4/26 스터디노트

- P221 그래픽 카드 예제 : 다형성이라는 개념이 필요한 이유?

✓ 부모클래스 GraphicCard 에 자식클래스 Amd, Nvidia 가 있고, 사용자인 메인 클래스 Computer 가 있다.
GraphicCard 클래스에 그래픽 처리 메서드 process( ); 가 있고,
세부적으로 Amd그래픽카드와 Nvidia그래픽카드가 process( ); 메서드를 약간씩 다르게 재정의한다.

```java
public class GraphicCard {
    int memory;
    public void process(){
        System.out.println("그래픽 처리");
    }
}
```

```java
public class Amd extends GraphicCard {
    public void process(){
        System.out.println("AMD 그래픽 처리");
    }
}
```

```java
public class Nvidia extends GraphicCard{
    public void process(){
        System.out.println("Nvidia 그래픽 처리");
    }
}
```

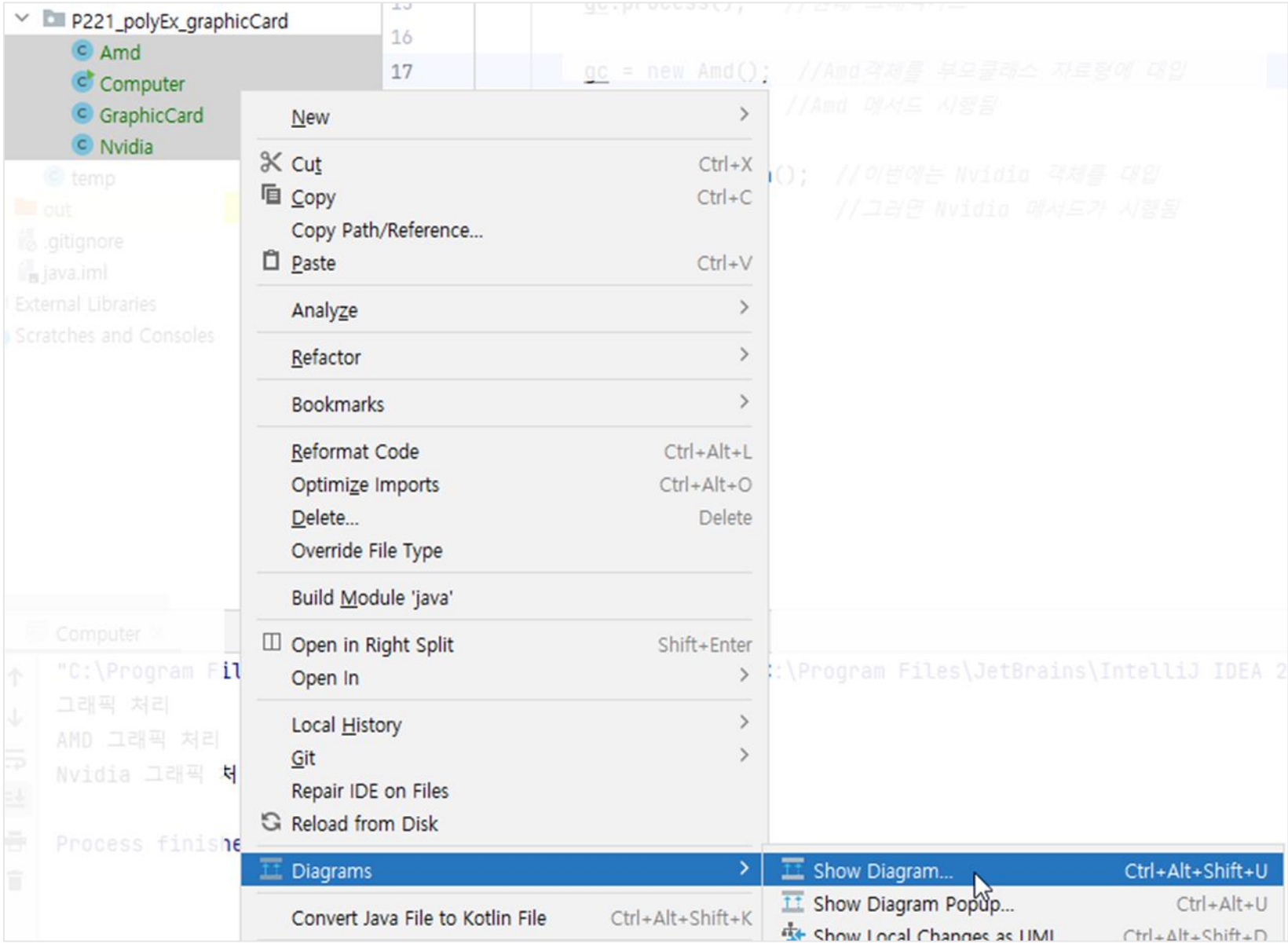
✓ 사용자인 Computer에서 Amd클래스의 객체든 Nvidia클래스의 객체든 일단 부모클래스(GraphicCard)의 자료형으로 형변환을 하면
✓ 사용자인 Computer 클래스에서는 그래픽 관련 업무 처리시에 process( ); 메서드만 사용하면 됨.

```java
public class Computer {
    public static void main(String[] args) {
        GraphicCard gc = new GraphicCard();
        gc.process();   //원래 그래픽카드

        gc = new Amd();  //Amd객체를 부모클래스 자료형에 대입
        gc.process();   //Amd 메서드 시행됨

        gc = new Nvidia();  //이번에는 Nvidia 객체를 대입
        gc.process();       //그러면 Nvidia 메서드가 시행됨
    }
}
```

