자바1

(클래스와 객체2)

Chapter 08
클래스와 객체2

Chapter08의 학습목표

- 클래스와 객체의 생성자에 대해 학습한다.
- 상속의 정의와 사용법에 대해 학습한다.
- 오버로딩, 오버라이딩의 차이에 대해 알아본다.

제어자

접근 제어자

- 클래스나 멤버의 사용을 제어하기 위해서 사용
- 접근 제어자는 외부에서 접근할 수 있는 정도와 범위를 정해줌

- **public : 접근 제한이 없음**
- **protected : 같은 패키지나 자식 클래스에서 접근 가능**
- **default : 같은 패키지 내에서만 접근 가능**
- **private : 같은 클래스 내에서만 접근 가능**

구분	클래스	패키지	자식 클래스	전체 세계
public	O	O	O	O
protected	O	O	O	X
default	O	O	X	X
private	O	X	X	X

```
class A{
    private int a; //private
    int b; //default
    public int c; //public
}

public class Test2 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        A obj = new A();

        obj.a = 10; //오류!
        obj.b = 20;
        obj.c = 30;
    }
}
```

제어자

접근자와 설정자

- 캡슐화(정보 은닉) : 구현의 세부 사항을 클래스 안에 감추는 것
- 클래스 안의 데이터를 외부에서 마음대로 변경하지 못하게 하는 것
- 외부에서 변수는 감추고, 메소드는 사용할 수 있도록 클래스를 생성

- **getter : 변수 값(필드 값)을 반환하는 접근자**
- **setter : 변수 값(필드 값)을 설정하는 설정자**

```
class Account{
    private String name;
    public int balance;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getBalance() { return balance; }
    public void setBalance(int balance) { this.balance = balance; }
}

public class Test2 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Account obj = new Account();

        obj.setName("Tom");
        obj.setBalance(10000);
        System.out.println("이름은 " + obj.getName() +
                "통장잔고는 " + obj.getBalance() + "입니다.");
    }
}
```

생성자

생성자

- 인스턴스를 생성할 때 자동으로 호출되는 메소드
- 모든 클래스는 기본 생성자를 가지고 있음
- 클래스에 생성자가 없을 경우 컴파일시 컴파일러가 자동으로 생성자를 추가함

기존 방식

```
myphone1.color = "gold";
myphone1.capacity = 32;
myphone1.model = "Galaxy";
myphone1.price = 640,000;
```

기본 생성자

```
class Cellphon{
    String model;
    String color;
    int capacity;
    public Cellphon() {}
}
```

#컴파일시 컴파일러가 자동으로 추가

생성자

생성자

매개변수를 갖지 않는 기본생성자

```
class Cellphone{
    String model = "Galaxy S21";
    String color;
    int capacity;

    public Cellphone() {
        this.color = "black";
        this.capacity = 32;
    }

}
public class Test {
    public static void main(String[] args) {
        Cellphone myphone = new Cellphone();

        System.out.println(myphone.model);
        System.out.println(myphone.color);
        System.out.println(myphone.capacity);
    }
}
```

매개변수를 갖는 생성자

```
class Cellphone{
    String model = "Galaxy S21";
    String color;
    int capacity;
    //매개변수가 있는 생성자
    public Cellphone(String color, int capacity) {
        this.capacity = capacity;
        this.color = color;
    }
}

public class Test {
    public static void main(String[] args) {
        //객체 생성 및 생성자 호출
        Cellphone myphone = new Cellphone("Silver", 64);

        System.out.println(myphone.model);
        System.out.println(myphone.color);
        System.out.println(myphone.capacity);
    }
}
```

오버로딩

오버로딩(Overloading)

- 매개변수의 개수와 타입은 다르지만 이름이 같은 메소드를 여러개 개 정의하는 것
- 메소드의 기능은 같지만 매개변수의 개수와 타입이 다를 때 효율적으로 사용이 가능

오버로딩을 사용하지 않았을 때

```
int sum(int a, int b) {
    return a+b;
}

int sum_2(int a, int b, int c) {
    return a+b+c;
}
```

