Abstract 클래스

경복대학교
스마트IT소프트웨어과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Abstract Class

- Abstract Class는 공통적인 Behavior을 여러 Sub Class에서 재사용할 수 있도록 만들어진 Class
- abstract Keyword를 사용하여 선언하며, Object를 직접 생성할 수 없음
- 공통된 Behavior(기능) (Method, 변수)을 Child Class에서 재사용할 수 있음
- 특정 Behavior은 Child Class에서 반드시 구현해야 하도록 강제할 수 있음 (Abstract Method)

- 언제 사용하면 좋을까?
  - 여러 개의 Class에서 공통적인 Attribute와 Behavior를 제공할 때 일부 Method는 구현하고, 일부는 Child Class가 직접 구현하도록 강제하고 싶을 때

Abstract Class

- Abstract Class를 사용하는 이유
  - Code 재 사용성 증가
  - 강제 구현 (일관성 유지)
  - 유지 보수성 향상

Animal Class Modeling

- Animal(Dog, Cat, Lion)은 공통적으로 "먹는다(eat())"와 "잠을 잔다(sleep())" Behavior이 있음
- 각 동물마다 소리를 내는 행동(makeSound())은 다름
- 따라서, 공통의 Behavior는 Parent Class에서 정의하여 상속 받도록 하고, 동물 별로 다른 Behavior는 Child Class에서 반드시 구현하도록 함

Animal Class Modeling

- Animal Class

```
abstract class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public void eat() {
        System.out.println(name + "이 먹이를 먹습니다.");
    }
    public void sleep() {
        System.out.println(name + "이 잠을 잡니다.");
    }
    public abstract void makeSound();
}
```

Animal Class Modeling

- Dog Class

```
public class Dog extends Animal{

    public Dog(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(super.getName() + "이 멍멍 웁니다");
    }
}
```

Animal Class Modeling

- Cat Class

```
public class Cat extends Animal{
    public Cat(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(super.getName() + "이 야옹야옹 웁니다");
    }
}
```

Animal Class Modeling

- Lion Class

```
public class Lion extends Animal{

    public Lion(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(super.getName() + "이 어흥하고 웁니다");
    }
}
```

Animal Class Modeling

- Main Class

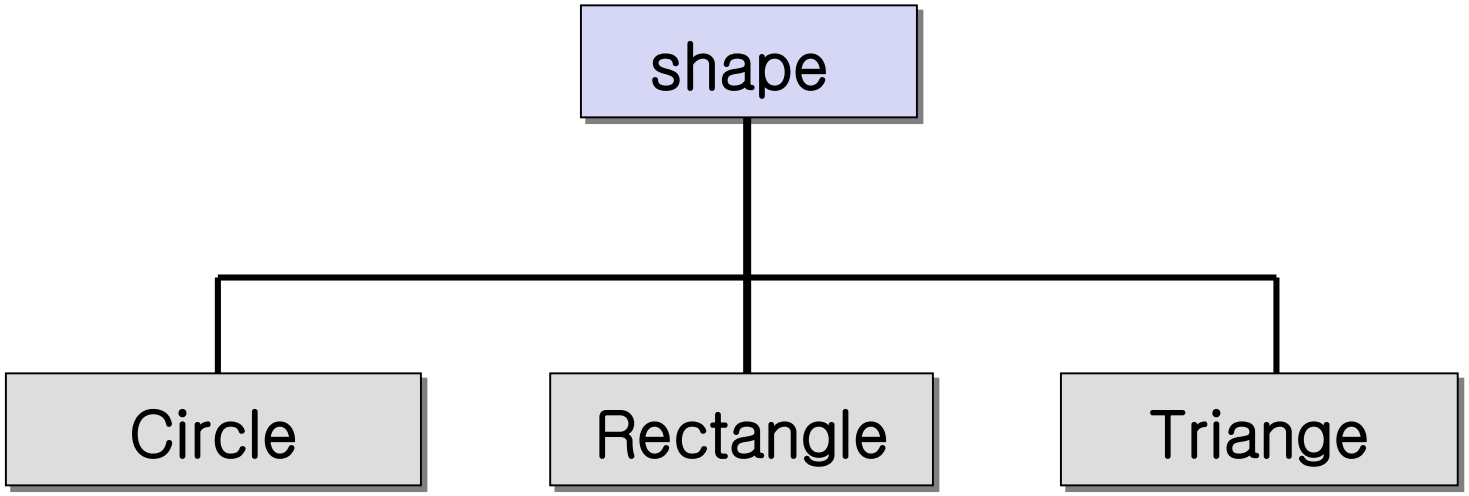
```
public static void main(String[] args) {
    // Animal 객체를 직접 생성할 수 없음! (컴파일 오류)
     Animal animal = new Animal();

    // 각각의 동물 객체 생성 (UpCasting 활용 가능)
    Animal dog = new Dog("우리 코코");
    Animal cat = new Cat("야맹이");
    Animal lion = new Lion("아기 사자");

    dog.eat();
    cat.sleep();
    lion.sleep();

     //개별적인 동물 소리 테스트 (다형성)
    dog.makeSound();
    cat.makeSound();
    lion.makeSound();
}
```

