spring 한스푼(2주차)

1주차_review / 상속 / 추상클래스/인터페이스

review

String

- string이란 하나의 자바 라이브러리로써 char과 다르게 하나의 문자열이다

클래스(class)

- 객체의 속성과 기능을 코드로 구현한 것. 선언한 것
- “클래스를 정의한다”라고 함

객체(object)

- 프로그램 실행중에 생성되는 실체. 메모리 공간을 갖는 구체적인 실체

String 쓰는 법.

String name;

name = “박성수”;

System.out.println(name)

```java
public class ClassMain {
    public static void main(String[] args) {
        String name;
        name = "홍길동";
        System.out.println(name);
    }
}
```

```
ClassMain
C:\Users\pc\.jdks\corretto-17.0.8.1\bin\java.
홍길동

Process finished with exit code 0
```

클래스 사용법

public class ClassEx { <- src생성하면 바로 나오는 것이 클래스

↑(명명 규칙 PascalCase : 모든 단어에서 첫 번째 문자는 대문자이며 나머지는 소문자이다.)

```
    String name; ┐
    int age;     ┘ 멤버변수(필드)
    public void print(){
        print("이름: "+name+"나이 : "+age); -> 멤버함수(메소드)
    }
}
```

객체 사용법

main(){

ClassEx class_ex = new ClassEx(); (객체 생성)

(클래스) (클래스변수명) (생성자 디폴트);

class_ex.name = “홍길동”; (객체 class_ex의 name을 “홍길동”으로 저장)

class_ex.age = 24; (객체 classex의 age를 24로 저장)

class_ex.print() (객체 class_ex의 함수 호출)

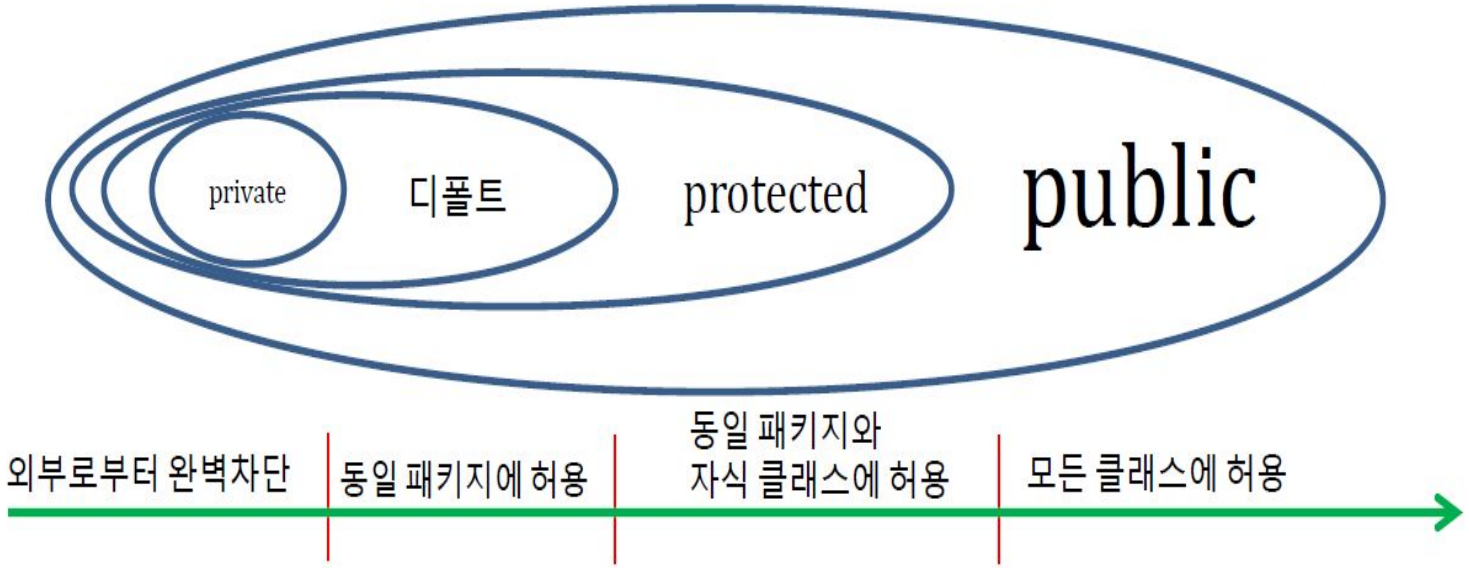
}

접근 제어자에 대해

클래스에서의 접근 제어자.

public ClassEx{} : 프로젝트 내 모두 접근 허용.

