

# 9장. 추상 클래스

*abstract class*



# 추상 클래스(Abstract Class)

## 추상 클래스

객체를 직접 생성할 수 있는 클래스를 실체 클래스라고 한다면 이 클래스들의 공통적인 특성을 추출해서 선언한 클래스를 추상 클래스라 한다.

추상클래스와 실체 클래스는 상속 관계를 구성한다.

왜 추상클래스를 사용하는가?

실체 클래스의 필드와 메서드의 이름을 통일할 목적으로 사용한다.

TelePhone 클래스 – owner(소유자), powerOn() - 전원을 켜다

SmartPhone 클래스 – user(소유자), trunOn() – 전원을 켜다

추상클래스 사용

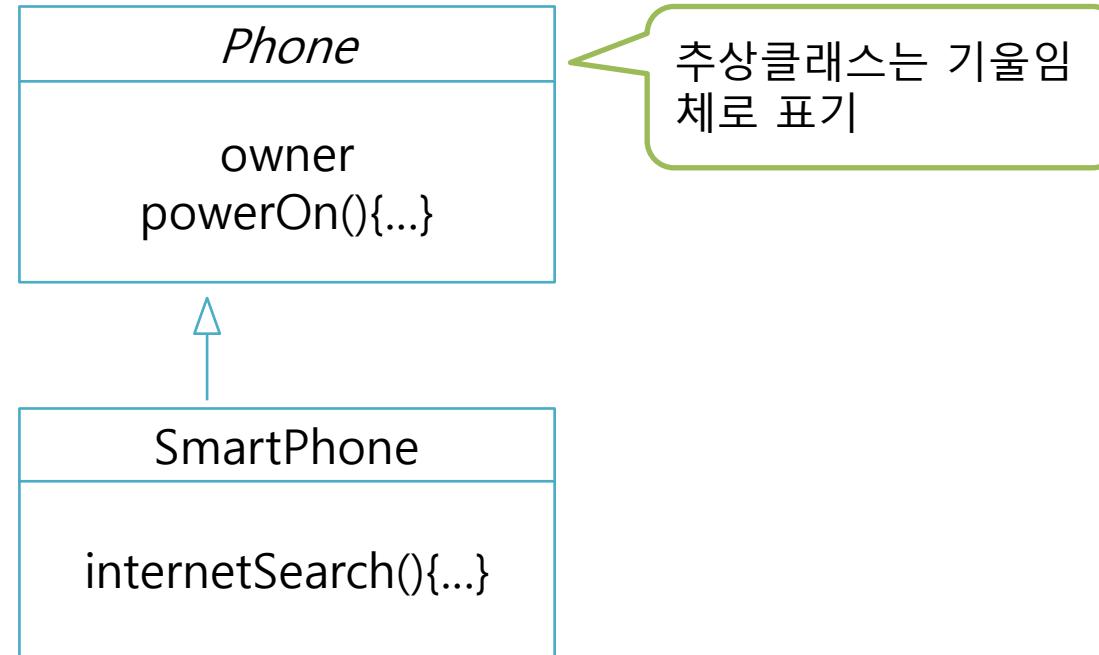
Phone – user, powerOn() -> TelePhone, SmartPhone이 상속 받음

## 추상 클래스의 선언

```
Public abstract class 클래스이름{  
    //필드, 생성자, 메서드  
}
```



# 추상 클래스(Abstract Class)



# 추상 클래스(Abstract Class)

```
package abstractex;

public abstract class Phone {

    public String owner;

    public Phone(String owner) {
        this.owner = owner;
    }

    public void powerOn() {
        System.out.println("폰 전원을 켭니다.");
    }

    public void powerOff() {
        System.out.println("폰 전원을 끕니다.");
    }
}
```

```
public class SmartPhone extends Phone{

    public SmartPhone(String owner) {
        super(owner);
    }

    public void internetSearch() {
        System.out.println("인터넷 검색을 합니다.");
    }
}
```



