}

```

```

<terminated> main [Java
자식
자식
자식
838411509
838411509
838411509

```

```
public class Parent extends GrandParent{  
  
    String name = "부모";  
}
```

```
public class Child extends Parent{  
  
    String name = "자식";  
  
    void test()  
    {  
  
        System.out.println(name);  
        System.out.println(this.name);  
        System.out.println(super.name);  
  
        System.out.println(System.identityHashCode(name));  
        System.out.println(System.identityHashCode(this.name));  
        System.out.println(System.identityHashCode(super.name));  
  
        System.out.println(this);  
        System.out.println(super.toString());  
    }  
}
```

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

```
    Child child = new Child();  
  
    child.test();  
}
```

```
<terminated> main [Java Application] C:\
```

자식

자식

부모

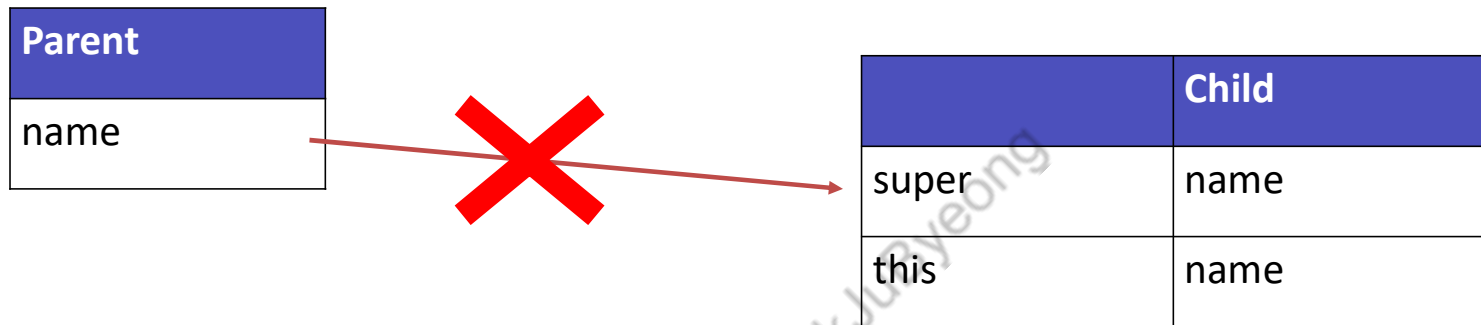
838411509

838411509

1434041222

joo.Child@5204062d

joo.Child@5204062d



부모 객체와는 아무 관계 없다.

부모 메서드 오버라이딩

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

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

부모로부터 상속받은 메서드를 재사용할수 없을까?

super() 생성자

```
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;
    }
}
```

```
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;  
}
```

1. 초기화 코드의 중복
2. class 라이브러리만 가져올경우 생성자내부를 확인해볼수 없다.

super 안 쓰고 그냥 직접 초기화 하면 안되나?

3. 어떤 필터링을 거치는지 확인할수 없다.
4. 부모의 생성자가 변경되면 같이 변경해줘야 한다.
5. 부모의 생성자가 길어지면 가독성이 떨어진다.

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

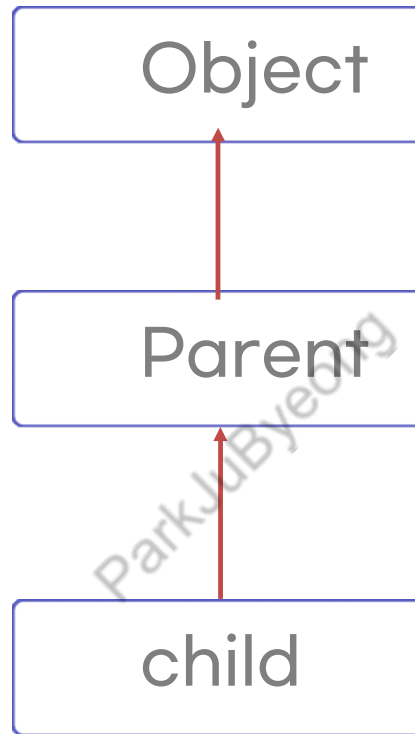
→ 항상 제일 먼저 수행 되어야 한다.

```
Child()
{
}

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

만약 super 생성자를 사용하지 않으면
컴파일러가 자동으로 끼워넣는다.