ClassEx{} : 패키지(같은 폴더) 내 접근 허용.



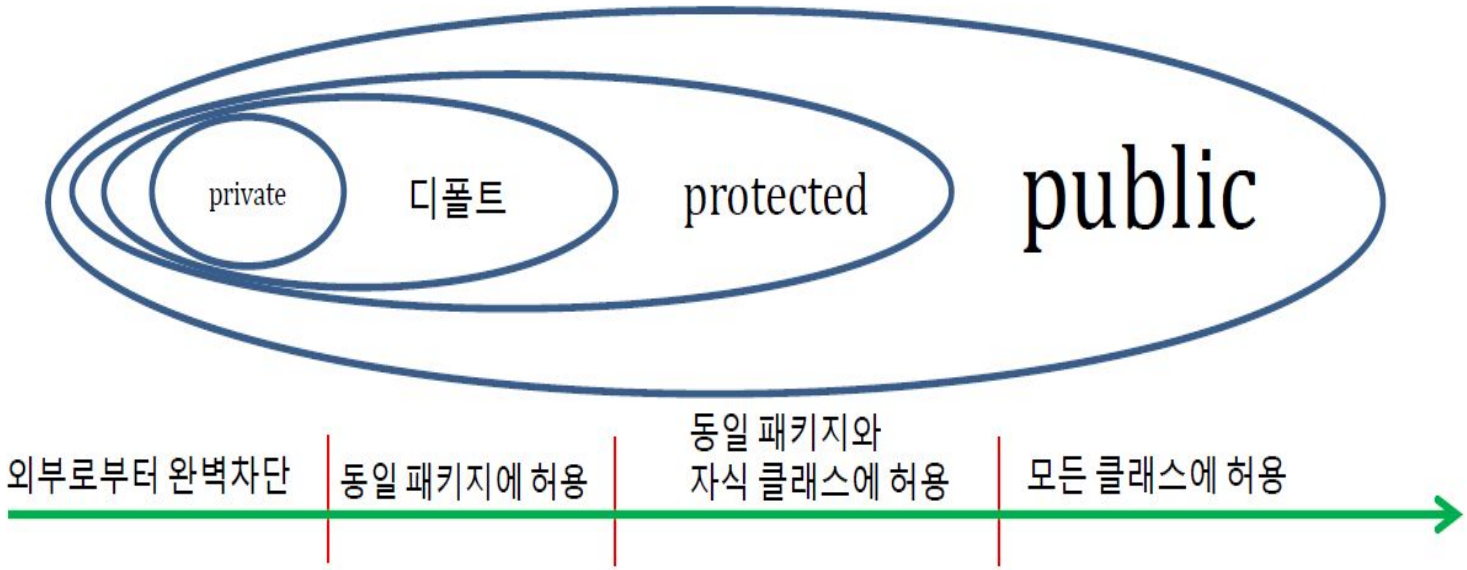
멤버 변수에서 접근제어자

private String name;

클래스 내 접근 허용 나머지 접근 불가. (정보 은닉 및 객체 보호)

String name;