# 추상 클래스(Abstract Class)

```
public class PhoneTest {

    public static void main(String[] args) {
        //Phone phone = new Phone(); 추상 클래스는 객체 생성 불가
        SmartPhone smartPhone = new SmartPhone("오상식");

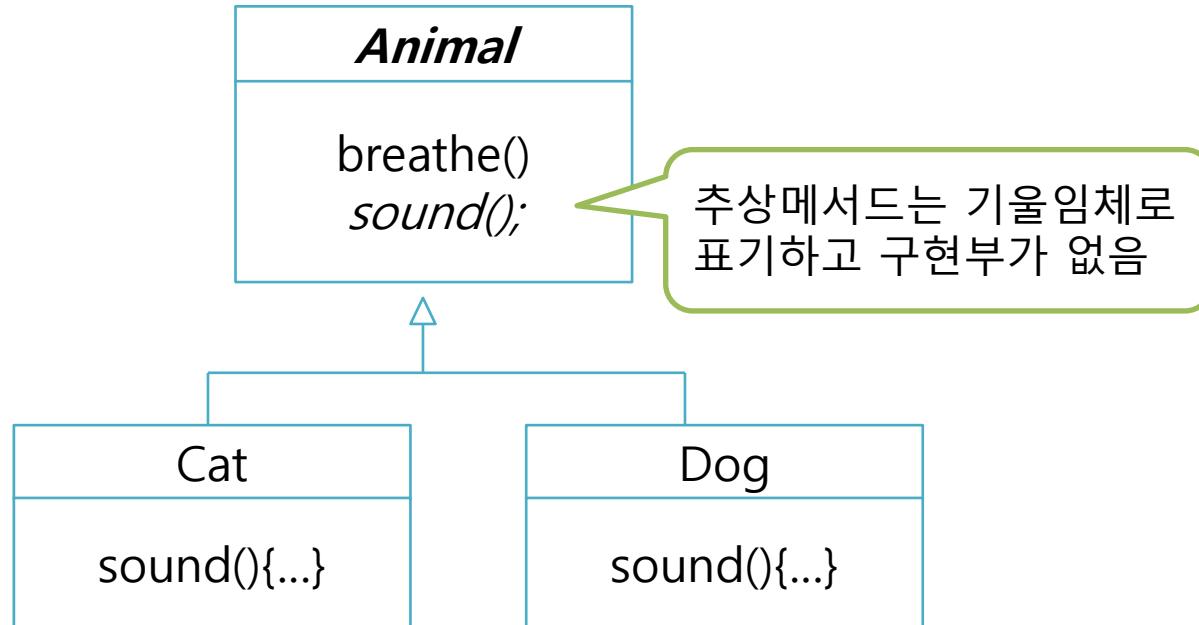
        smartPhone.powerOn();
        smartPhone.internetSearch();
        smartPhone.powerOff();
    }
}
```



# 추상 클래스(Abstract class)

## ■ 추상 메서드

- 추상 메서드도 **abstract** 예약어를 사용한다.
- 메서드를 구현하지 않고 선언만 한다. {} 구현부가 없다.
- 상속받는 실체 클래스는 추상메서드를 필수적으로 구현해야 한다.



# 추상메서드

```
package abstractex.animal;

public abstract class Animal {

    public String kind;

    public void breathe() {
        System.out.println("숨을 쉽니다.");
    }

    public abstract void sound();
}
```



# 추상메서드

- 동물의 소리를 구현한 추상클래스 상속 예.

```
*Cat.java ✘
1 package abstractex;
2
3 public class Cat extends Animal{
4
5 }
6
7 
```

The type Cat must implement the inherited abstract method Animal.cry()  
2 quick fixes available:  
Add unimplemented methods  
Make type 'Cat' abstract

```
public class Dog extends Animal{

    public Dog() {
        this.kind = "포유류";
    }

    @Override
    public void sound() {
        System.out.println("멍멍");
    }
}
```