오버로딩을 사용했을 때

```
int sum(int a, int b) {
    return a+b;
}

int sum(int a, int b, int c) {
    return a+b+c;
}
```

```
public class Test {
    static int sum(int a, int b) {
        return a+b;
    }

    static int sum(int a, int b, int c) {
        return a+b+c;
    }

    static double sum(double a, double b, double c) {
        return a+b+c;
    }

    public static void main(String[] args) {
        System.out.println(sum(3, 2));
        System.out.println(sum(2, 3, 4));
        System.out.println(sum(2.5, 3.5, 4.5));
    }
}
```

오버로딩

생성자 오버로딩(Overloading)

- 생성자 오버로딩을 통하여 다양한 방법으로 객체를 생성
- this : 현재 객체를 나타냄
- this() : 객체의 기본 생성자 호출
- this()를 통한 다른 생성자 호출은 항상 첫 번째 문장이여야 함

```
class Rectangle{
    private int x, y;
    private int width, height;

    Rectangle(){
        this(0, 0, 1, 1);
    }
    Rectangle(int width, int height){
        this(0, 0, width, height)
    }
    Rectangle(int x, int y, int width, int height){
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```

실습1

1. 다음 지시사항을 읽고 이름과 전체 인구수를 저장할 수 있는 Person 클래스를 생성하세요.

지시사항

1. 다음과 같은 방법으로 man과 woman 인스턴스를 생성하세요.

```
Person man = Person(“james”)
Person woman = Person(“emily”)
```

2. man과 woman 인스턴스가 생성되면 다음과 같은 메시지를 출력할 수 있도록 처리하세요.

```
james is born.
Emily is born.
```

3. 다음 코드를 통해서 전체 인구수를 조회할 수 있도록 처리하세요.

```
System.out.println("전체 인구수 : " + Person.get_population());
```

```
james is born.
emily is born.
전체 인구수 : 2명
```

실습2

2. 다음 지시사항을 읽고 가게의 매출을 구할 수 있는 Shop 클래스를 구현하세요.

지시사항

1. Shop 클래스는 전체 매출액을 저장하는 total변수를 가지고 있습니다.
2. 각 메뉴의 가격은 다음과 같습니다. **'떡볶이'**=3000원, **'순대'**=1000원, **'튀김'**=500원, **'김밥'**=2000원
3. 매출이 생기면 다음과 같은 방식으로 메뉴와 판매량을 작성합니다.
4. 1개를 팔 때는 개수를 작성하지 않습니다.
5. 주문은 한번에 2개까지 가능합니다.

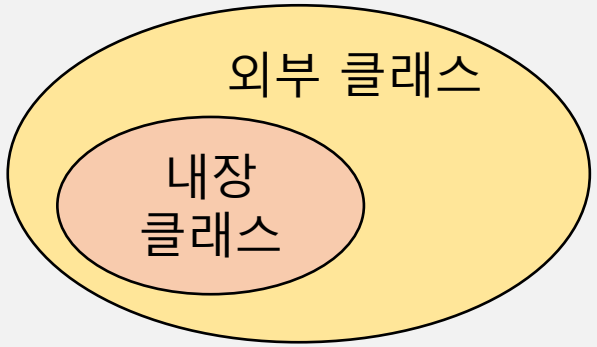
```
Shop.sales('떡볶이') //떡볶이를 1개 판매
Shop.sales('김밥', 2) //김밥을 2개 판매
Shop.sales('튀김', 5) //튀김을 5개 판매
Shop.sales('순대', 3, '김밥', 4) //순대 3개와 김밥을 4개 판매
```

3. 모든 매출을 작성한 뒤 다음과 같은 방식으로 전체 매출액을 확인합니다.