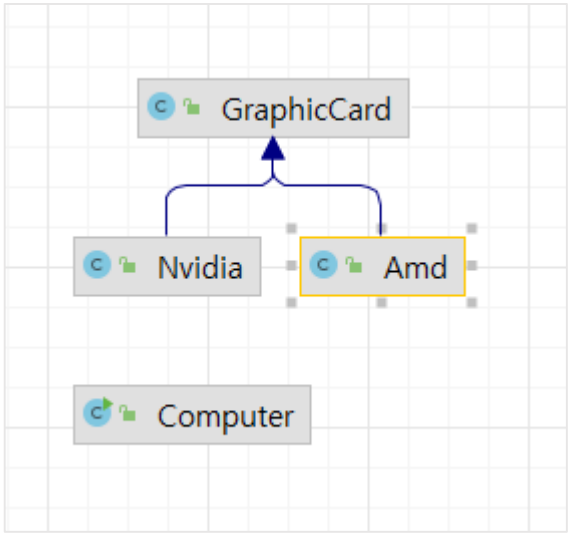
```java
// 사용자는 그래픽처리 관련 업무를 하려고 할 때
// GraphicCard 클래스의 인스턴스를 통한 process() 메소드만 알면 된다.
// Amd, Nvidia는 신경쓰지 않아도 된다.

// Amd, Nvidia는 마치 부품처럼 교체될 수 있다.
// 부모-자식 클래스간에서 자식의 인스턴스를 교체하면
// 동일한 인터페이스를 사용하면서 큰틀에서는 동일한 기능을 수행하지만
// 자식 클래스에서 제공하는 특별한 부가기능을 수행하는 방식으로 객체 지향의 다형성을 지원한다.
```

```
그래픽 처리
AMD 그래픽 처리
Nvidia 그래픽 처리
```

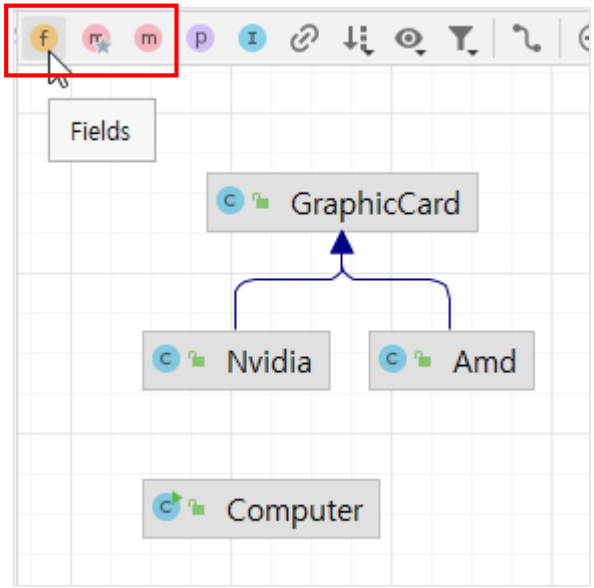
- (교재X) 인텔리제이 기능 - 클래스 다이어그램 보기
  - ✓ 클래스 다이어그램을 함께 보고 싶은 부모-자식-메인 클래스 모두 선택하여 마우스 우클릭