추상메서드는 반드시 구현해야 함



# 추상메서드

```
public class AnimalMain {  
    public static void main(String[] args) {  
        Dog dog = new Dog();  
        Cat cat = new Cat();  
  
        dog.sound();  
        cat.sound();  
        System.out.println("=====");  
  
        //자동 타입 변환  
        Animal animal = null;  
        animal = new Dog();  
        animal.sound();  
  
        animal = new Cat();  
        animal.sound();  
        System.out.println("=====");  
  
        //메소드의 다형성  
        animalSound(new Dog());  
        animalSound(new Cat());  
    }  
  
    private static void animalSound(Animal animal) {  
        animal.sound();  
    }  
}
```

명명  
야옹  
=====

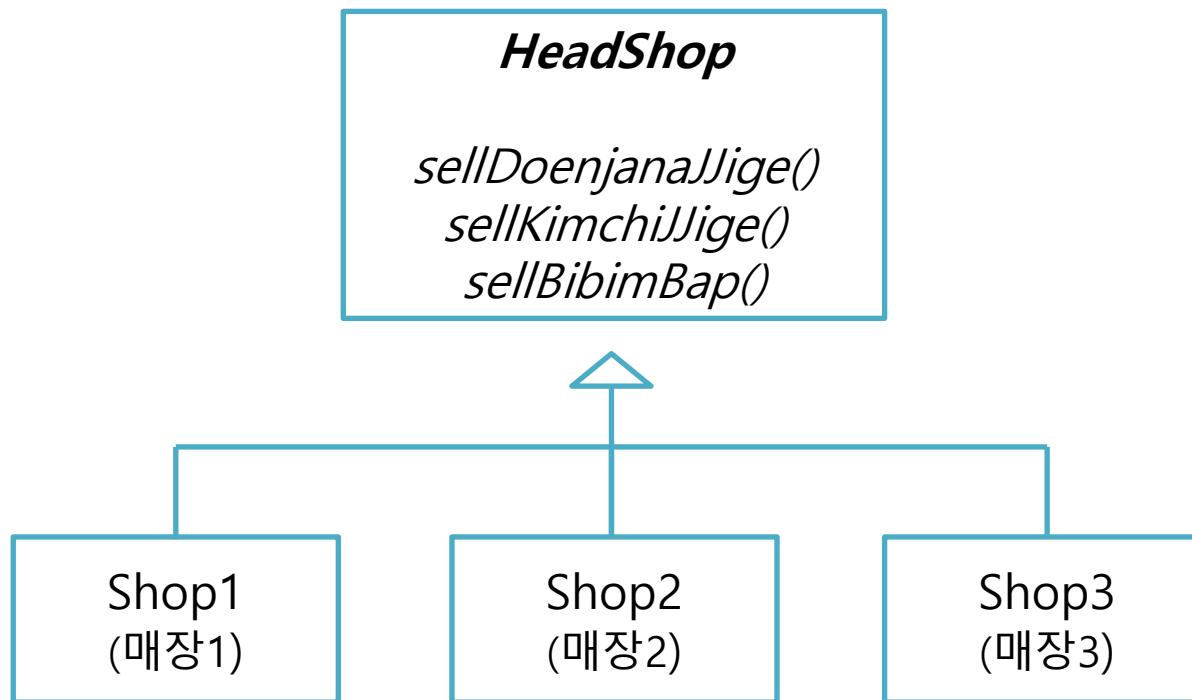
명명  
야옹  
=====

명명  
야옹



# 추상 클래스(Abstract Class)

## ▪ 한식 프랜차이즈 추상클래스 상속 예



# 추상클래스 실습

## ■ 한식 프랜차이즈 추상클래스 상속 예

```
public abstract class HeadShop {  
  
    public abstract void sellDoenjangJJige();  
  
    public abstract void sellKimchiJJige();  
  
    public abstract void sellBibimBap();  
}
```

```
public class Shop1 extends HeadShop{  
  
    public Shop1() {  
        System.out.println("대학가 매장입니다.");  
    }  
  
    @Override  
    public void sellDoenjangJJige() {  
        System.out.println("된장찌개 : 5,000원 ");  
    }  
  
    @Override  
    public void sellKimchiJJige() {  
        System.out.println("김치찌개 : 5,500원 ");  
    }  
  
    @Override  
    public void sellBibimBap() {  
        System.out.println("비빔밥 : 6,000원 ");  
    }  
}
```