```
System.out.println("총 매출: " + Shop.total + "원" );
```

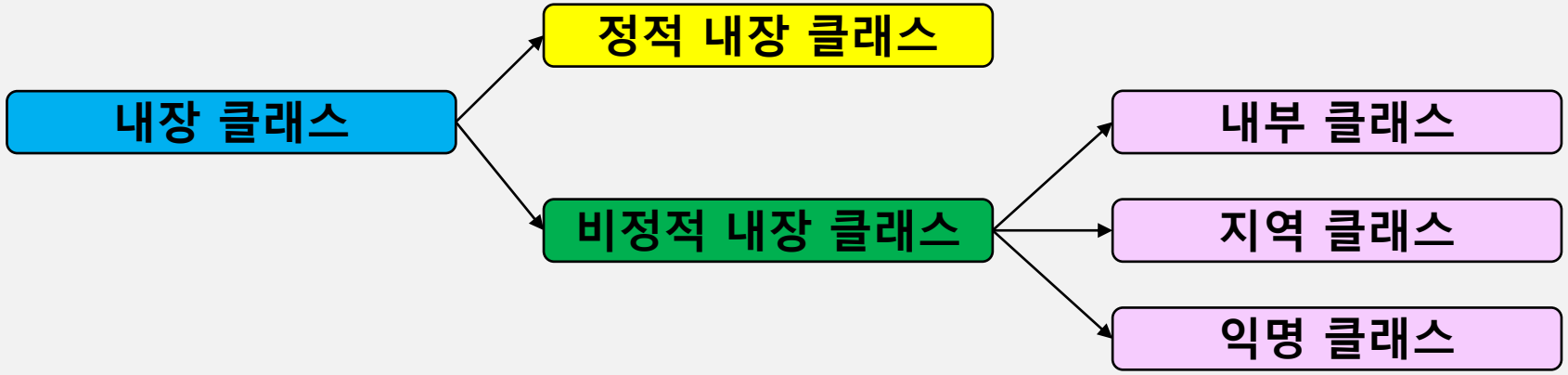
내장 클래스

내장 클래스와 외부 클래스



- 클래스 안에 클래스를 정의 가능
- 외부 클래스(outer class) : 내부에 클래스를 가지고 있는 클래스
- 내장 클래스(nested class) : 클래스 내부에 포함되는 클래스

정적 내장 클래스 : static이 붙어서 내장되는 클래스
비정적 내장 클래스 : static이 붙지 않는 내장 클래스
- **내부 클래스(inner class) : 클래스의 멤버처럼 선언되는 내장 클래스**
- **지역 클래스(local class) : 메소드의 몸체 안에서 선언되는 내장 클래스**
- **무명 클래스(anonymous class) : 수식의 중간에서 선언되고 바로 객체화 되는 클래스**



내장 클래스

내부 클래스

- 클래스 안에 선언된 클래스
- 외부 클래스의 인스턴스 변수와 메소드 전부 사용가능
- private로 선언되어 있어도 접근이 가능

```
class OuterClass{
    private int value = 10;
    class InnerClass{
        public void myMethod() {
            System.out.println("외부 클래스의 private 변수 값: " + value);
        }
    }

    OuterClass() {
        InnerClass obj = new InnerClass();
        obj.myMethod();
    }
}

public class Test2 {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
    }
}
```

지역 클래스

- 메소드 안에 정의되는 클래스
- 접근 제어 지정자를 가질 수 없음
- 지역 클래스는 abstract 또는 final로만 지정 가능

```
class localInner{
    private int data = 30;
    void display() {
        final String msg = "현재의 데이터값은 ";
        class Local{
            void printMsg() {
                System.out.println(msg + data);
            }
        }
        Local obj = new Local();
        obj.printMsg();
    }
}

public class Test2 {
    public static void main(String[] args) {
        localInner inner = new localInner();
        inner.display();
    }
}
```

내장 클래스

익명 클래스

- 이름이 없는 클래스
- 인스턴스를 생성할 때 블록이 있음
- 블록 안에서 클래스 내용을 재정의

```
class Some{
    private int a = 3;
    int getter() {return this.a;}
    void setter(int a) {this.a = a;}
}