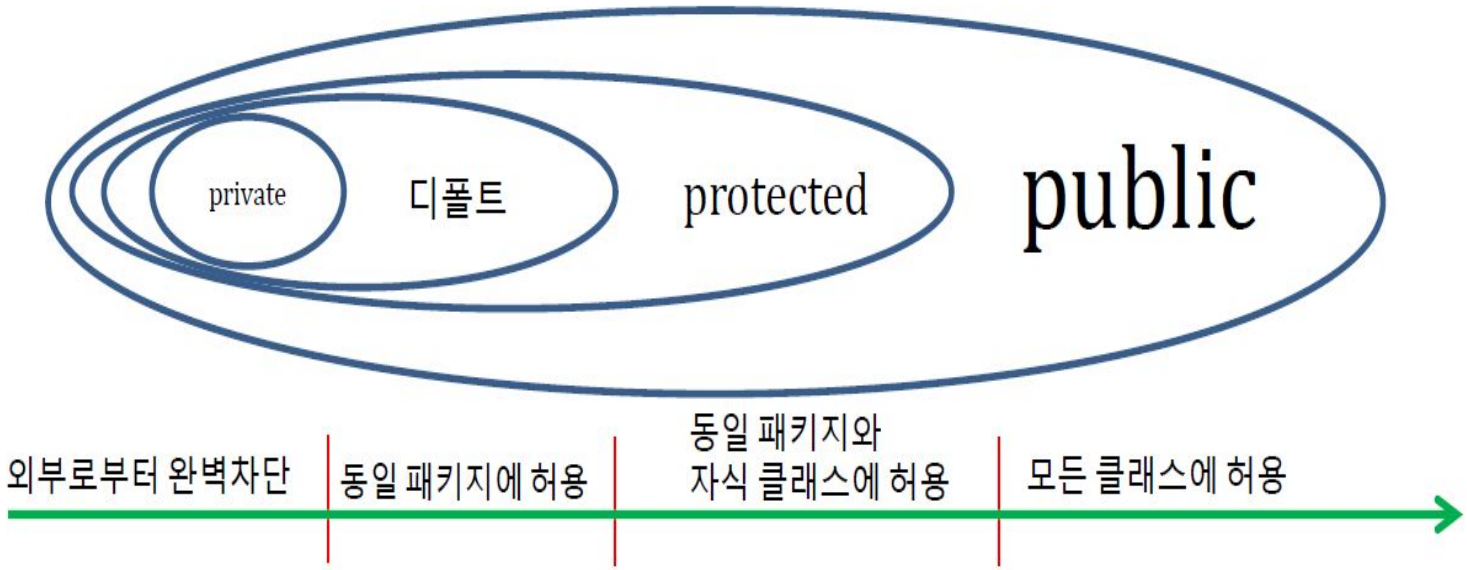
패키지(같은 폴더) 내 접근 허용.



멤버 함수에서 접근제어자

public void print(){} : 프로젝트 내 모두 접근 허용

String toString(){} : 패키지(같은 폴더) 내 접근 허용



this와 생성자

this 왜 쓰는 걸까?

- 객체의 같은이름의 매개변수와 멤버변수 구분지어 접근하기 위함(효율↑)
- 생성자가 다른 생성자를 호출할 때 사용

this 종류

- this 멤버변수
- this 멤버함수(생성자 전용)

생성자?

- main함수에서 객체를 생성하는 함수

생성자 종류

- 디폴트 생성자
- 매개변수 생성자

this 예약어 쓰는 예

```
public class ClassEx{
    1 usage
    String name;
    no usages
    public ClassEx(){
        this( name: "홍길동");
    }
    1 usage
    public ClassEx(String name){
        this.name=name;
    }
}
```

```
public class ClassMain {
    public static void main(String[] args) {
        ClassEx classEx = new ClassEx();
        System.out.println(classEx.name);
        ClassEx classEx2 = new ClassEx( name: "김연아");
        System.out.println(classEx2.name);
    }
}
```

```
ClassMain
C:\Users\pc\.jdks\corretto-17.0.8.1\bin\java.exe "-jav
홍길동
김연아
```

getter와 setter

getter? setter?

- 멤버변수의 값을 변경해야 할 때(setter)
- 멤버변수의 값을 가져와야 할 때(getter)

왜 필요할까?

- 무결성 보장(private로 접근이 불가한 변수를 안전하게 변경 및 제공 가능)

  데이터의 정확성과 일관성을 유지하고 보증하는 것을 말함.