# 추상클래스 실습

## ■ 한식 프랜차이즈 추상클래스 상속 예

```
public class Shop2 extends HeadShop{

    public Shop2() {
        System.out.println("역세권 매장입니다.");
    }

    @Override
    public void sellDoenjangJJige() {
        System.out.println("된장찌개 : 6,000원");
    }

    @Override
    public void sellKimchiJJige() {
        System.out.println("김치찌개 : 6,500원");
    }

    @Override
    public void sellBibimBap() {
        System.out.println("비빔밥 : 7,000원");
    }
}
```



# 추상클래스 실습

## ■ 한식 프랜차이즈 추상클래스 상속 예

```
HeadShop shop1 = new Shop1();
shop1.sellDoenjangJJige();
shop1.sellKimchiJJige();
shop1.sellBibimBap();
System.out.println("=====");  
  
HeadShop shop2 = new Shop2();
shop2.sellDoenjangJJige();
shop2.sellKimchiJJige();
shop2.sellBibimBap();
System.out.println("=====");
```

대학가 매장입니다.

된장찌게 : 5,000원

김치찌게 : 5,500원

비빔밥 : 6,000원

=====

역세권 매장입니다.

된장찌게 : 6,000원

비빔밥 : 6,500원

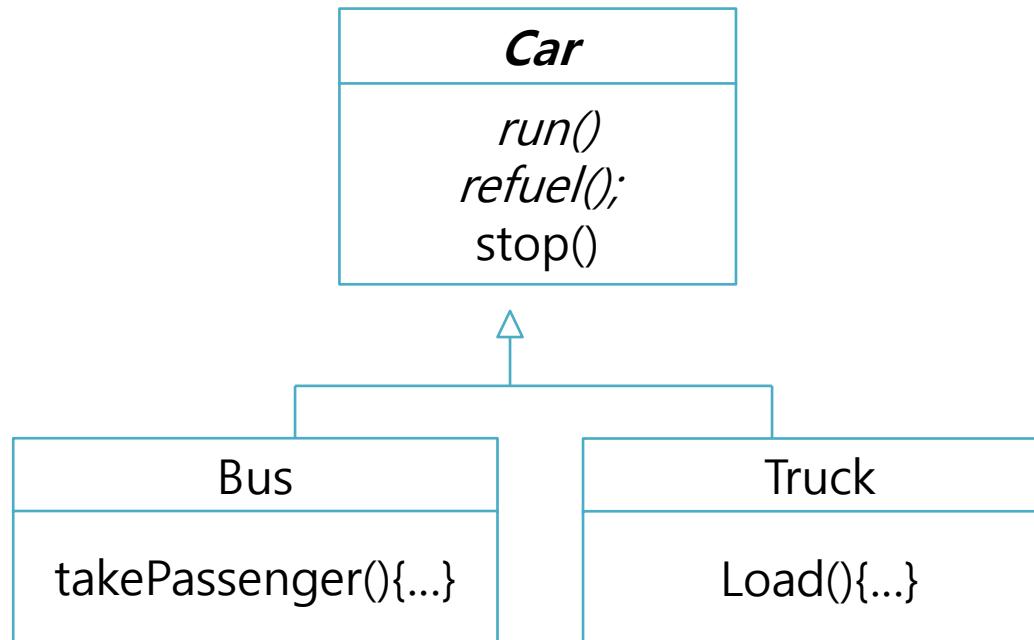
김치찌게 : 7,000원

=====



# 추상 클래스(Abstract Class)

- 자동차를 구현한 추상 클래스 상속 예.