[도입 예제] 도형 그리기

- “모든 도형은 색을 가지고 있고, 그릴 수 있어야 한다!" 라는 규칙을 적용 해보자
- 하지만 정사각형(Rectangle), 원(Circle), 삼각형(Triangle) 같은 도형들은 각각 다르게 그려지겠죠?

[도입 예제] 도형 그리기

- Class 구조는 Circle, Rectangle, Triangle Class가 Shape Class로부터 상속된 형태이며 3개의 Sub Class에 각각의 모형을 화면에 그리는 draw() Method가 필요 함



[도입 예제] 도형 그리기

- Shape Class

```
abstract class Shape {
    String color;       // 모든 도형이 공통으로 가지는 속성

    public Shape(String color) {      // 생성자
        this.color = color;
    }

    abstract void draw();  // 추상 Method(구체적인 구현 없이 선언)

    void displayColor() {    // concrete Method
       System.out.println("이 도형의 색상은 " + color + "입니다.");
    }
}
```

[도입 예제] 도형 그리기

- Rectangle Class

```
class Rectangle extends Shape {
   int width;
   int height;

   public Square(String color, int width, int height) {
        super(color);
        this.width = width;
        this.height = height;
   }

   @Override
   void draw() {  // 추상 메서드 구현 (사각형을 그리는 방법 정의)
         System.out.println( " 가로 길이가 " + width + “이고, 세로의
                              길이가 " + height + “이고, 색상이 “ color +
                                                  " 인 사각형을 그립니다.");
   }
}
```

[도입 예제] 도형 그리기

- Circle Class

```
class Circle extends Shape {
    int radius;

    public Circle(String color, int radius) {
        super(color);
        this.radius = radius;
    }

    @Override
    void draw() {      // 추상 메서드 구현 (원을 그리는 방법 정의)
        System.out.println("반지름 " + radius + "인 " + color
                                                + " 원을 그립니다.");
    }
}
```

[도입 예제] 도형 그리기

- Triangle Class

```
class Triangle extends Shape {
    int base;
    int height;

    public Triangle(String color, int base, int height) {
        super(color);
        this.base = base;
        this.height = height;
    }

    @Override
    void draw() {      // 추상 메서드 구현 (원을 그리는 방법 정의)
        System.out.println("밑변 길이가 " + width + "이고, 높이의
                        길이가 " + height + "이고, 색상이 " color +
                                        " 인 삼각형을 그립니다.");
    }
}
```

[도입 예제] 도형 그리기

- Main Class

```
public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle("빨강", 5);
        Rectangle rectangle = new Square("파랑", 4, 5);

        circle.draw();
        rectangle.draw();

        circle.displayColor();
        square.displayColor();
    }
}
```

[도입 예제] 도형 그리기

- 상속의 개념을 이용하여 3개의 Sub Class에서 공통적으로 필요한 Method를 Super Class인 Shape에 정의
- 즉 모형을 화면에 그리는 Method인 draw()를 Shape Class에 정의하면, 상속된 Sub Class에서는 다시 정의하지 않고 사용할 수 있음
- 하지만 각각의 상속 받은 Sub Class에서 실제 그리는 방법은 각각의 Sub Class마다 다른 방법을 사용하여야 함
- 실제 화면에 삼각형이나 사각형을 그리는 방법이 다르기 때문임
- 즉 Sub Class에서는 Super Class에서 Abstract Method로 정의된 Method를 반드시 Overriding하여 사용함