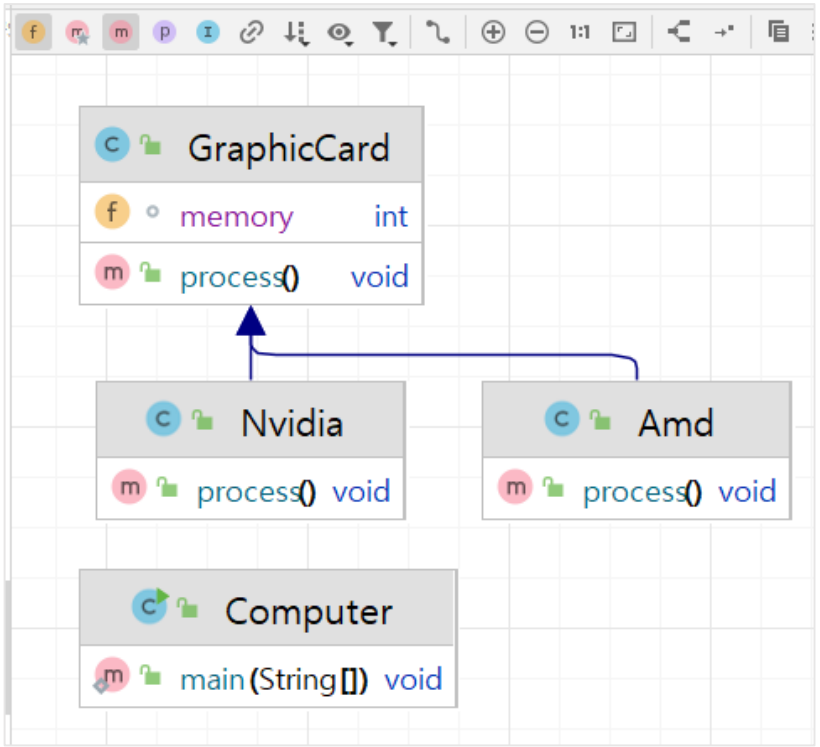
✓ 다이어그램을 볼 수 있음 : 자식 → 부모 형태로 화살표가 있음



✓ 상단 메뉴에서 필드 / 생성자 / 메서드 등 버튼을 클릭하면 클래스별 하위내용을 볼 수 있음



➔



- P223 매개변수의 다형성
  - ✓ P221 GraphicCard 예제에서 process( ); 라는 메서드를 Amd, Nvidia 클래스에서 재정의했던걸로 사용하고 싶으면 메서드 수행 전에 해당 클래스의 객체를 부모클래스 자료형으로 대입해주는 과정이 필요했다.

```
GraphicCard gc = new GraphicCard();
gc.process();   //원래 그래픽카드

gc = new Amd();  //Amd객체를 부모클래스 자료형에 대입
gc.process();   //Amd 메서드 시행됨

gc = new Nvidia();  //이번에는 Nvidia 객체를 대입
gc.process();       //그러면 Nvidia 메서드가 시행됨
```

  - ✓ 위의 형변환 과정 없이 바로 메서드를 수행하고 싶을 때, 메서드의 매개변수를 형변환하는 방법이 있음.

  - ✓ 예제 : Game 이라는 클래스가 있다.

```
public class Game {
    void display(GraphicCard gc){
        gc.process();
    }
    void display(Amd gc){
        gc.process();
    }
    void display(Nvidia gc){
        gc.process();
    }
}
```

➔ process( ); 와 같은 기능을 하는 display 라는 메서드를 만드는데

GraphicCard 클래스의 객체를 매개변수로 받을 수도 있고

Amd 클래스의 객체를 매개변수로 받을 수도 있고

Nvidia 클래스의 객체를 매개변수로 받을 수도 있다.

(메서드 오버로딩)

✓ 사용자인 Computer2 클래스를 만들어 보자

```
public class Computer2 {
    public static void main(String[] args) {
        Game g = new Game();

        GraphicCard gc =new GraphicCard();
        g.display(gc);

        Amd gc2 = new Amd();
        g.display(gc2);

        Nvidia gc3 = new Nvidia();
        g.display(gc3);
    }
}
```

```
그래픽 처리
AMD 그래픽 처리
Nvidia 그래픽 처리
```

➔ display 라는 메서드를 가졌던 Game 클래스의 인스턴스 g를 생성해주고

g에 대해서 display라는 메서드를 수행하면

GraphicCard의 인스턴스인 gc,

Amd의 인스턴스인 gc2,

Nvidia의 인스턴스인 gc3

셋 중 어느것을 매개변수로 넣어도 process( ); 메서드가 동작함.

- P225 (참조) 모든 클래스를 매개변수로 받고 싶으면, 자바 내장 최상위 클래스인 Object 를 활용한다.
  - ✓ Game 처럼 각 클래스의 객체를 매개변수로 받을 수 있도록 display 메서드를 오버로딩(같은 이름을 가졌지만 매개변수가 다른 여러 메소드) 해주지 않고 바로 모든 클래스를 매개변수로 받고 싶을때는 Object 클래스를 활용함.

  - ✓ Object 클래스란 : 자바 내장 최상위클래스로서, 사용자가 정의하는 모든 클래스는 Object 의 자식 클래스가 됨.

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

    allObject(new GraphicCard());
    allObject(new Amd());
    allObject(new Nvidia());
    allObject("안녕"); // new String("안녕") 처럼 스트링 클래스의 인스턴스가 넘어간다.

}

4 usages   KieR402 *
public static void allObject(Object obj) {

    // Object 클래스에서 toString은 객체의 정보를 출력한다.
    // 패키지.클래스명.식별자
    // String클래스인 경우는 toString()메소드를 오버라이딩하였다.
    // String클래스의 toString메소드는 String 클래스의 값을 출력한다.
    System.out.println(obj.toString());

}
```

- P229 추상클래스
  - ✓ 추상클래스란 추상메서드를 1개 이상 가지는 클래스를 말함. 생성 방법 :

```
package chapter08.P229_abstract;

abstract class Shape {
}
```

  - ➢ 추상메서드란
    - ✓ 일반적으로 지금까지 메서드를 선언할 때

[선언부] | *[구현부]*
[접근제한자] [리턴타입] [메서드명] (매개변수) | { 실행문; } 의 형태로 선언해왔음.

ex) `public void display(Amd gc){ gc.process(); }`

    - ✓ 추상 메서드는 선언할 때 접근제한자 뒤에 'abstract' 를 표시하고, 구현부인 **{ }** 부분이 없다.

`[접근제한자] abstract [리턴타입] [메서드명] (매개변수) ;` 의 형태로 선언함.

    - ✓ 추상메서드는 선언될때 '구현부'가 없었기 때문에
추상 클래스의 자식 클래스에서는 상속받은 추상 메소드를 반드시 재정의(Override)해야 한다.

    - ✓ **@Override**: 자바의 어노테이션임. (@[어노테이션] : 해당 어노테이션의 기능을 내부적으로 수행한다.)
@Override 어노테이션을 사용하면 자식클래스에서 추상클래스의 메서드를 override 하지 않을 경우 에러를 발생한다.
만약에 사용하지 않을 경우에는 추상 클래스의 메서드와 동일한 이름으로 작성해도 새로운 메소드로 간주한다.

※ 추상클래스, 추상메서드를 왜 사용하는가?

회사 실무 현장에서 프로그램을 만들 때,

기획 → 요구사항분석 → 설계***(시니어개발자)*** → 구현***(시니어+주니어개발자)*** → 테스트 의 단계를 거치게 됨.

이때 설계단계에서 특정한 기능을 수행하는 메서드를 설계하여도,

자식클래스에서 해당 메서드를 사용하려고 할 때, 클래스마다 메서드의 실행문이 달라져야할 필요성이 있을 수 있음.

따라서 추상클래스에서 메서드를 선언만 하고, 각 자식클래스에서 구현부를 재정의하여 사용함

교재 : 클래스를 설계할 때 변수와 메서드의 이름을 공통적으로 적용 할 수 있음.

중복 소스 줄일 수 있음 (상속할 때 모든 변수와 메서드를 물려받으므로)

다형성의 개념 적용 – 소스의 수정/변경이 있을 때, 전체를 바꾸는 게 아니라 특정 클래스만 새 클래스로 바꾸면 쉽게 수정 가능.

✓ 추상클래스 예제

✓ Shape 라는 부모 클래스에서 넓이와 둘레를 구하는 메서드인 area( ); 랑 length( ); 를 선언하는데,
Circle 이라는 자식 클래스, Rectangle 이라는 자식 클래스 각각 넓이와 둘레를 구하는 식이 다르다.
따라서 Shape 에서 area( ); 와 length( ); 를 추상메서드로 선언하고, Circle과 Rectangel 자식클래스에서 오버라이딩 (재정의) 하여 사용한다.

부모 추상 클래스 Shape