# 추상클래스 실습

```
public abstract class Car {  
    public abstract void run();  
  
    public abstract void refuel();  
  
    public void stop() {  
        System.out.println("차가 멈춥니다")  
    }  
}
```

```
public class Bus extends Car{  
  
    public void takePassenger() {  
        System.out.println("승객을 버스에 태웁니다.");  
    }  
  
    @Override  
    public void run() {  
        System.out.println("버스가 달립니다.");  
    }  
  
    @Override  
    public void refuel() {  
        System.out.println("천연 가스를 충전합니다.");  
    }  
}
```



# 추상클래스 실습

```
public class Truck extends Car{  
  
    @Override  
    public void run() {  
        System.out.println("트럭이 달립니다.");  
    }  
  
    @Override  
    public void refuel() {  
        System.out.println("휘발유를 주유합니다.");  
    }  
  
    public void load() {  
        System.out.println("짐을 싣습니다.");  
    }  
}
```



# 추상클래스 실습

## ■ 자동차를 구현한 추상 클래스 상속 예

```
Bus bus = new Bus();
Truck truck = new Truck();

bus.run();
truck.run();

bus.refuel();
truck.refuel();

bus.takePassenger();
truck.load();

bus.stop();
truck.stop();
```

버스가 달립니다.  
트럭이 달립니다.  
천연 가스를 충전합니다.  
휘발유를 주유합니다.  
승객을 버스에 태웁니다.  
짐을 싣습니다.  
차가 멈춥니다.  
차가 멈춥니다.



# final 예약어

## ▪ 상수를 의미하는 final 변수

- 해당 선언이 최종 상태이고, 결코 수정될 수 없음을 뜻한다.

```
package finalex;

public class Constant {
    int num = 10;
    final int NUM = 100;

    public static void main(String[] args) {
        Constant cons = new Constant();
        cons.num = 20;
        //cons.NUM = 1000; //final 상수이므로 변경할 수 없음

        System.out.println(cons.num);
        System.out.println(cons.NUM);
    }
}
```



# final 상수

- 여러 파일에서 공유하는 상수

```
public class Define {  
    public static final int MIN = 1;  
    public static final int MAX = 99999;  
    public static final int ENG = 1001;  
    public static final int MATH = 2001;  
    public static final double PI = 3.14;  
    public static final String GOOD_MORNING = "Good Morning!";  
}
```



# final 상수

## ■ 여러 파일에서 공유하는 상수

```
public class UsingDefine {  
  
    public static void main(String[] args) {  
        System.out.println(Define.GOOD_MORNING);  
        System.out.println("최솟값은 " + Define.MIN + "입니다.");  
        System.out.println("최대값은 " + Define.MAX + "입니다.");  
        System.out.println("수학 과목 코드값은 " + Define.MATH + "입니다.");  
        System.out.println("영어 과목 코드값은 " + Define.ENG + "입니다.");  
    }  
}
```

Good Morning!  
최솟값은 1입니다.  
최대값은 99999입니다.  
수학 과목 코드값은 2001입니다.  
영어 과목 코드값은 1001입니다.



# final 클래스

- 보안과 관련되어 있거나 기반클래스가 변하면 안 되는 경우

- ✓ String이나 Integer 클래스 등.

The screenshot shows the Eclipse IDE interface with a Java code editor. The code defines a class `MyString` that extends the `String` class. A tooltip message is displayed: "The type MyString cannot subclass the final class String".

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {

    /**
     * The value is used for character storage.
     *
     * @implNote This field is trusted by the VM, and is a subject to
     * constant folding
     * field after const
     */