[도입 예제] 도형 그리기

- Sub Class에서 체계적인 Class를 설계하도록 하기 위해서 JAVA에서는 Abstract Class를 제공
- Abstract Class에 동일한 Interface를 요구하는 Abstract Method를 정의하면 이 Abstract Class를 Super Class를 가지는 Sub Class에서 반드시 동일한 이름의 Method(Abstract Method)를 정의하도록 강제성을 부여함
- 그리고, 구체적인 기능들을 Sub Class에서 구현하도록 함으로서 Polymopism을 제공

[도입 예제] 도형 그리기

- 왜 사용할까?
  - Code를 깔끔하고 일관성 있게 유지
    - 모든 도형(Shape)이 그릴 수 있어야 한다는 규칙을 강제할 수 있음
  - Code 재 사용성 증가
    - 공통된 기능을 추상 Class에 넣어서 중복을 줄일 수 있음
  - 유지 보수 편리
    - 새 도형을 추가할 때(Triangle), 기본적인 구조를 그대로 활용할 수 있음

Abstract Class 예제 0

- 다음의 추상 클래스 Calculator를 상속받는 GoodCalc 클래스를 작성하고, 테스트하여라

```
abstract class Calculator {
	public abstract int add(int a, int b);
	public abstract int subtract(int a, int b);
	public abstract double average(int[] a);
}
```

Abstract Class 예제 0

```
public class GoodCalc extends Calculator {
	public int add(int a, int b) {
		return a + b;
	}

	public int subtract(int a, int b) {
		return a - b;
	}

	public double average(int[] a) {
		double sum = 0;
		for (int i = 0; i < a.length; i++)
			sum += a[i];
		return sum / a.length;
	}
}
```

Abstract Class 예제 0

```
public static void main(String [] args) {
        Calculator c = new GoodCalc();

        System.out.println(c.add(2, 3));
        System.out.println(c.subtract(2, 3));
        System.out.println(c.average(new int [] {2, 3, 4}));
}
```

```
5
-1
3.0
```

Abstract Class 예제 1

- 추상 메소드 사용 전

```
public class Student {
    public String name;     // 이름
    public int grade;       // 학년

    public Student() {              // 생성자
    }
    public Student(String name, int grade) {    // 생성자
        this.name = name;
        this.grade = grade;
    }

    public String getSchool(String school) {     // 메소드
        return school +"학교";
    }

    public String toString() {
        return "이름 : " + name + " 학년 : " + grade;
    }
}
```

Abstract Class 예제 1

- 추상 메소드 사용 전

```
public class HighSchool extends Student {
    public HighSchool(String name, int grade) {
        super(name, grade);
    }
    public String getSchool(String school) {
        return school + "고등학교";
    }
}
```

```
public class University extends Student {
    public University(String name, int grade) {
        super(name, grade);
    }
    public String getUniversity(String school){
        return school +"대학교";
    }
}
```

Abstract Class 예제 1

- 추상 메소드 사용 전

```
public class Main {
    public static void main(String[] args) {
        Student kim = new Student();
        Student lee = new University("이대한", 3);
        HighSchool park  = new HighSchool("나경복", 2);

        System.out.println(lee.getUniversity("경복") );
        System.out.println(park.getSchool("대한") );
    }
}
```

