

JAVA 프로그래밍

추상 클래스

추상 클래스

- 추상적인 개념을 표현하고, 완성되지 않은 메서드를 가지고 있는 클래스

- 추상 메서드는 구현코드 없이 메서드의 선언만 있음

예) **abstract** int add(int x, int y); // 선언만 있는 추상 메서드

int add(int x, int y) { } // { } 부분이 구현 내용임, 추상 메서드 아님

- 메서드가 미완성되어 있기 때문에 추상 클래스로는 객체를 생성할 수 없다.
- 추상 클래스는 주로 상속관계에서 추상적인 개념을 나타내기 위한 목적으로 사용되고, 단일 상속이 가능하다.
- 추상 메서드를 하나라도 가지고 있으면 추상 클래스가 된다.
 - 추상 클래스를 설계할 때,
하위 클래스가 반드시 실행할 내용을 구현하도록 강요할 필요가 있는 메서드는 추상 메서드로 선언하면 된다.

추상 클래스

- 추상 메서드 :

동작 방식을 결정할 수 없을 때, 동작을 확정할 수 없는 경우, 동작 부분을 기술하지 않고 비워두는 메서드 (몸체 [내용/구현부]가 없음)

- 메서드 오버라이딩이 빈번하게 사용될 것이 예측될 경우에도 사용한다.

- 예) 동물 클래스를 구현하려고 할 때 동물에 대한 개념은 알고 있지만 구체적으로 어떤 동작을 하는지 알 수 없기 때문에 구체적인 동작을 구현하기 어렵다. 이런 경우 `move()` 메서드를 정의한다면, 동물이 움직인다는 것은 알고 있지만 구체적으로 날아다니는지 기어다니는 지는 알 수 없다. 이런 경우에 동물 클래스는 추상 클래스로 구현한다.
- 추상 클래스의 예 : 동물, 포유류, 어류, 조류, 도형 등
- 추상 메서드의 대표적인 예 : 스레드의 `run()` 메서드

추상 클래스 구현하기

```
package abstractex;
```

abstract 예약어 추가

```
public abstract class Computer {
```

```
    public abstract void display( );
```

```
    public abstract void typing( );
```

더 이상 오류 없음

```
    ...
```

```
}
```

- 메서드에 구현코드가 없으면 abstract로 선언해야 함
- abstract로 선언된 메서드를 가진 클래스는 abstract로 선언
- 만약 모든 메서드가 구현코드가 있지만,
클래스가 abstract로 선언된 경우 추상 클래스가 됨 → new 할 수 없음

추상 클래스의 사용

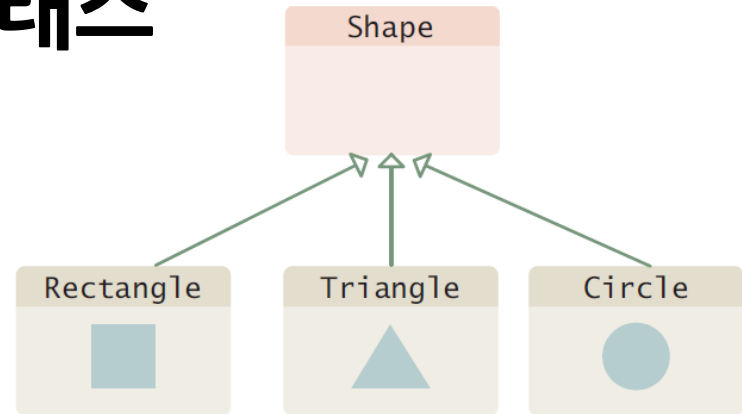
- 추상 클래스는 상속을 위한 클래스
- 추상 메서드 : 하위 클래스가 구현해야 할 각 하위 클래스마다 다르게 구현되어야 하는 기능
- 구현된 메서드 : 하위 클래스가 공통으로 사용할 수 있는 기능 구현

경우에 따라서는 하위 클래스가 재정의(overriding) 할 수 있음

```
public class DeskTop extends Computer {  
    @Override  
    public void display( ) {  
        System.out.println("DeskTop display( )");  
    }  
  
    @Override  
    public void typing( ) {  
        System.out.println("DeskTop typing( )");  
    }  
}
```