    private final char[] value;
}
```

```
1 package finalex;
2
3 public class MyString extends String{
4
5 }
6
```

The tooltip message is: "The type MyString cannot subclass the final class String  
Press 'F2' for focus".

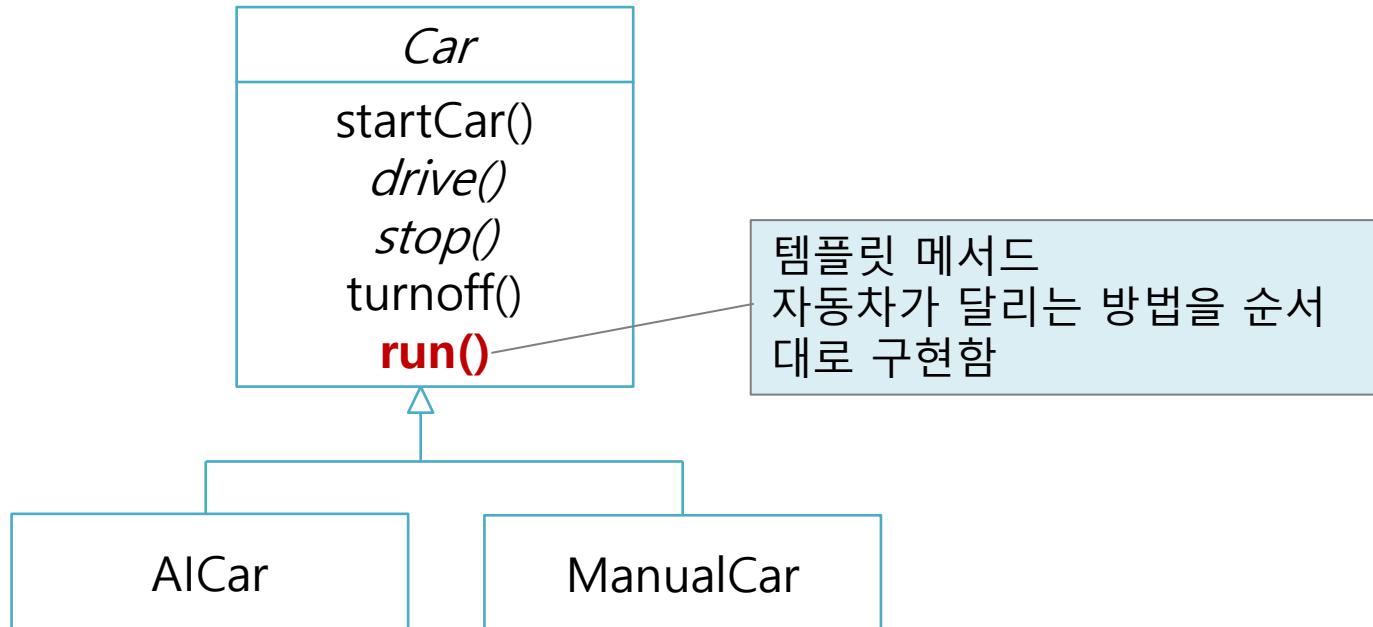
String은 final 클래스이므로  
상속받을 수 없다.



# 템플릿 메서드

## ■ 템플릿 메서드란?

- 템플릿 메서드 : 추상 메서드나 구현된 메서드를 활용하여 전체 기능의 흐름(시나리오)를 정의하는 메서드.
- **final**로 선언하면 하위 클래스에서 재정의 할 수 없음



# 템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public abstract class Car {  
    public abstract void drive();  
    public abstract void stop();  
  
    public void startCar() {  
        System.out.println("시동을 켭니다.");  
    }  
  
    public void turnOff() {  
        System.out.println("시동을 끕니다.");  
    }  
  
    public final void run() {  
        startCar();  
        drive();  
        stop();  
        turnOff();  
    }  
}
```

final로 선언  
상속받은 하위 클래스가 메서드  
를 재정의 할 수 없다.



# 템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class AICar extends Car{  
  
    @Override  
    public void drive() {  
        System.out.println("자율 주행합니다.");  
        System.out.println("자동차가 스스로 방향을 전환합니다.");  
    }  
  
    @Override  
    public void stop() {  
        System.out.println("자동차가 스스로 멈춥니다.");  
    }  
}
```



# 템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class ManualCar extends Car{  
  
    @Override  
    public void drive() {  
        System.out.println("사람이 운전합니다.");  
        System.out.println("사람이 핸들을 조작합니다.");  
    }  
  
    @Override  
    public void stop() {  
        System.out.println("브레이크로 정지합니다.");  
    }  
}
```