✓ 문제점
  ✓ 상위 클래스에서 구현한 메소드를 무시하고 하위 클래스에서 추가된 메소드로 구현
  ✓ 강제성과 통일성이 없음

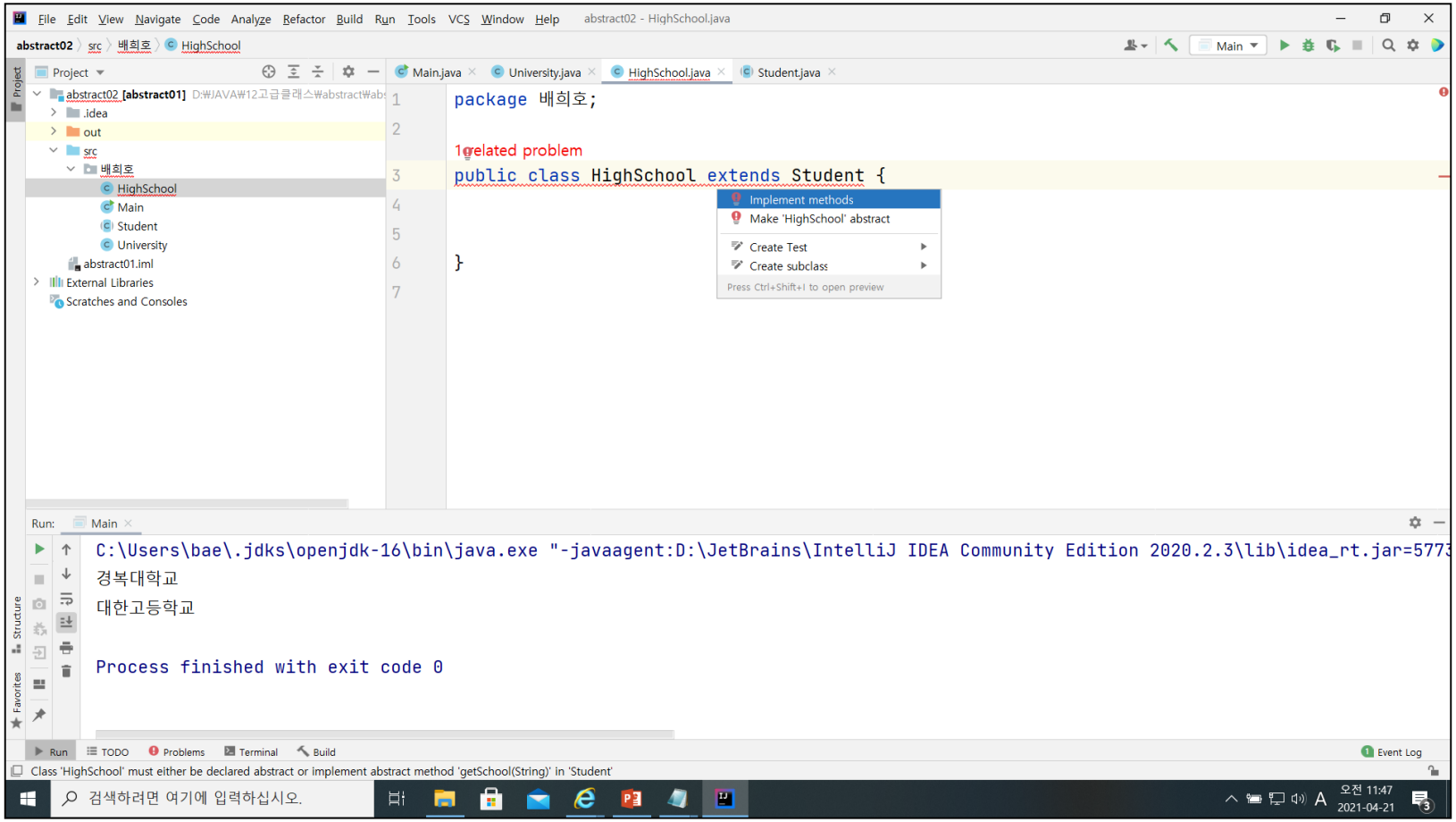
Abstract Class 예제 1

- 추상 메소드 사용 후

```
public abstract class Student {              // 추상 클래스
    public String name;     //이름
    public int grade;       //학년

    public Student() {
    }

    public Student(String name, int grade) {
        this.name = name;
        this.grade = grade;
    }

    public String toString() {
        return "이름 : " + name + " 학년 : " + grade;
    }

    public abstract String getSchool(String school);    // 추상 메소드
}
```

Abstract Class 예제 1

- 추상 메소드 사용 후



Abstract Class 예제 1

- 추상 메소드 사용 후

```
public class HighSchool extends Student {
    public HighSchool(String name, int grade) {
        super(name, grade);
    }
    public String getSchool(String school) {
        return school + "고등학교";
    }
}
```