추상 메서드의
몸체 코드 작성

추상 클래스



```
abstract class Shape {    // 추상 클래스(추상 클래스는 객체 생성할 수 없다.)
    int x;
    int y;
    public void move(int x, int y) {    // 일반 메서드 정의 가능
        this.x = x;
        this.y = y;
    }
    public abstract void draw();    // 추상 메서드 : 몸체(body)가 없는 메서드
}
```

```
class Rectangle extends Shape {    // 자식 클래스
    int width;
    int height;
    // 부모 클래스의 추상 메서드는 자식 클래스에서 반드시 구현해야 한다.
    // 추상 메서드를 구현하지 않을 시 컴파일 오류 발생한다.
    public void draw() {
        System.out.println("사각형 그리기 메서드");
    }
}

class Circle extends Shape {        // 자식 클래스
    int radius;
    public void draw() {              // 부모 클래스의 추상 메서드 구현
        System.out.println("원 그리기 메서드");
    }
}

// ※ Triangle 클래스를 구현해 보세요.
```

Quiz 추상 클래스

〈FoodTest.java〉

추상 클래스 “요리”를 만들고, 상속받은 자식 클래스(국수, 김치찌개)에서 추상 메서드를 구현해 보세요.

[추상 클래스] Food

[추상 메서드] cook(요리 방법), taste(음식의 맛)

[자식 클래스] Noodle { 추상 메서드 구현 }

[자식 클래스] Kimchi { 추상 메서드 구현 }

[main() 메서드] 자식 클래스의 객체 생성, 생성된 객체로 각각의 메서드 호출

[실행 결과]

<국수를 만드는 방법과 맛>

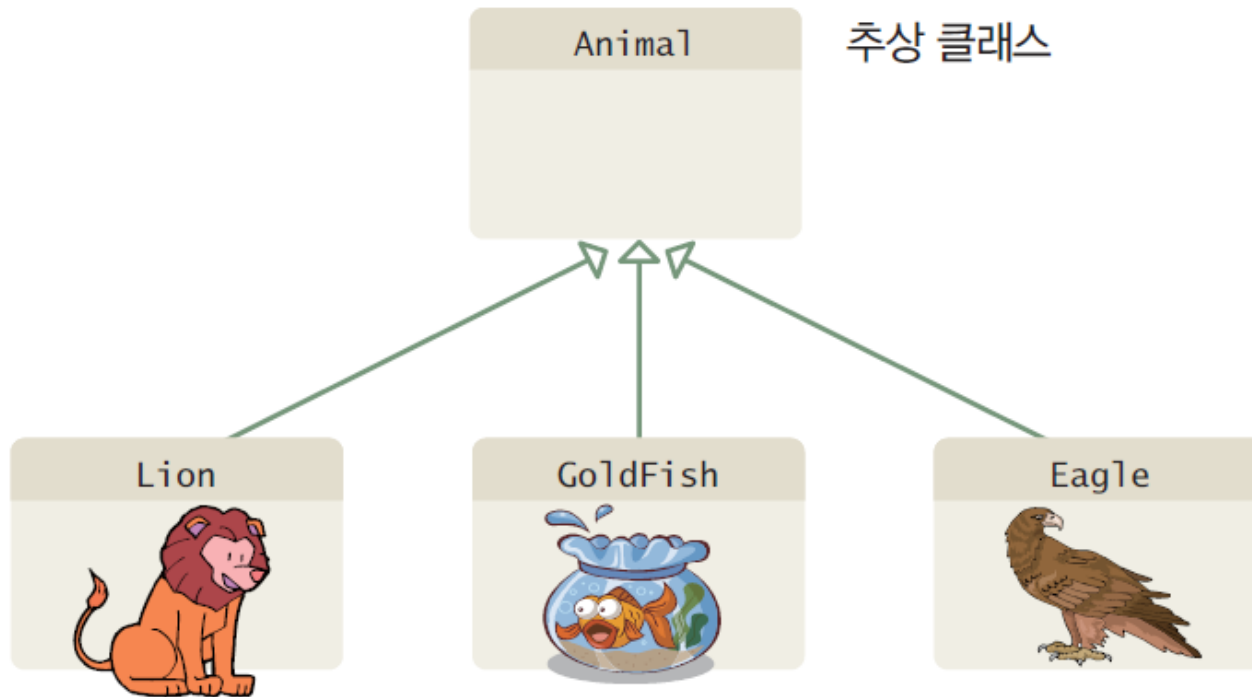
1. 국수를 삶는다.
 2. 양념장과 고명을 만든다.
 3. 국수에 양념장을 넣고 고명으로 장식한다.
- 국수는 쫄깃하고 양념은 간이 잘 맞네요 ^^

<김치찌개 만드는 방법과 맛>

1. 육수를 만든다.
 2. 돼지고기와 김치를 볶는다.
 3. 만든 육수에 돼지고기와 김치를 넣고 끓인다.
- 돼지고기와 김치가 잘 어울려 깊은 맛이 나네요 ^^

Quiz 추상 클래스

〈AnimalTest.java〉 아래의 그림을 추상 클래스로 구현하세요.