  + setter는 java에서 지양하는 게 많다(builder를 지향)

getter/setter 예시

```java
public class ClassEx{
        private String name;
        private int age;
        private String address;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    ...

}
```

```java
public class ClassMain {
    public static void main(String[] args) {
        ClassEx classEx = new ClassEx();
        classEx.setName("김연아");
        ClassEx classEx1 = new ClassEx();
        classEx1.setName("손흥민");
        System.out.println(classEx.getName()+"/"+classEx1.getName());
    }
```

```
ClassMain
C:\Users\pc\.jdks\corretto-17.0.8.1\bin\java.exe "-javaagent:C:\Progr
김연아/손흥민

Process finished with exit code 0
```

직접 해보기!

나의 정보를 적어보자!

클래스 이름 - MyInfo

멤버 변수들 - 모두 private로 {String name, int age,

String student_id, String school_name}

멤버 함수들 - {public void printAll(), getter/setter}(

public MyInfo(){

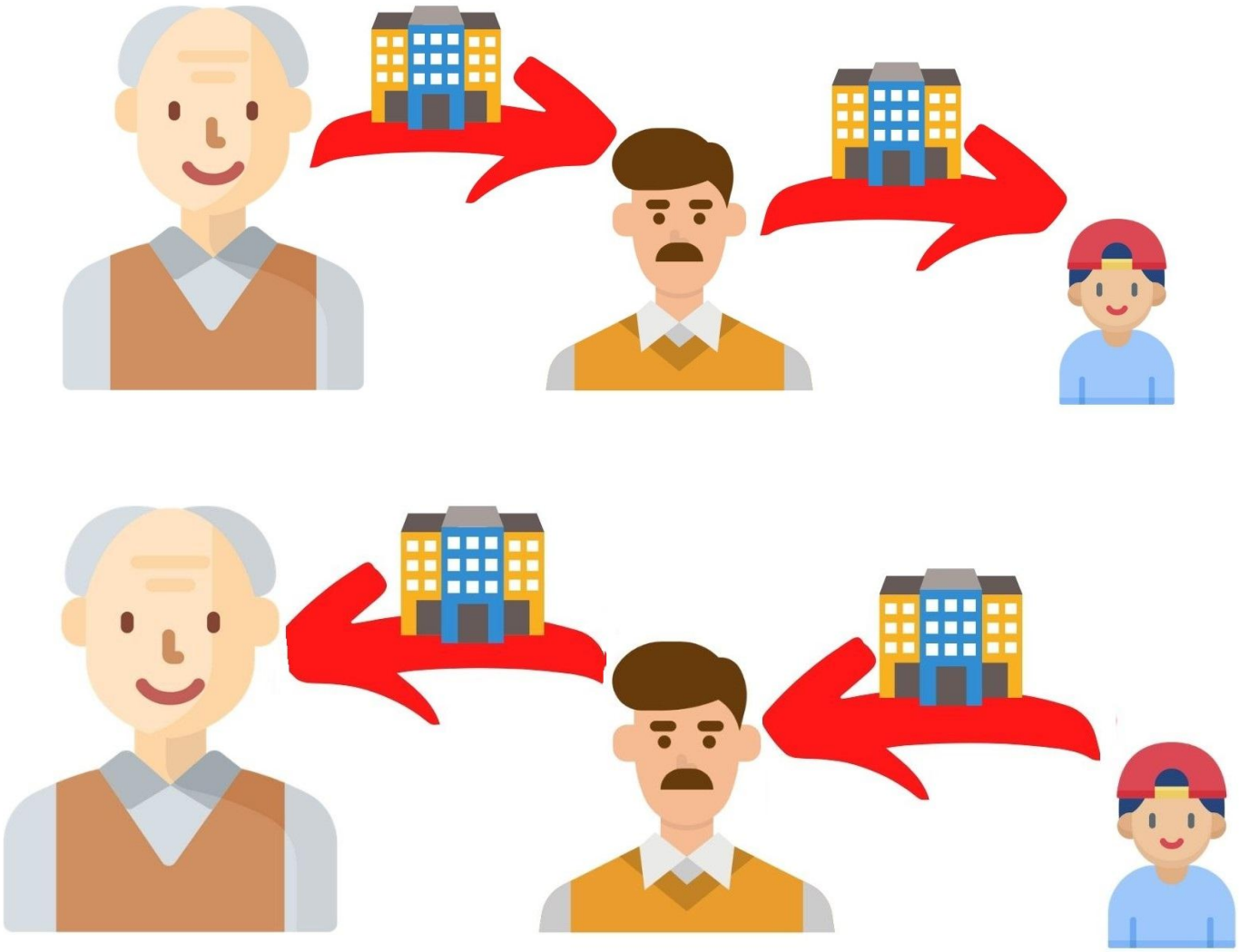
school_name = “한국IT전문직업학교”