# 템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class CarTest {  
  
    public static void main(String[] args) {  
        System.out.println("===== 자율 주행하는 자동차 =====");  
        Car myCar = new AICar();  
        myCar.run();  
  
        System.out.println("===== 사람이 운전하는 자동차 =====");  
        Car hisCar = new ManualCar();  
        hisCar.run();  
    }  
}
```

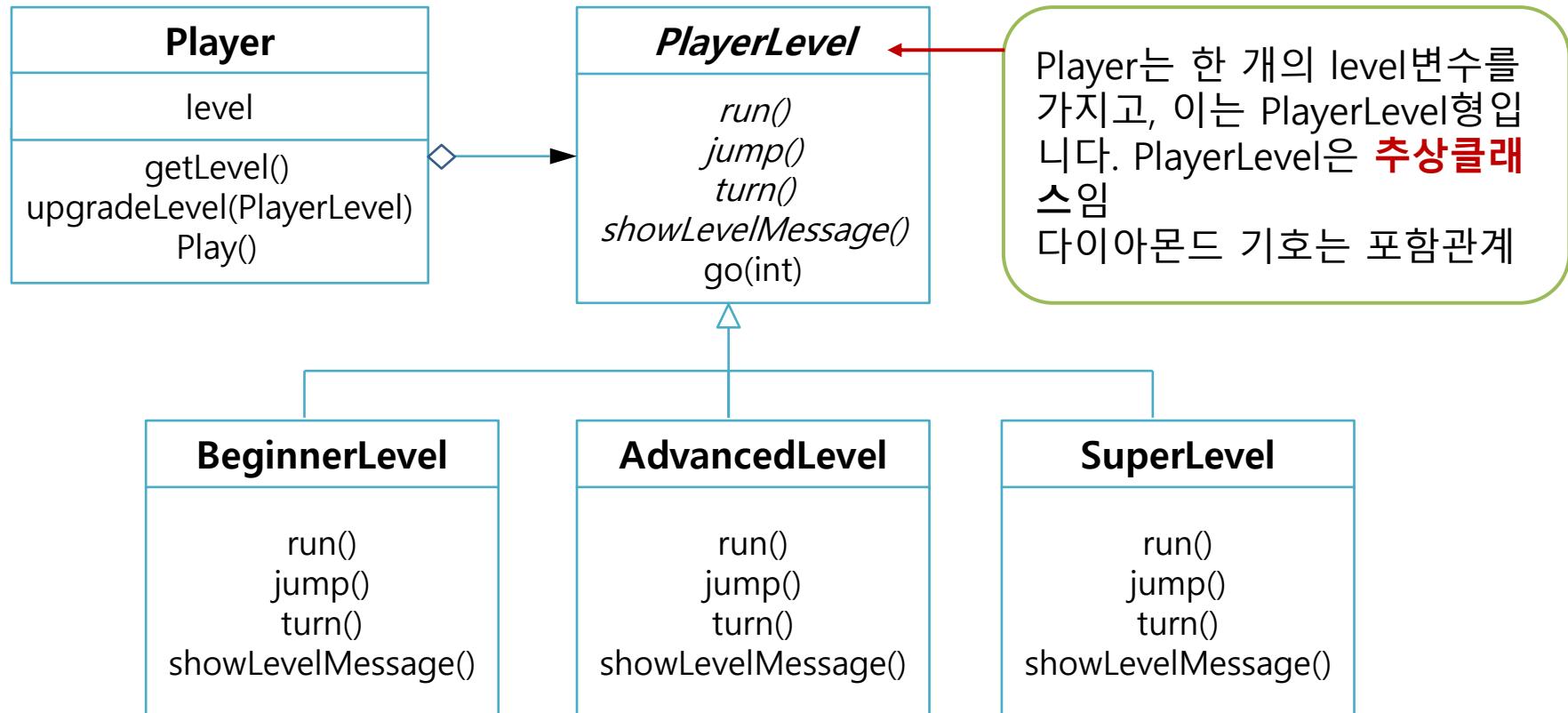
===== 자율 주행하는 자동차 =====  
시동을 겁니다.  
자율 주행합니다.  
스스로 멈춥니다.  
시동을 끕니다.  
===== 사람이 운전하는 자동차 =====  
시동을 겁니다.  
사람이 운전합니다.  
브레이크로 정지합니다.  
시동을 끕니다.



# Game Level App

## 예제 시나리오

Player가 있고 이 플레이어가 게임을 합니다. 게임에서 Player가 가지는 레벨에 따라 세가지 기능이 있는데 run(), jump(), turn()입니다.



# Game Level App

## PlayerLevel 클래스

각 레벨에서 수행할 공통 기능은 PlayLevel 추상 클래스에서 선언한다.