public class Test2 {
    public static void main(String[] args) {
        //익명 클래스 정의
        Some s = new Some() {
            private int a = 10;
            int getter() {return this.a;}
            void setter(int a) {this.a = a;}
        };
    }
}
```

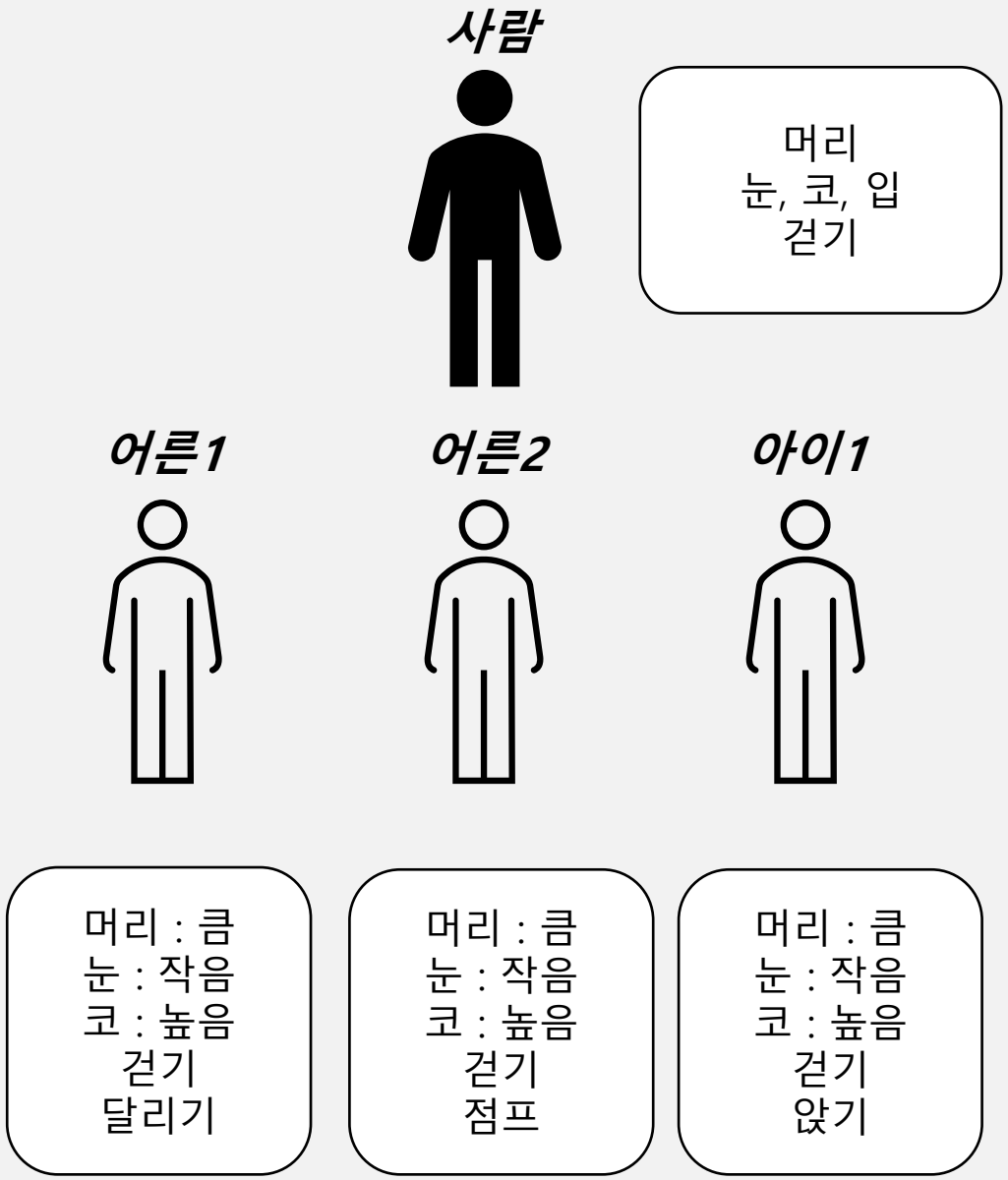
```
class OuterClass{
    void a() {
        System.out.println("method a");
    }
}

public class Test2 {
    public static void main(String[] args) {
        //익명 클래스 정의
        OuterClass outer1 = new OuterClass() {
            void a() {
                System.out.println("재정의한 익명 클래스의 메소드 a");
            }
        };
        outer1.a();

        OuterClass outer2 = new OuterClass();
        outer2.a();
    }
}
```

상속

상속(Inheritance)

- 다른 클래스가 가지고 있는 기능을 그대로 사용
- 부모 클래스와 자식 클래스로 구성
- IS-A 관계가 성립(~는 ~다)

구분	부모 클래스	자식 클래스
의미	상속해 주는 클래스	상속받는 클래스
용어	슈퍼 클래스(super class) 기반 클래스(base class)	서브 클래스 (sub class) 파생 클래스 (derived class)



상속

상속 관계 구현

- 다른 클래스가 가지고 있는 기능을 그대로 사용
- 부모 클래스와 자식 클래스로 구성
- 부모 클래스의 생성자는 상속되지 않음
- 다중 상속은 지원하지 않음