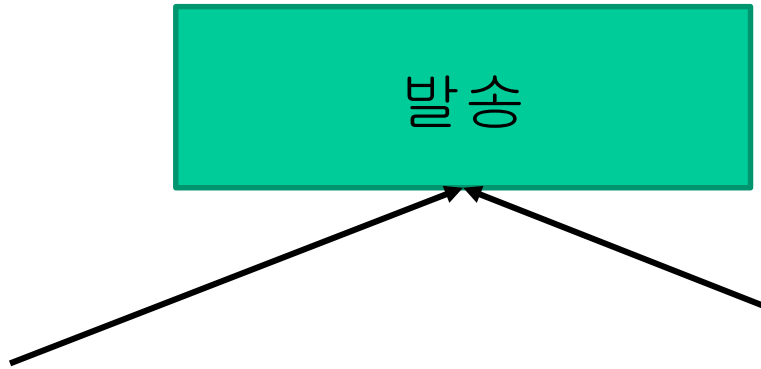
```
public class University extends Student {
    public University(String name, int grade) {
        super(name, grade);
    }
    public String getSchool(String school){
        return school +"대학교";
    }
}
```

- ✓ 상위 클래스에서 선언한 추상 메소드를 하위 클래스에서 구현
- ✓ 강제성과 통일성을 줌
- ✓ 즉, 재사용이 가능

Abstract Class 예제 2

- 고객센터에서 생일 축하 10% 할인 쿠폰을 문자 메시지나 E-mail로 발송하는 발송 시스템을 만들어보자

Abstract Class 예제 2



제목:
보내는 사람:
받는 사람:
회신 전화번호 :
내용:

제목:
보내는 사람:　　보내는 사람 주소
받는 사람:
내용:

Abstract Class 예제 3

- MessageSender 클래스

```
abstract class MessageSender {
    String title;
    String senderName;

    public MessageSender(String title, String senderName) {
        this.title = title;
        this.senderName = senderName;
    }

    abstract void sendMessage(String recipient);
}
```

클래스 자체도 추상 클래스로 선언

추상 메소드 선언

Abstract Class 예제 3

- EmailSender 클래스

```
public class EmailSender extends MessageSender {
    String senderAddr;
    String emailBody;

    public EmailSender(String title, String senderName,
                                        String senderAddr, String emailBody) {
        super(title, senderName);
        this.senderAddr = senderAddr;
        this.emailBody = emailBody;
    }

    public void sendMessage(String recipient) {
        System.out.println("------------------------------");
        System.out.println("제목: " + title);
        System.out.println("보내는 사람: " + senderName + " " + senderAddr);
        System.out.println("받는 사람: " + recipient);
        System.out.println("내용: " + emailBody);
    }
}
```

슈퍼 클래스의 메소드를 오버라이드하는 메소드

Abstract Class 예제 3

- SMSSender 클래스

```
public class SMSSender extends MessageSender {
    String returnPhoneNo;
    String message;

    public SMSSender(String title, String senderName,
                                    String returnPhoneNo, String message) {
        super(title, senderName);
        this.returnPhoneNo = returnPhoneNo;
        this.message = message;
    }
    public void sendMessage(String recipient) {
        System.out.println("------------------------------");
        System.out.println("제목: " + title);
        System.out.println("보내는 사람: " + senderName);
        System.out.println("전화번호: " + recipient);
        System.out.println("회신 전화번호: " + returnPhoneNo);
        System.out.println("메시지 내용: " + message);
    }
}
```

Abstract Class 예제 3

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        EmailSender email = new EmailSender("생일을 축하합니다", "고객센터",
                "admin@kbushop.co.kr", "10% 할인쿠폰이 발행되었습니다.");
        SMSSender sms = new SMSSender("생일을 축하합니다", "고객센터",
                "02-845-5673", "10% 할인쿠폰이 발행되었습니다.");

        email.sendMessage("test@gmail.com");
        email.sendMessage("happy@gmail.com");
        sms.sendMessage("010-345-1234");
    }
}
```

Abstract Class 예제 4

- 집을 건축할 때 나무집과 유리집으로 구분할 수 있다
- 집을 건축하는 방법은 동일한데 벽체나 기둥을 어떤 소재로 마감공사하는 가에 따라 구분된다.

Abstract Class 예제 4

House

Glass House

Wood House

Abstract Class 예제 4

- House 클래스

```
abstract public class House {
    public final void buildHouse(String house) {
        buildFoundation();
        buildPillars();
        buildWalls();
        buildWindows();
        System.out.println(house + "이 완성되었습니다");
    }
    private void buildWindows() {
        System.out.println("창문은 유리로 공사를 합니다");
    }

    public abstract void buildWalls();
    public abstract void buildPillars();
    private void buildFoundation() {
        System.out.println("건물의 기초공사를 합니다");
    }
}
```

Abstract Class 예제 4

- WoodHouse 클래스