```
abstract class Shape {
    String type;
    Shape(String type){ this.type = type; }
    abstract double area();
    abstract double length();
}
```

자식 클래스 Circle

```
class Circle extends Shape {
    int r;
    Circle(int r){
        super( type: "원");
        this.r=r;
    }
    @Override
    double area(){ return r*r*Math.PI; }
    @Override
    double length(){ return 2*r*Math.PI; }
    @Override
    public String toString(){ return "Shape [type="+type+", r="+r+"]"; }
}
```

자식 클래스 Rectangle

```
class Rectangle extends Shape {
    int width, height;
    Rectangle(int width, int height) {
        super( type: "사각형");
        this.width = width;
        this.height = height;
    }
    @Override
    double area(){ return width*height; }
    @Override
    double length(){ return 2*(width+height); }
    @Override
    public String toString(){ return "Shape [type="+type+", width="+width+", height="+height+"]"; }
}
```

메인 클래스 ShapeEx

```
public class ShapeEx {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[2];
        shapes[0] = new Circle( r: 10);
        shapes[1] = new Rectangle( width: 5, height: 5);
        for(Shape s : shapes){
            System.out.println(s);
            System.out.println("넓이: "+s.area()+" 둘레: "+s.length());
        }
    }
}
```

```
Shape [type=원, r=10]
넓이: 314.1592653589793 둘레: 62.83185307179586
Shape [type=사각형, width=5, height=5]
넓이: 25.0 둘레: 20.0
```

➔ shape 클래스의 객체로 나열된 shapes 라는 배열 생성
첫번째는 Circle 객체
두번째는 Rectangle 객체

for문 사용하여 Shape 배열의 첫번째와 두번째에 대해 실행문 수행

- P232 객체를 배열로 처리하기
  - ✓ 서로 다른 자식 클래스를 공통의 부모클래스의 배열로 관리할 수 있음.
    (앞의 예제에서도 ShapeEx 클래스에서 자식인 Circle, Rectangel 을 부모인 Shape 의 배열로 처리한 것임)
  - ✓ 자식클래스는 부모클래스로 형변환이 가능하고 자신의 특성, 메서드를 유지할 수 있기 때문이다.

부모 클래스 Animal

```
public class Animal {
    String type;
    String name;

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

    void sleep(){
        System.out.println(this.name+"은(는) 잠을 잔다.");
    }
}
```

자식 클래스 Eagle, Lion, Tiger, Shark

```
public class Eagle extends Animal{
    public Eagle(String type, String name) {
        super(type, name);
    }

    void sleep(){
        System.out.println(this.name+"은(는) 하늘에서 잠을 잔다.");
    }
}
```

```
public class Lion extends Animal{
    public Lion(String type, String name) { super(type, name); }

    void sleep() { System.out.println(this.name+"은(는) 숲속에서 잠을 잔다."); }
}
```

```
public class Tiger extends Animal{
    public Tiger(String type, String name) { super(type, name); }

    void sleep() { System.out.println(this.name+"은(는) 산속에서 잠을 잔다."); }
}
```

```
public class Shark extends Animal{
    public Shark(String type, String name) { super(type, name); }

    void sleep() { System.out.println(this.name+"은(는) 물속에서 잠을 잔다."); }
}
```

메인 클래스 AnimalMain

```java
public class AnimalMain {
    public static void main(String[] args) {
        Animal[] ani = new Animal[4];

        Animal eagle = new Eagle( type: "조류",  name: "독수리");
        Animal tiger = new Tiger( type: "포유류",  name: "호랑이");
        Animal lion = new Lion ( type: "포유류",  name: "사자");
        Animal shark = new Shark ( type: "어류",  name: "상어");

        ani[0] = eagle;
        ani[1] = tiger;
        ani[2] = lion;
        ani[3] = shark;

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

```
독수리은(는) 하늘에서 잠을 잔다.
호랑이은(는) 산속에서 잠을 잔다.
사자은(는) 숲속에서 잠을 잔다.
상어은(는) 물속에서 잠을 잔다.
```