```
class 부모 클래스{
    본문
}

class 자식 클래스 extends 부모 클래스{
    //부모 클래스의 멤버들을 상속받음
    본문
}
```

```java
class Person{
    void breath() {
        System.out.println("숨쉬기");
    }
    void eat() {
        System.out.println("먹기");
    }
}

class Man extends Person{
    void run() {
        System.out.println("뛰기");
    }
}

class Woman extends Person{
    void walk() {
        System.out.println("걷기");
    }
}

public class Test {
    public static void main(String[] args) {
        Man man1 = new Man();
        man1.breath();
        man1.run();

        Woman woman1 = new Woman();
        woman1.eat();
        woman1.walk();
    }
}
```

부모 클래스 Person 정의

부모 클래스 Person을 상속받는
자식 클래스 Man과 Woman 정의

부모 클래스의 메소드 breath, eat을
자식 클래스에서도 사용 가능

상속

상속과 생성자 – super()

- 자식 클래스의 생성자 작업 시 부모 클래스의 생성자도 반드시 호출해야 함
- 부모 클래스의 생성자 호출은 상위 클래스를 의미하는 super()를 통해 이루어짐
- 자식 생성자 내부에 부모 클래스의 생성자를 따로 작성하지 않았다면 자동적으로 컴파일러가 자식 클래스 생성자에 super();을 추가하여 부모 클래스의 생성자를 호출함

```
class Car{
    String color;

    Car(String color) {
        this.color = color;
    }
}

class SportsCar extends Car{
    int speedLimit;

    SportsCar(String color, int speedLimit) {
        this.color = color;
        this.speedLimit = speedLimit;
    }
}
```

```
public class Test2 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        SportsCar sports_car = new SportsCar("red", 330);
        System.out.println(sports_car.color);
        System.out.println(sports_car.speedLimit);
    }
}
```

상속

상속과 생성자 – super()

super클래스의 기본 생성자 사용

```
class Car{
    String color;

    Car(String color) {
        this.color = color;
        System.out.println("매개변수 생성자 호출");
    }
    public Car() {
        System.out.println("기본 생성자 호출");
    }
}

class SportsCar extends Car{
    int speedLimit;

    SportsCar(String color, int speedLimit) {
        super();
        this.color = color;
        this.speedLimit = speedLimit;
    }
}
```

super클래스의 매개변수 생성자 사용

```
class Car{
    String color;

    Car(String color) {
        this.color = color;
        System.out.println("매개변수 생성자 호출");
    }
    public Car() {
        System.out.println("기본 생성자 호출");
    }
}

class SportsCar extends Car{
    int speedLimit;

    SportsCar(String color, int speedLimit) {
        super("blue");
        this.color = super.color;
        this.speedLimit = speedLimit;
    }
}
```

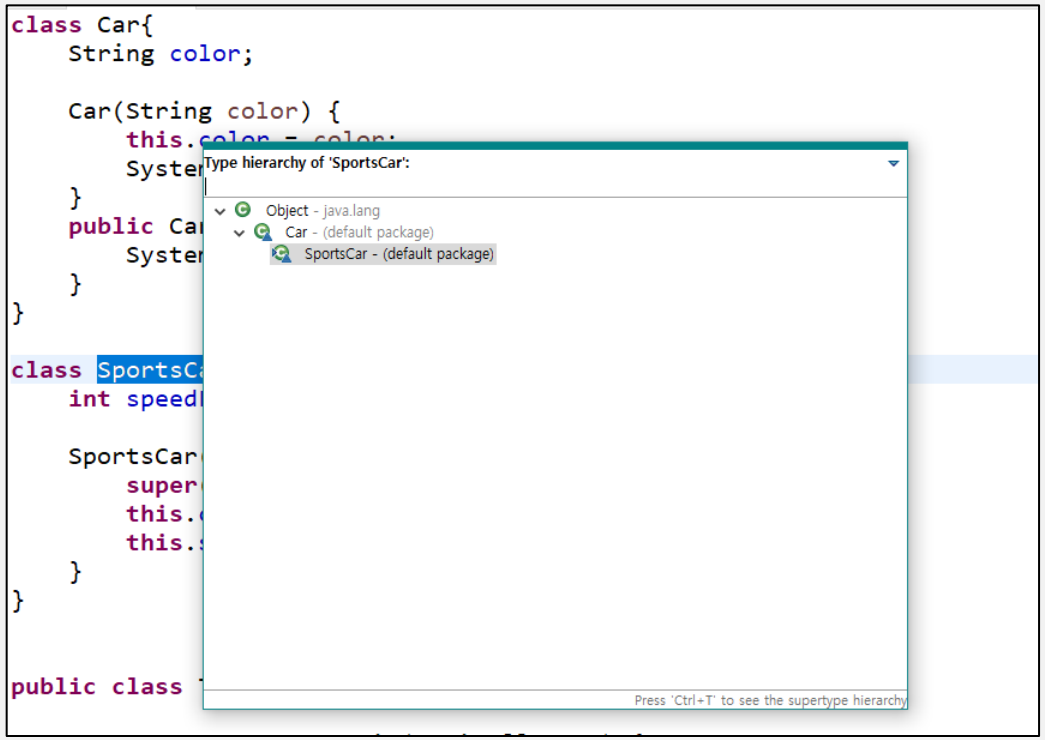
상속

Object 클래스

- 모든 클래스의 조상격인 클래스로, 모든 클래스의 부모 클래스
- 상속을 받지 않는 독자적인 클래스는 자동으로 Object클래스를 상속함 (Ctrl+T로 상속관계 확인 가능)