```
public class WoodHouse extends House {
    @Override
    public void buildWalls() {
        System.out.println("벽을 나무로 마감공사 합니다");
    }

    @Override
    public void buildPillars() {
        System.out.println("건물의 기둥을 나무로 마감공사 합니다");
    }
}
```

Abstract class 예제 4

- GlassHouse 클래스

```
public class GlassHouse extends House {
    @Override
    public void buildWalls() {
        System.out.println("건물의 벽을 Glass로 마감공사 합니다");
    }

    @Override
    public void buildPillars() {
        System.out.println("건물의 기둥을 Glass로 마감공사 합니다");
    }
}
```

Abstract Class 예제 4

- Main 클래스

```
public class Main {

    public static void main(String[] args) {
        House woodenHouse = new WoodHouse();
        woodenHouse.buildHouse("나무집");

        House glassHouse = new GlassHouse();
        glassHouse.buildHouse("유리집");
    }
}
```

Abstract Class 예제 5

- Mobile Phone은 초기 모델인 Feature Phone과 Smart Phone으로 구분할 수 있음
  - Feature Phone
    - 스마트폰이 아닌 기존 일반 휴대폰
    - 통화, 부과하는 이동전화 통신망을 이용해 Internet 메시지 보내기가 주요 기능
    - Data 통신료를 사용 가능
  - Smart phone
    - WIFI 내장
    - Application 설치 가능
    - 어플리케이션을 통한 폰의 내장된 기능을 다양하게 변화시켜 사용 가능
    - 인터넷 사용이 자유로움
    - 운영체제 : iOS, 안드로이드……

Abstract Class 예제 5

Abstract Class 예제 5

- MobilePhone 클래스

```
abstract public class MobilePhone {
    String name;
    int fee;

    public abstract void WIFIconnect();
    public abstract void useInternet();



    public void call() {
        System.out.println(name+"이 전화를 겁니다.
                                        통화요금으로 100원이 부가됩니다.");
        fee += 100;
    }
    public void message() {
        System.out.println(name+"이 메시지를 전송합니다.
                                        문자요금으로 50원이 부가됩니다.");
        fee += 50;
    }
```

Abstract Class 예제 5

- MobilePhone 클래스

```
    public void charge() {
        System.out.println("총 요금은 "+ fee +"원 입니다");
    }

final void usePhone() {
        if(this instanceof Smartphone) {
            name = "Smart Phone";
        } else if(this instanceof Featurephone) {
            name = "Feature Phone";
        }
        call();
        message();
        useInternet();
        WIFIconnect();
        useInternet();
        charge();
    }
}
```

Abstract Class 예제 5

- Featurehone 클래스

```
public class Featurehone extends MobilePhone{
    String name;

    public Featurephone() {
        this.name = "Feature Phone";
        fee = 0;
    }

    @Override //추상 메서드 재정의
    public void WIFIconnect() {
        System.out.println(name+"은 WIFI에 연결할 수 없습니다");
    }
    @Override //추상 메서드 재정의
    public void useInternet() {
        System.out.println(name + "이 요금제 500원을 사용하여 인터넷에 접속합니다.");
        fee += 500;
    }
}
```

Abstract class 예제 4

- Smartphone 클래스

```
public class Smartphone extends MobilePhone {
    String name;
    private boolean wifi;

    public Smartphone() {
        this.name = "Smart Phone";
        this.wifi = false;
    }

    @Override     //추상 메서드 재정의
    public void WIFIconnect() {
        System.out.println(name + "이 WIFI에 연결합니다.");
        wifi = true;
    }
```

Abstract Class 예제 5

- Smartphone 클래스

```
    @Override //추상 메서드 재정의
    public void useInternet() {
        if (wifi) {
            System.out.println(name + "이 WI-FI를 사용하여 인터넷에 접속합니다.");
        } else {
            System.out.println(name + "이 데이터 요금 100원을 사용하여
                                                    인터넷에 접속합니다.");
            fee += 100;
        }
    }
}
```

Abstract Class 예제 5

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        MobilePhone phone1 = new Featurephone();
        MobilePhone phone2 = new Smartphone();

        phone1.usePhone();
        phone2.usePhone();
    }
}
```