➔ Animal 클래스의 객체로 나열된 ani 라는 배열 생성
자식클래스 Eagle, Tiger, Lion, Shark 의 인스턴스 생성

첫번째는 Eagle 클래스 객체 eagle
두번째는 Tiger 클래스 객체 tiger
세번째는 Lion 클래스 객체 lion
네번째는 Shark 클래스 객체 shark

for문 사용하여 ani 배열에 대해
실행문 sleep( ); 수행

- (교재x) 선생님 추상클래스 사용 예시 RealAbstractClassExample
  - ✓ Integer.parseInt( ); : Integer 클래스의 parseInt 메서드임. string 문자형을 매개변수로 받아서 int 정수형으로 반환해줌
  - ✓ Double.parseDouble(); : Double 클래스의 parseDouble 메서드임. string 문자형을 매개변수로 받아서 double 실수형으로 반환해줌

  - ✓ Integer와 Double 클래스는, 사실은, 추상클래스인 Number 클래스의 자식 클래스임.
    하지만 우리는 Number 클래스를 모르는 채로 parse 메서드를 사용함

```java
package chapter08.P235_realAbstractEx_hyunkooEx;
import java.util.Scanner;
public class RealAbstractClassExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("첫 번째 수를 입력하세요: ");
        String num1 = scanner.next();

        System.out.print("두 번째 수를 입력하세요: ");
        String num2 = scanner.next();
    //정수의 덧셈
        int result = Integer.parseInt(num1) + Integer.parseInt(num2);

    //실수의 덧셈
        double result = Double.parseDouble(num1)+Double.parseDouble(num2);

        System.out.println(num1+" " +num2 + " = "+result);

    }
}
```

- P235 final 제어자
  - ✓ final 이 변수 앞에 붙으면 더 이상 값이 변하지 않는 변수라고 했었음.

➢ final 클래스

- ✓ 더 이상 상속이 불가능한 클래스라는 것을 의미한다. = 다른 클래스의 부모클래스가 될 수 없다.
- ✓ 대표적인 final 클래스의 예시로는 String, Math 등이 있다.

```
public class finalClass_child extends finalClass_parent{

}


public final class finalClass_parent{

}
```

➔ final 클래스로 생성된 finalClass_parent 를 finalClass_child 클래스에서 상속받으려고 시도하면 에러발생

➢ final 메서드

✓ 오버라이딩(재정의)가 불가능한 메서드.

✓ 부모클래스에서 메서드를 final 로 선언하면 해당 메서드를 상속받은 자식클래스에서 변경(재정의)할 수 없다.

```
public class FinalMethod {

    void method(){ }

    final void finalMethod(){ }

}


class SubFinalMethod extends FinalMethod{

    void method(){ System.out.println("method 재정의"); }

    void finalMethod(){ System.out.println("method 재정의 불가"); }

}
```

➔ 부모클래스인 FinalMethod 에서

finalMethod( ); 를 final 메서드로 선언했음

따라서 자식클래스인 SubFinalMethod 에서

finalMethod( ); 를 재정의하려고 하면 에러발생

- P237 연습문제 풀이
  - ✓ 1번 화면 출력 순서는 무엇인가 2143

```
1. package chapter08;
2.
3. public class Excercise1 {
4.
5.     public static void main(String[] args) {
6.
7.         ChildEx ce = new ChildEx();
8.
9.     }
10.
11. }
12.
13. class ParentEx {
14.     ParentEx() {
15.         this (1);
16.         System.out.println("(1)");
17.     }
18.     ParentEx(int x) {
19.         System.out.println("(2)");
20.     }
21. }
22.
23. class ChildEx extends ParentEx {
24.     ChildEx() {
25.         this (1);
26.         System.out.println("(3)");
27.     }
28.     ChildEx(int x) {
29.         System.out.println("(4)");
30.     }
31. }
```

풀이

컴파일순서 (행번호)

5번 7번 23번 14번 15번 18번 (2)출력

15번 16번 (1)출력

24번 25번 28번 (4)출력

25번 26번 (3)출력

✓ 2번 : 클래스 객체간 형변환으로 올바르지 않은 것 : 3

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

    ChildEx ce = new ChildEx();
    ParentEx pe = new ParentEx();
    /*
    pe = ce;            // 1
    pe = (ParentEx)pe;  // 2
    ce = pe;            // 3
    ce = (ChildEx)pe;   // 4
    */
}
```

풀이

자식은 부모가 될 수 있고 부모는 자식이 될 수 없다.

자식 → 부모 (가능) / 부모 → 자식 (불가능)

⇨ **java 코드인 =(대입연산자) 사용하여 표현한다면**

(부모) = (자식) 은 가능하지만

(자식) = (부모) 는 불가능

☆ 부모가 자식되는 건 안 되지만

자식이 부모가 되었다가 다시 자식이 되는 건 가능(강제형변환 필요)

1. 자식이 부모되기
2. 1번에서 부모가 된 자식을 부모로 강제형변환 혹은 자기자신을 자신으로 형변환(?)
3. 부모가 자식되기: 자동 형변환 불가
4. 1번에서 부모가 된 자식을 부모로 강제형변환

✓ 3번 : 아래 3개의 클래스를 보고 공통적인 부분을 뽑아서 상위클래스(Character)을 만들어봐라