Quiz 추상 클래스

〈 AnimalTest.java 〉 추상 클래스 "Animal"을 만들고, 자식 클래스(사자, 금붕어, 독수리)에서 추상 메서드를 구현해 보세요.

[추상 클래스] Animal

[추상 메서드] move(동물의 움직임), shape(동물의 생김새)

[실행 결과]

[자식 클래스] Lion { 추상 메서드 구현; }

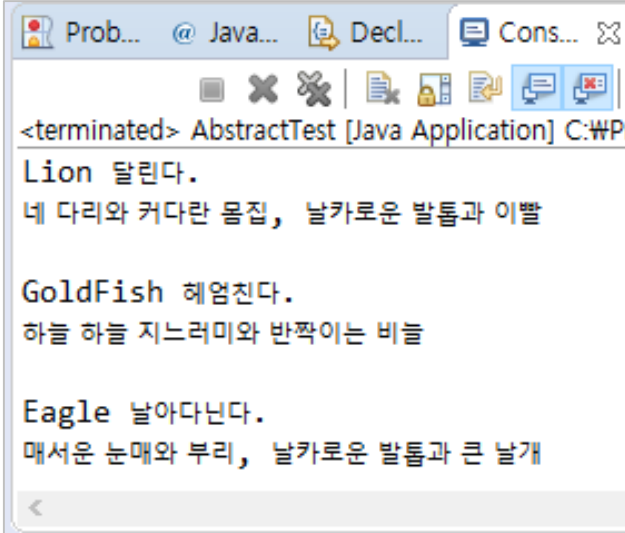
[자식 클래스] GoldFish { 추상 메서드 구현; }

[자식 클래스] Eagle { 추상 메서드 구현; }

[main() 메서드]

자식 클래스의 객체 생성,

생성된 객체로 각각의 메서드 호출



```
<terminated> AbstractTest [Java Application] C:\WP...
Lion 달린다.
네 다리와 커다란 몸집, 날카로운 발톱과 이빨

GoldFish 헤엄친다.
하늘 하늘 지느러미와 반짝이는 비늘

Eagle 날아다닌다.
매서운 눈매와 부리, 날카로운 발톱과 큰 날개
```

추상 클래스와 템플릿 메서드

- 템플릿 메서드 : 추상 메서드나 구현된 메서드를 활용하여

전체 기능의 흐름(시나리오)을 정의하는 메서드


final로 선언하면 하위 클래스에서 재정의 할 수 없음

- 프레임 워크에서 많이 사용되는 설계 패턴
- 추상 클래스로 선언된 상위 클래스에 템플릿 메서드를 활용하여 전체적인 흐름을 정의하고 하위 클래스에서 다르게 구현되어야 하는 부분은 추상 메서드로 선언해 하위 클래스가 구현하도록 함

추상 클래스와 템플릿 메서드

```
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.");  
    }  
  
    final public void run() {  
        startCar();  
        drive();  
        stop();  
        turnOff();  
    }  
}
```

템플릿 메서드



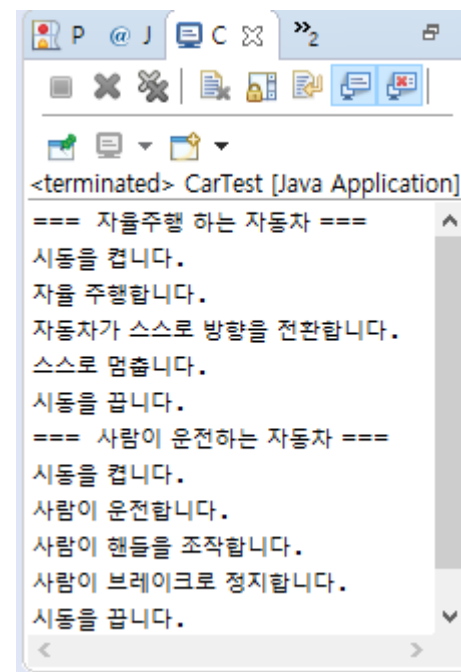
추상 클래스와 템플릿 메서드

```
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("사람이 브레이크로 정지합니다.");  
    }  
}
```

```
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("스스로 멈춥니다.");  
    }  
}
```