ParkJuByeong



모든 클래스들은 객체 생성시 Object 생성자를 호출한다.


```

2
3 public class Parent {
4
5
6     String name = "부모";
7
8
9
10 Parent(String name)
11 {
12     this.name = name;
13 }
14

```

```

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.

실습문제

웹하드및 스터디룸 9강 연습문제 참조

웹하드 : <http://kedudisk.com/pm/>

아이디: it3000

패스워드:koreait11!

스터디룸 : <http://koreastudyroom.com>

아이디 패스워드 학원 문의

실습문제1

섯다카드 20장을 포함하는 섯다카드 한벌(SutdaDeck)을 정의한 것이다. 섯다카드 20장을 담는 SutdaCard 배열을 초기화 하자.

단, 섯다카드는 1~10까지의 숫자가 적힌 카드가 한 쌍씩 있고, 숫자가 1,3,8인 경우 둘 중의 한장은 광이어야 한다.

즉, SutdaCard의 인스턴스변수 isKwang의 값이 true이다.

```
3 public class SutdaDeck {
4
5     final int CARD_NUM = 20;
6     public SutdaCard[] cards = new SutdaCard[CARD_NUM];
7     public SutdaDeck()
8     {
9
10        //카드배열을 초기화 하시오
11
12    }
13 }
14 |
15
16
```

```
<terminated> main [Java Application] C:\Users\Wzest1\p2\pool\plugins\org.eclipse.justj.op
1K,2,3K,4,5,6,7,8K,9,10,1K,2,3K,4,5,6,7,8K,9,10,
```

```
public class SutdaCard {

    int num;
    boolean isKwang;
    public SutdaCard()
    {
        this(1, true);
    }

    public SutdaCard(int num, boolean isKwang)
    {
        this.num = num;
        this.isKwang = isKwang;
    }

    public String toString()
    {
        return num + ( isKwang ? "K":"" );
    }

}
```

실습문제2

실습문제 1의 SutdaDeck클래스에 다음에 정의된 새로운 메서드를 추가하고 테스트 하시오

메서드명 : shuffle

기능 : 배열 cards에 담긴 카드의 위치를 뒤섞는다. (Math.random 활용)

반환,매개변수 없음

메서드명: pick

기능 : 배열 cards에서 지정된 위치의 SutdaCard를 반환한다.

반환타입 : SutdaCard

매개변수 : int index, 배열의 인덱스

메서드명 pick

기능 : 배열 cards에서 랜덤위치의 SutdaCard를 반환 (Math.random 활용)

반환타입 : SutdaCard

매개변수 : 없음

실습문제 풀이1

SutdaDeck의 생성자에서 Cards 배열에 들어갈 카드객체를 만들어 넣는다.

```
public SutdaDeck()
{
    int cardNum ;
    for(int i =0 ; i<CARD_NUM ; i++)
    {
        int cardNumber =i%10+1;
        boolean isKwang=false;

        if( cardNumber ==1 || cardNumber ==3 || cardNumber ==8 )
            isKwang = true;
        else
            isKwang = false;

        cards[i] = new SutdaCard(cardNumber,isKwang);
    }
}
```

실습문제 풀이1

print 메서드는 Object의 toString을 활용하여 출력한다.

```
public class SutdaCard {  
    int num;  
    boolean isKwang;  
    public SutdaCard()  
    {  
        this(1, true);  
    }  
  
    public SutdaCard(int num, boolean isKwang)  
    {  
        this.num = num;  
        this.isKwang = isKwang;  
    }  
  
    public String toString()  
    {  
        return num + ( isKwang ? "K":"" );  
    }  
}
```

```
for(int i=0; i < deck.cards.length; i++)  
    System.out.print(deck.cards[i]+",");
```

실습문제 풀이2

Math.Random 함수는 0.0~0.9999... 사이의 랜덤한수를 반환한다.
따라서 Math.Random * 20을 하면 0~19 사이의 랜덤한 숫자가 나온다.

shuffle 메서드 구현 : 0번째 부터 끝까지 순차적으로 랜덤한 위치의 카드와 서로 맞바꾸면서 카드의 순서를 뒤섞는다.

```
public void shuffle()
{
    for(int i=0; i<cards.length;i++)
    {
        int target = (int)(Math.random()*cards.length);

        SutdaCard tmp = cards[i];
        cards[i] = cards[target];
        cards[target] = tmp;
    }
}
```

실습문제 풀이2

매개변수가 있는 pick 구현 : 매개변수에 해당하는 위치의 카드를 반환한다. index가 음수이거나 가진 카드의 숫자보다 클 경우 null을 돌려주는 필터링을 한다.

```
SutdaCard pick(int index)
{
    if(index < 0 || index >= CARD_NUM) // 배열의 범위 내에서만 되도록
        return null;

    return cards[index];
}
```

매개변수 없는 pick 구현 : 랜덤한 위치의 카드를 반환한다. 이때 기존에 만들어둔 pick 메서드를 활용하고 있다.

```
SutdaCard pick()
{
    int index = (int)(Math.random()*cards.length);

    return pick(index); //pick 메서드를 활용한다.
}
```




THANK YOU



ParkJuByeong



강사 박주병