```java
class Warrior {
    int hp;
    int power;
    int weapon;
    public void attack(Object target){ System.out.println("공격"); }
    public void defence(Object target){ System.out.println("방어"); }
}

class Gladiator {
    int hp;
    int power;
    int shield;
    public void attack(Object target){ System.out.println("공격"); }
    public void powerattack(Object target){ System.out.println("파워공격"); }
}

class Wizard {
    int hp;
    int power;
    int heal;
    public void attack(Object target){ System.out.println("공격"); }
    public void healing(Object target){ System.out.println("치료"); }
}
```

풀이

클래스 다이어그램으로 정리하기

클래스	Warrior	Gladiator	Wizard
필드	hp power weapon	hp power shield	hp power heal
메서드	attack( ); defence( );	attack( ); powerAttack( );	attack( ); healing( );

→

Character
hp power
attack( );

공통된 부분만 확인하여 상위클래스로 표현

```java
class Character {
    int hp;
    int power;
    public void attack(Object target)
     { System.out.println("공격"); }
}

class Warrior extends Character {
    int weapon;
    public void defence(Object target)
     { System.out.println("방어"); }
}
class Gladiator extends Character {
    int shield;
    public void powerattack(Object target)
     { System.out.println("파워공격"); }
}
class Wizard extends Character {
    int heal;
    public void healing(Object target)
     { System.out.println("치료"); }
}
```

✓ 4번 : final 에 대한 설명으로 올바른 것은?

1) final 필드는 값이 대입되면 수정할 수 없다 : ㅇㅇ final은 안돼 못 바꿔줘 돌아가

2) final 메서드는 재정의 할 수 있다. : x 재정의 불가

3) final 클래스는 상속 받을 수 있다. : x final클래스는 부모클래스가 될 수 없다.

4) 상수는 private final 로 정의한다. : 상수는 static final 로 정의함. 꼭 private 일 필요 없음

Chapter 09 인터페이스

● 선언하기

✓ 기본 접근 제한자는 public 임.

✓ 인터페이스의 필드는 상수만 선언 가능

: public static final 로 선언해야하며 반드시 선언할 때 초기값 넣어주기

ㄴ (static final 을 생략해도 컴파일시 자동 포함됨)

✓ 인터페이스의 메서드는 기본적으로 추상메서드

: public abstract 로 선언해야하며, 생략해도 컴파일과정에서 자동으로 포함해줌.

✓ 자바 7버전 까지는 인터페이스에는 추상메서드만 가능했지만 자바 8버전 부터는 구현부가 있는 default 메서드 사용 가능.

: 'default' 라는 키워드를 반드시 입력해야함 (접근제한자는 생략하면 public 으로 자동 실행됨)

✓ 다른 클래스에서 사용가능한 상수, 추상메서드 등을 가지는 인터페이스는 기능, 동작에 대한 설계로서 사용함

```java
public interface P243_interfaceCreate {

 //인터페이스의 필드 : 상수만 선언
    public static final int MIN_PRICE = 0;
    public int MAX_PRICE = 100000;  // public static final 키워드 생략 가능

 //인터페이스의 추상메서드
    public abstract double meanPrice();
    public double totalPrice();   // public abstract 키워드 생략가능

 //default 메서드 (구현부가 있음)
    default double getSalePrice(double price){
        return price-(price*0.05);
    }
}
```

- 인터페이스 구현하기 (클래스의 상속과 같은 개념)
  - ✓ 클래스를 상속할 때 **[자식클래스명] extends [부모클래스명]** 으로 사용했음.
  - ✓ 인터페이스의 추상메서드를 클래스에서 상속받아서 구현하여(정의하여) 사용한다 = 클래스에서 인터페이스를 "구현한다." 라고 표현함.
  - ✓ 인터페이스는 구현할 때 **[구현클래스명] implements [인터페이스명]** 으로 사용함

  - ✓ 자식클래스는 오직 하나의 부모클래스를 상속받을 수 있다. (자바는 다중상속 불가)
  - ✓ 구현클래스는 여러 인터페이스를 구현할 수 있음.(인터페이스를 통해 다중상속의 효과를 냄)

  - ✓ 추상클래스와 인터페이스의 공통점은 모두 설계를 담당한다는 점.
    자식클래스와 구현클래스의 공통점은 모두 구현을 담당한다.

✓ P246-P248 인터페이스 구현 예제

```java
public interface Printer {
    int INK = 100;
    void print();
}
```

```java
public interface Fax {
    String FAX_NUMBER = "02-1234-5678";
    void send(String tel);
    void receive(String tel);
}
```

```java
public interface Scanner {
    void scan();
}
```

인터페이스 Printer, Scanner, Fax 만들기

: Printer 에서는 상수 **INK** 를 선언했고
: Fax 에서는 상수 **FAX_NUMBER** 를 선언함.

(상수는 보통 대문자로 표현한다.)

각각의 인터페이스에 적당한 이름의 추상메서드 선언.
ex) Printer 에는 print( );를, Scanner 에는 scan( ); 을 선언함

```java
public class Complexer implements Printer, Scanner, Fax {
    @Override
    public void send(String tel){
        System.out.println(FAX_NUMBER+"에서 "+tel+"로 Fax 전송");
    }
    @Override
    public void receive(String tel){
        System.out.println(tel+"에서 "+FAX_NUMBER+"로 Fax 수신");
    }
    @Override
    public void scan(){
        System.out.println("스캔 실행");
    }
    @Override
    public void print(){
        System.out.println("출력 실행");
    }
}
```

Printer, Scanner, Fax 인터페이스를 구현하는

Complexer 라는 클래스 생성

➔ 구현하는 인터페이스의 모든 추상메서드를 꼭 전부 구현해야함. (안하면 에러남).

따라서 send( );, receive( );, scan( );, print( ); 모두 구현부를 정의해줌

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

        Complexer com = new Complexer();

        System.out.println(Complexer.INK);
        System.out.println(Complexer.FAX_NUMBER);

        com.print();
        com.scan();
        com.send( tel: "02-8765-4321");
        com.receive( tel: "02-8765-4321");
    }
}
```