Abstract Class 예제 6

- Coffee를 만들어보자

Abstract Class 예제 6

- Coffee Class

```
abstract class Coffee {
    public final void prepareCoffee() {    // Template Method
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }
    private void boilWater() {
        System.out.println("물을 끓입니다.");
    }
    abstract void brew();       // 추상 메소드 (하위 클래스가 구현)
    private void pourInCup() {
        System.out.println("컵에 따릅니다.");
    }
    abstract void addCondiments(); // 추상 메소드 (하위 클래스가 구현)
}
```

Abstract Class 예제 6

- Americano Class

```
class Americano extends Coffee {
    @Override
    void brew() {
        System.out.println("커피 원두를 우려냅니다.");
    }

    @Override
    void addCondiments() {
       System.out.println("아무것도 추가하지 않습니다.");
   }
}
```

Abstract Class 예제 6

- Latte Class

```
class Latte extends Coffee {
    @Override
    void brew() {
       System.out.println("에스프레소를 내립니다.");
    }

    @Override
    void addCondiments() {
        System.out.println("우유를 추가합니다.");
   }
}
```

Abstract Class 예제 6

- Main Class

```
public static void main(String[] args) {
    System.out.println("아메리카노 준비:");
    Coffee americano = new Americano();
    americano.prepareCoffee();

    System.out.println("₩n 라떼 준비:");
    Coffee latte = new Latte();
    latte.prepareCoffee();
}
```

Abstract Class 예제 6

- Template Method Pattern
  - Class 상속을 이용하여 Algorithm의 구조를 Super Class에서 정의하고, 세부적인 구현은 Sub Class에서 담당하도록 하는 Design Pattern

- Template Method
  - Template Method는 Logic 흐름을 정의하는 역할
  - 이 흐름은 모든 Sub Class가 공통으로 사용하고 Code를 변경하면 안 되기 때문에 주로 final로 선언
  - 예) 통화와 문자를 남기고 인터넷에 접속한 후 요금을 청구 받는 과정은 휴대폰의 종류와 상관없이 동일한 과정

Abstract Class 예제 6

- Template Method Pattern의 핵심
  - 공통 Algorithm의 Template을 Super Class에서 정의 (Template Method)
  - 세부적인 단계는 Sub Class에서 구현 (Abstract Method)
  - Method 순서는 변하지 않음 (final Keyword를 사용하여 재정의 방지 가능)