}

getter/setter 단축키

윈도우 - Alt+Insert / 맥 - Command + N

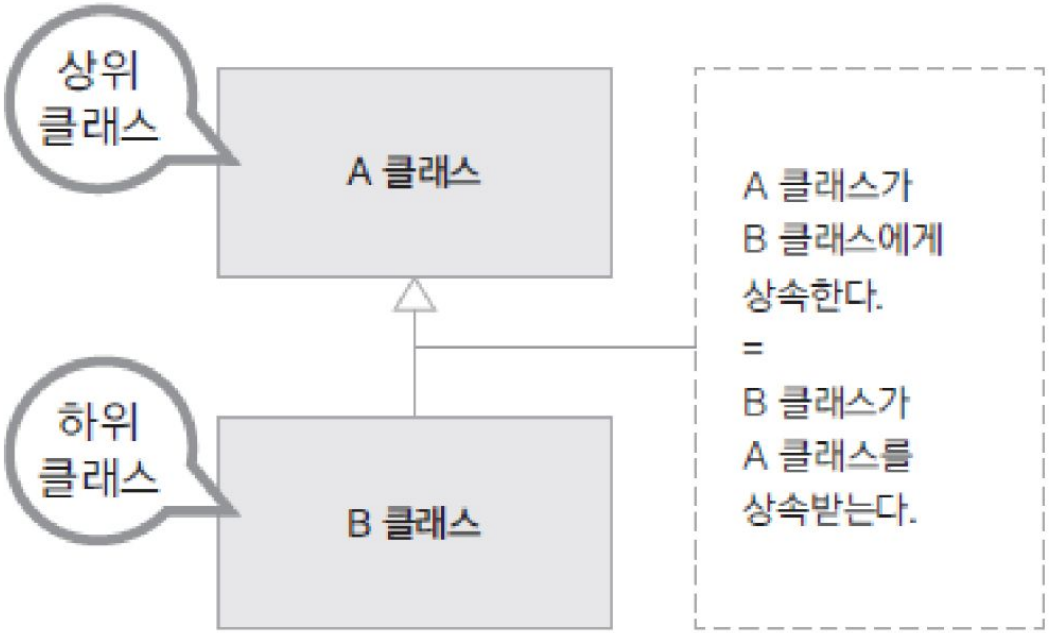
미리보기(상속)



상속이란?

- 자식(클래스)이 상속받고 싶은 부모(클래스)를 선택해서 물려 받는 것.

- 상속 받은 클래스를 **자식 클래스**, 상속을 해주는 클래스를 **부모 클래스**



상속하는 법

```
2개 상속자
public class Animal {
    protected int a;
    1개 재정의
    public void sleep(){
        System.out.println("잠을 잔다");
    }
    1개 재정의
    public void eat() { System.out.println("먹다."); }
}
```

부모 클래스(Animal)

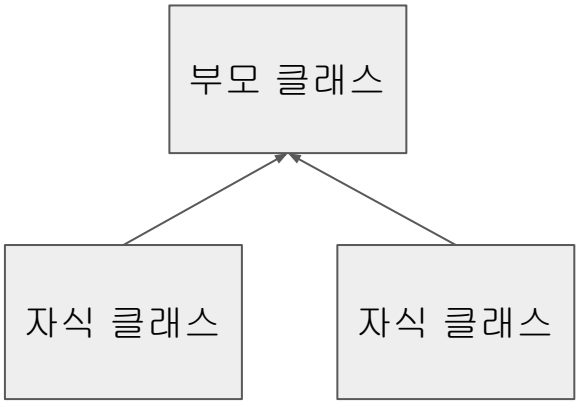
```
public class Eagle extends Animal{
    public void fly() { System.out.println("날다"); }
    @Override
    public void sleep() { System.out.println("서서 자다"+a); }
}
```