Printer, Scanner, Fax 인터페이스를 구현하는 Complexer 라는 클래스를 호출하는 클래스인 ComplexerMain 클래스를 생성

Complexer 클래스의 객체를 생성해주고

➔ Complexer 클래스에서 구현했던

인터페이스들의 상수와 메서드를 실행시켜본다

```
100
02-1234-5678
출력 실행
스캔 실행
02-1234-5678에서 02-8765-4321로 Fax 전송
02-8765-4321에서 02-1234-5678로 Fax 수신
```

- 추상클래스와 인터페이스 차이 (선생님 설명)

비행기를 탔을 때 나오는 기내안내방송 예시

* 규약: 반드시, (must)

You **must** remain seated and keep your seat belts securely fastened at all times while the seat belt sign is on.

반드시 좌석에 앉아서 안전벨트를 매시라~~

* 권고(규약 보다 강제성이 낮은): ('have to, need to, should ~ 'd better)

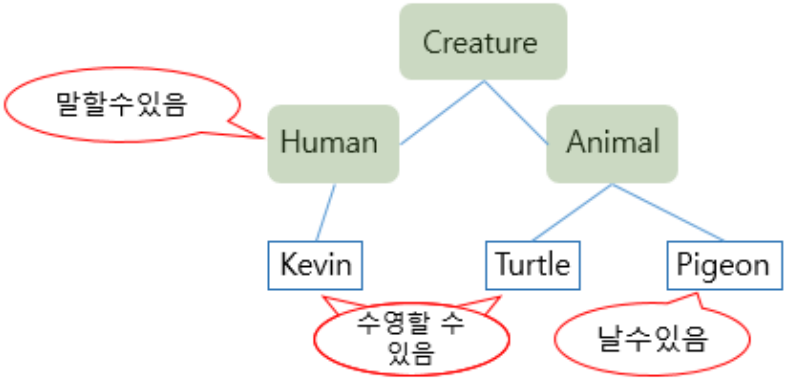
We have a few requests to mak.

First, You**'d better** avoid consuming alcohol excessively during the flight as it may cause discomfort and affect your health.

Second, ... 당신의 건강과 편안한 비행을 위해서 과음하지 않도록 **부탁드립니다**~

Last, ..

Thank you for your cooperation.



추상클래스는 규약임. 근본, 뿌리, base 의 개념

인터페이스는 권고임. 기능의 확장.

초록 동그란 네모는 추상클래스

빨간 테두리 말풍선은 인터페이스

파란 네모는 클래스라고 할 수 있음

- 선생님 예시 _ 로봇 설계하기 : RobotMain

➢ **1단계** 구상 및 설계하기

```
* 로봇 (핵심기능)
> 데이터, 상태
 - 베터리 잔량
> 기능
 - 전원을 킨다.
 - 전원을 끈다.
```

```
* 로봇 (확장기능)
 > 이동
 - 전진
 - 후진
 - 좌회전
 - 우회전

> 인공지능
 - 장애물 감지
 - 대화
```

```
> 청소
 - 쓸기
 - 닦기
```

➔ 핵심기능은 추상클래스로,
확장기능은 인터페이스로 구현할 수 있음.

➢ **2단계** 추상클래스 만들기

```
abstract class Robot{
    int batteryLevel;
    public abstract void activate();
    public abstract void deactivate();
}
```

➢ **3단계** 인터페이스 – 이동 기능 만들기

```
interface Moveable{
    void moveForward();
    void moveBackward();
    void turnLeft();
    void turnRight();
}
```

- 인공지능 기능 만들기

```
interface AI{
    void detectObstruction();
    void talk();
}
```