```
class Tree extends Object{
    //본문
}
```

메소드	설명
Object clone()	객체 자신의 복사본을 생성하여 반환
Boolean equals(Object obj)	obj가 현재 객체와 같은지를 반환
void finalize()	사용되지 않는 객체가 제거되기 직전에 호출
class getClass()	실행 시간에 객체의 클래스 정보를 반환
int hashCode()	객체에 대한 해쉬 코드를 반환
String toString()	객체를 기술하는 문자열을 반환



오버로딩

오버라이딩(Overriding)

- 자식 클래스에서 부모 클래스로부터 물려받은 메소드를 다시 작성하는 것
- 부모 클래스에서 물려받은 메소드의 기능을 변경할 때
- 메소드 이름을 같게 하여 새로운 내용을 작성 (재정의)

```
public class Test {
    public static void main(String[] args) {
        Dog dog1 = new Dog();
        Cat cat1 = new Cat();

        dog1.bark();
        cat1.bark();
    }
}
```

```
class Animal{
    void jump() {
        System.out.println("Jump!");
    }
    void bark() {
        System.out.println("컹컹");
    }
}

class Dog extends Animal{
    @Override
    void bark() {
        System.out.println("왕왕");
    }
}

class Cat extends Animal{
    @Override
    void bark() {
        System.out.println("냐옹");
    }
}
```

부모 클래스 Animal을 상속받아 Animal 클래스의 bark() 메소드를 자식 클래스에서 재정의하여 사용

상속과 오버라이딩 실습

Bank 클래스에 저금을 했을 때, 다양한 이자율을 가지는 은행 클래스 3개에서 (BadBank는 10%, NormalBank는 5%, GoodBank는 3%) 해당 저금 금액의 이자를 반환 받아서 출력하는 프로그램을 구현하세요. (사용자에게 저금 금액을 입력 받습니다.)

Class1 정의
- class이름 : Bank
- 인스턴스 메소드 : getInterestRate()

Class2 정의
- class이름 : BadBank
- 상속받는 class : Bank
- 상속받는 메소드 : getInterestRate()

Class3 정의
- class이름 : NormalBank
- 상속받는 class : Bank
- 상속받는 메소드 : getInterestRate()

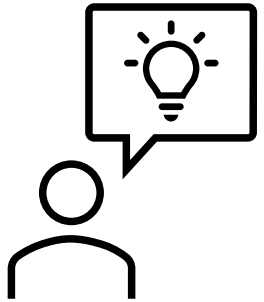
Class2 정의
- class이름 : GoodBank
- 상속받는 class : Bank
- 상속받는 메소드 : getInterestRate()

```
200원의 GoodBank의 이자 : 20.0
200원의 NormalBank의 이자 : 10.0
200원의 BadBank의 이자 : 6.0
```

수고하셨습니다.

Chapter08에서 배운 내용

- 오버로딩
- 오버라이딩
- 생성자
- 상속
- 클래스의 종류



Chapter09에서 배울 내용

- 추상클래스
- 다형성