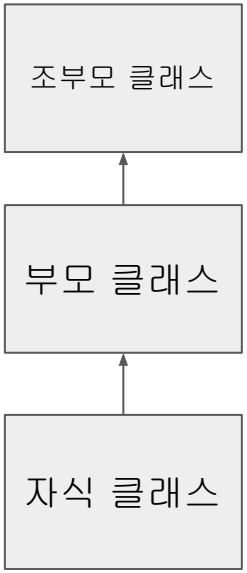
자식 클래스(Eagle)

```
public class Lion extends Animal{
    @Override
    public void eat() { System.out.println("씹어 먹다"); }
    public void run() { System.out.println("달리다"+a); }
}
```

자식 클래스(Lion)

class (자식)클래스 extends (부모)클래스, (부모)클래스2 { .. }	불가
class (자식)클래스1 extends (부모)클래스 { .. } class (자식)클래스2 extends (부모)클래스 { .. }	가능
class (부모)클래스 extends (조부모)클래스 { .. } class (자식)클래스 extends (부모)클래스 { .. }	가능





상속이 필요한 이유!

- 중복된 코드를 줄일 수 있다.
- 유지 보수↑(오류를 쉽게 찾고, 고치기 쉬움)
- 통일성이 있다.
- 다형성을 구현할 수 있다.(멤버십 프로그램)

같이 해보기!

- 동물클래스를 만들어 구현하고 상속받아 새클래스를 구현해보자!

- 차 클래스를 만들어 구현하고 상속받아 기아와 현대차를 구현해보자!

직접 해보기!

customerName – 고객 이름

customerGrade – 고객 등급 (생성시 기본값 SILVER)

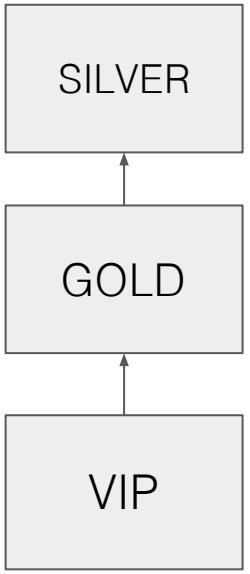
bonusPoint – 고객의 보너스 포인트

bonusRatio – 보너스 포인트 적립 비율 10

멤버 변수들 모두 private로 설정.

int price(int a) – 구매시 적립하는 코드+ 금액 리턴

void print() – 고객의 정보 모두 출력



Gold는 위에 부분 포함 + grade : GOLD + 적립 비율 15

VIP는 Gold 부분 포함 + grade : VIP + 적립 비율 20 + 상담사 agent() 멤버함수 추가

agent() – `안녕하십니까 길동 고객님, 무엇을 도와드릴까요?`

미리보기(추상 클래스)

추상 클래스란?

- 추상 메서드를 선언해 놓고 상속을 통해 자식 클래스에서 메서드를 완성하도록 유도하는 클래스
- 추상 클래스는 메인 함수에 new(인스턴스 화)선언 할 수 없음.
- 예약어 abstaract

```
// 추상 메소드를 가진 추상 클래스
abstract class Shape {
    public Shape() { ... }
    public void edit() { ... }

    abstract public void draw(); // 추상 메소드
}
```

```
// 추상 메소드 없는 추상 클래스
abstract class Component {
    String name;
    public void load(String name ) {
        this.name= name;
    }
}
```

추상 메소드

- 구현 코드 없이 메소드 선언만 있다.
- int add(int x, int y) { } : {} 부분이 구현 내용. 추상 메소드 x

- 다형성 실현
- 추상 클래스는 상속을 위한 클래스
- 구현된 메서드
- 하위 클래스가 공통으로 사용할 수 있는 기능 구현
- 경우에 따라서는 하위 클래스가 재정의 (overriding) 할 수 있다.