- 청소기능 만들기

```
interface Cleanable{
    void sweep();
    void mop();
}
```

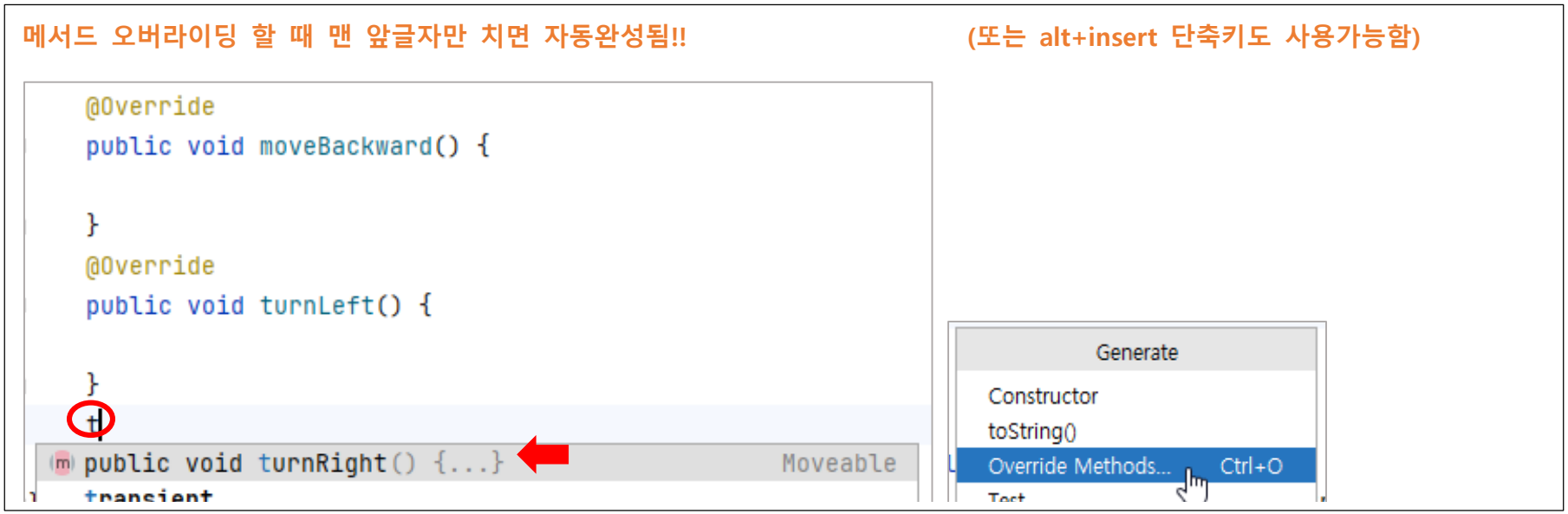
➢ 4단계 추상클래스를 상속받고 인터페이스를 구현하는 개별의 클래스 만들기

- 로봇 핵심기능과 확장기능 중 이동 기능이 있는 RcBot 클래스

```
class RCBot extends Robot implements Moveable{
    @Override
    public void activate() { System.out.println("RC봇 전원을 킵니다."); }
    @Override
    public void deactivate() { System.out.println("RC봇 전원을 끕니다."); }
    @Override
    public void moveForward() { System.out.println("전진합니다."); }
    @Override
    public void moveBackward() { System.out.println("후진합니다."); }
    @Override
    public void turnLeft() { System.out.println("좌회전합니다."); }
    @Override
    public void turnRight() { System.out.println("우회전합니다."); }
}
```

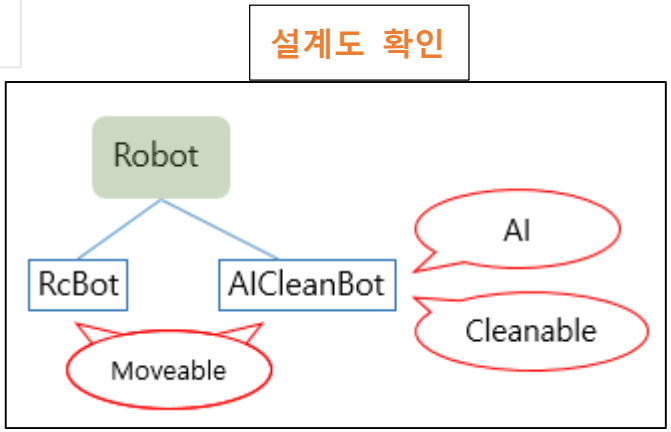
메서드 오버라이딩 할 때 맨 앞글자만 치면 자동완성됨!!

(또는 alt+insert 단축키도 사용가능함)



- 로봇 핵심기능과 확장기능 전부 다 가지는 인공지능 청소로봇인 AICleanBot 클래스

```
class AICleanBot extends Robot implements Moveable, AI, Cleanable{
    @Override
    public void activate() { System.out.println("AIClean봇을 활성화시킵니다."); }
    @Override
    public void deactivate() { System.out.println("AIClean봇을 종료합니다."); }
    @Override
    public void moveForward() { System.out.println("앞으로 이동합니다."); }
    @Override
    public void moveBackward() { System.out.println("후진합니다."); }
    @Override
    public void turnLeft() { System.out.println("좌회전하여 직진합니다."); }
    @Override
    public void turnRight() { System.out.println("우회전하여 직진합니다."); }
    @Override
    public void detectObstruction() { System.out.println("장애물을 감지합니다."); }
    @Override
    public void talk() { System.out.println("대화모드로 전환합니다."); }
    @Override
    public void sweep() { System.out.println("바닥쓸기를 시작합니다."); }
    @Override
    public void mop() { System.out.println("바닥닦기를 시작합니다."); }
}
```



➢ 5단계 : 메인클래스인 RobotMain 클래스를 생성하여 로봇을 동작시켜 본다.

```java
public class RobotMain {
    public static void main(String[] args) {
        RCBot rcBot = new RCBot();
        rcBot.activate();
        rcBot.moveForward();
        rcBot.turnLeft();
        rcBot.moveForward();
        rcBot.turnRight();
        rcBot.moveBackward();
        rcBot.deactivate();

        AICleanBot cleanee = new AICleanBot();
        cleanee.activate();
        cleanee.moveForward();
        cleanee.sweep();
        cleanee.turnLeft();
        cleanee.turnRight();
        cleanee.mop();
        cleanee.moveBackward();
        cleanee.deactivate();
    }
}
```

```
RC봇 전원을 킵니다.
전진합니다.
좌회전합니다.
전진합니다.
우회전합니다.
후진합니다.
RC봇 전원을 끕니다.
AIClean봇을 활성화시킵니다.
앞으로 이동합니다.
바닥쓸기를 시작합니다.
좌회전하여 직진합니다.
우회전하여 직진합니다.
바닥닦기를 시작합니다.
후진합니다.
AIClean봇을 종료합니다.
```