```
public abstract class PlayerLevel {  
    public abstract void run();  
    public abstract void jump();  
    public abstract void turn();  
    public abstract void showLevelMessage();  
  
    final public void go(int count) {  
        run();  
        for(int i=0; i<count; i++) {  
            jump();  
        }  
        turn();  
    }  
}
```

한번 run하고, count만큼  
jump하고, 한번 turn한다.



# Game Level App

## Beginner 클래스

초보자 레벨에서는 천천히 달리 수만 있다. 점프나 턴을 할 수 없다.

```
public class Beginner extends PlayerLevel {

    @Override
    public void run() {
        System.out.println("천천히 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("jump할 줄 모르지롱.");
    }

    @Override
    public void turn() {
        System.out.println("Turn할 줄 모르지롱.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****초보자 레벨입니다.*****");
    }
}
```



# Game Level App

## Advanced 클래스

중급자 레벨에서는 빠르게 달릴 수 있고, 높이 점프할 수 있다. 턴을 할 수 없다.

```
public class AdvancedLevel extends PlayerLevel{

    @Override
    public void run() {
        System.out.println("빨리 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("높이 jump합니다.");
    }

    @Override
    public void turn() {
        System.out.println("Turn 할 줄 모르지롱.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****중급자 레벨입니다.*****");
    }
}
```



# Game Level App

## SuperLevel 클래스

고급자 레벨에서는 매우 빠르게 달릴 수 있고, 매우 높이 점프할 수 있다. 턴하는 기술도 사용할 수 있다.

```
public class SuperLevel extends PlayerLevel{

    @Override
    public void run() {
        System.out.println("매우 빨리 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("매우 높이 jump합니다.");
    }

    @Override
    public void turn() {
        System.out.println("한 바퀴 돋니다.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****고급자 레벨입니다.*****");
    }
}
```



# Game Level App

## Player 클래스

```
public class Player {
    //PlayerLevel 클래스 참조
    private PlayerLevel level;

    public Player() {
        level = new Beginner();
        level.showLevelMessage();
    }

    public void upgradeLevel(PlayerLevel level) { //매개변수의 다형성
        this.level = level;
        level.showLevelMessage();
    }

    public void play(int count) { //템플릿 메서드 호출
        level.go(count);
    }
}
```



# Game Level App

## 테스트 프로그램 실행

```
Player player = new Player();
//처음 생성시 BeginnerLevel
player.play(1);

//중급자 레벨
AdvancedLevel aLevel = new AdvancedLevel();
player.upgradeLevel(aLevel);
player.play(2);

//고급자 레벨
SuperLevel sLevel = new SuperLevel();
player.upgradeLevel(sLevel);
player.play(3);
```

\*\*\*\*\*초보자 레벨입니다.\*\*\*\*\*  
천천히 달립니다.  
jump할 줄 모르지롱.  
Turn할 줄 모르지롱.  
\*\*\*\*\*중급자 레벨입니다.\*\*\*\*\*  
빨리 달립니다.  
높이 jump합니다.  
높이 jump합니다.  
Turn 할 줄 모르지롱.  
\*\*\*\*\*고급자 레벨입니다.\*\*\*\*\*  
매우 빨리 달립니다.  
매우 높이 jump합니다.  
매우 높이 jump합니다.  
매우 높이 jump합니다.  
한 바퀴 돋니다.