Abstract Class 예제 7

- 운송수단(자동차, 배, 비행기)의 속도를 나타내는 방법이 다양하다

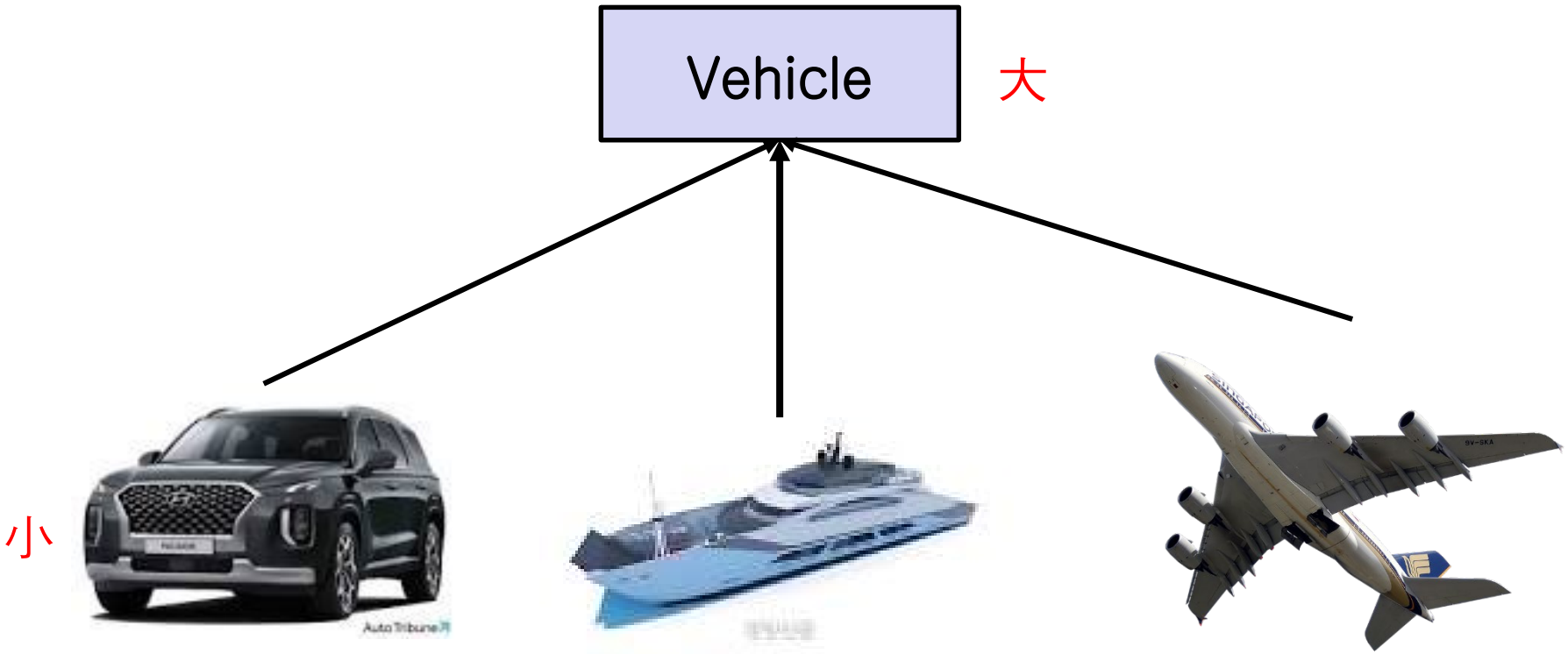
차량번호 :
연료량 :
속도: Km/h

편명 :
속도: knot

편명 :
속도: mile

Abstract Class 예제 7

- 운송수단



Abstract Class 예제 7

- Vehicle 추상 클래스

```
public abstract class Vehicle {
    protected int speed;

    public Vehicle(int speed) {
        this.speed = speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    abstract void print();      // 추상 메소드
}
```

Abstract Class 예제 7

- Car 클래스

```
public class Car extends Vehicle{
    private String number;
    private double gas;

    public Car(String number, double gas, int speed) {
        super(speed);
        this.number = number;
        this.gas = gas;
        System.out.println("차량번호 " + number + ", 연료양 " + gas +
                                                    "인 자동차가 만들어졌습니다.");
    }
    @Override
    void print() {
        System.out.println("차량번호는 " + number + "입니다.");
        System.out.println("연료 양은 " + gas + "입니다.");
        System.out.println("속도는 " + speed + " Km/h 입니다.");
    }
}
```

Abstract Class 예제 7

- Ship 클래스

```
public class Ship extends Vehicle{
    private String vessleName;

    public Ship(String vessleName, int knot) {
        super(knot);
        this.vessleName = vessleName;
        System.out.println("배편 이름 " + vessleName + "인 배가 만들어졌습니다.");
    }

    @Override
    public void print() {              // 추상 메소드 오버라이딩
        System.out.println("배 이름은 " + vessleName + "입니다.");
        System.out.println("속도는 " + speed + " 노트 입니다.");
    }
}
```

Abstract Class 예제 7

- Airplane 클래스

```
public class Airplane extends Vehicle {
    private String flightNo;

    public Airplane(String flightNo, int mile) {
        super(mile);
        this.flightNo = flightNo;
        System.out.println("비행기 번호가 " + flightNo +
                                                "인 비행기가 만들어졌습니다.");
    }

    @Override
    public void print() {              //추상 메소드 오버라이딩
        System.out.println("비행기 번호는 " + flightNo + "입니다.");
        System.out.println("속도는 " + speed + " miles 입니다.");
    }
}
```

Abstract Class 예제 7

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Vehicle[] vehicles = new Vehicle[3];
        vehicles[0] = new Car("85가3456", 20.5, 0);
        vehicles[0].setSpeed(60);
        vehicles[1] = new Airplane("KE 905", 232);
        vehicles[1].setSpeed(1500);
        vehicles[2] = new Ship("퀸메리호",12);
        vehicles[2].setSpeed(40);

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