final 예약어

final 변수는 값이 변경될 수 없는 상수

- public static final double PI = 3.14;

final 변수는 오직 한 번만 값을 할당 가능

final 메소드는 하위 클래스에서 재정의(overriding) 할 수 없음

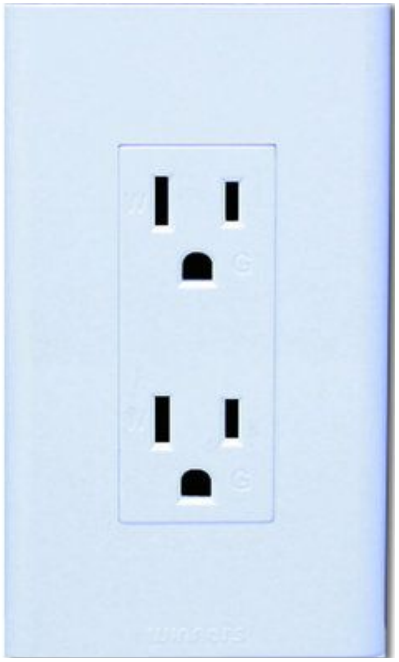
final 클래스는 더 이상 상속되지 않음

같이 해보기!

추상 클래스 – 도형을 넓이를 계산하는 함수선언.

원, 직사각형, 정삼각형 클래스를 만들어서 상속하고 재정의하여 계산 해보자

미리보기 (인터페이스)



인터페이스(interface)

- 클래스에서 메소드를 구현하도록 강제 할 수 있는 기능.

- 모든 메서드가 추상 메서드로 이루어진 클래스로 형식적인 선언만 있고 구현은 없음.

- 인터페이스에 선언된 모든 메소드는 public abstract로 추상 메소드.

- 인터페이스에 선언된 모든 변수는 public static final로 상수

인터페이스 특징

- 상수 – 모든 변수는 상수로 변환 됨. public만 허용, public static final 생략

- 추상 메소드 – public abstract 생략 가능
- default 메소드
  - 인터페이스에 코드가 작성된 메소드
  - 인터페이스를 구현하는 클래스에 자동 상속. 구현 클래스에서 재정의 가능
  - public 접근 지정만 허용. 생략 가능
- private 메소드
  - 인터페이스 내에 메소드 코드가 작성되어야 함
  - 인터페이스 내에 있는 다른 메소드에 의해서만 호출 가능
- static 메소드 – public, private 모두 지정 가능. 생략하면 public

- 인터페이스의 객체 생성 불가
- 인터페이스 타입의 레퍼런스 변수 선언 가능(ex ExInterface Ex;)

인터페이스 상속

- 인터페이스 간에 상속 가능
  - 인터페이스를 상속하여 확장된 인터페이스 작성 가능
  - extends 키워드로 상속 선언

```
interface MobilePhoneInterface extends PhoneInterface {
   void sendSMS();          // 추상 메소드 추가
   void receiveSMS();       // 추상 메소드 추가
}
```

  - 인터페이스는 다중 상속 허용

```
interface MusicPhoneInterface extends PhoneInterface,
MP3Interface {
                .....
}
```

인터페이스 구현

- 인터페이스의 추상 메소드를 모두 구현한 클래스 작성
  - implements 키워드 사용
  - 여러 개의 인터페이스 동시 구현 가능

예시)

```
class SamsungPhone implements PhoneInterface { // 인터페이스 구현
   // PhoneInterface의 모든 메소드 구현
   public void sendCall() { System.out.println("띠리리리링"); }
   public void receiveCall() { System.out.println("전화가 왔습니다."); }

   // 메소드 추가 작성
   public void flash() { System.out.println("전화기에 불이 켜졌습니다."); }
}
```

  - 인터페이스를 구현한 클래스들을 하나의 인터페이스 타입으로 다룰 수 있음
  - 기능(메소드)의 구현을 강제함으로써, 클래스의 설계 또는 표준화를 유도 할 수 있음

같이 해보기!

인터페이스 Car에 필요한 기능들을 넣어서 KiaCar 클래스에 구현 해보자.

인터페이스 Phone에 필요한 기능들을 넣어서 SamsungPhone 클래스에 구현해보자

직접 해보기!

게임 캐릭터의 기능을 구현해보자!

인터페이스 Player - jump(), run(), turn(), showLevelMessage()

클래스 - BeginnerLevel, AdvancedLevel, SuperLevel

  ㄴ>맴버 함수 - 인터페이스 받아 재정의 해보자!