추상 클래스와 템플릿 메서드

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



Quiz 추상 클래스

- 모든 차에 와이퍼 기능을 추가하려고 합니다.
 - 추상 메서드를 사용하여 차종이 여러 개 일 때 각 클래스에 와이퍼 기능을 구현
 - wiper() 추상 메서드를 추가한 Car 클래스를 수정

```
<terminated> CarTest (2) [Java Application] C:\Program Files\W...  
=== 자율주행 하는 자동차 ===  
시동을 켭니다.  
자율 주행합니다.  
자동차가 스스로 방향을 전환합니다.  
비나 눈의 양에 따라 빠르기가 자동으로 조절됩니다.  
스스로 멈춥니다.  
시동을 끕니다.  
=== 사람이 운전하는 자동차 ===  
시동을 켭니다.  
사람이 운전합니다.  
사람이 핸들을 조작합니다.  
사람이 빠르기를 조절합니다.  
사람이 브레이크로 정지합니다.  
시동을 끕니다.
```

final 예약어

- final 변수는 값이 변경될 수 없는 상수임
public static final double PI = 3.14;
- final 변수는 오직 한 번만 값을 할당할 수 있음
- final 메서드는 하위 클래스에서 재정의 (overriding) 할 수 없음
- final 클래스는 더 이상 상속되지 않음
예) java의 String 클래스

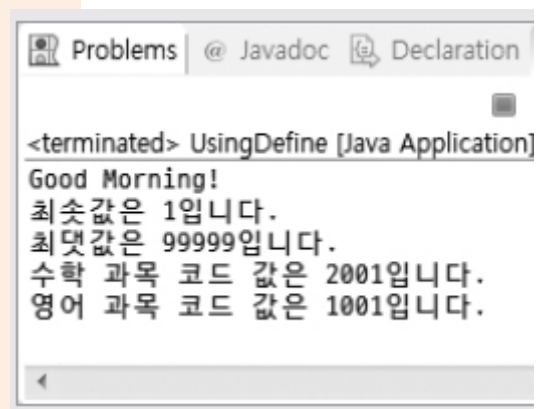
여러 자바 파일에서 공유하는 상수 값 정의하기

- 프로젝트 구현 시, 여러 파일에서 공유해야 하는 상수 값은 하나의 파일에 선언하여 사용하면 편리함

```
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!";  
}
```

```
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("최댓값은 " + Define.MAX + "입니다.");  
        System.out.println("수학 과목 코드 값은 " + Define.MATH + "입니다.");  
        System.out.println("영어 과목 코드 값은 " + Define.ENG + "입니다.");  
    }  
}
```

static으로 선언했으므로 인스턴스를 생성
하지않고 클래스 이름으로 참조 가능



제어자를 조합할 경우 주의사항

- 메서드에 **static**과 **abstract**를 함께 사용할 수 없다.
 - static 메서드는 몸체(구현부)가 있는 메서드에만 사용할 수 있기 때문이다.
- 클래스에 **abstract**와 **final**을 동시에 사용할 수 없다.
 - 클래스에 사용되는 **final**은 클래스를 확장할 수 없다는 의미이고, **abstract**는 상속을 통해서 완성되어야 한다는 의미이므로 서로 '모순'되기 때문이다.
- **abstract** 메서드의 접근 제어자가 **private**일 수 없다.
 - **abstract** 메서드는 자손 클래스에서 구현해주어야 하는데, 접근 제어자가 **private**일 경우 자손 클래스에서 접근할 수 없기 때문이다.
- 메서드에 **private** 접근 제어자와 **final**을 같이 사용할 필요는 없다.
 - 접근 제어자가 **final**인 메서드는 오버라이딩할 수 없기 때문이다.

대상	사용 가능한 제어자
클래스	public, final, abstract
메서드	모든 접근 제어자, final, abstract, static
멤버 변수	모든 접근 제어자, final, static
지역 변수	final

Quiz 템플릿 메서드와 추상 클래스

- 추상 클래스 Car를 상속받는 Avante, Sonata, Grandeur, Genesis 클래스가 있습니다. 각 차량은 다음의 순서로 움직입니다.

```
run() {  
    start();  
    drive();  
    stop();  
    turnOff();  
}
```

- run() 메서드는 템플릿 메서드로 구현하여 다음의 출력 결과와 같이 나오도록 Avante, Sonata, Grandeur, Genesis 클래스를 구현해 보세요.

```
public class CarTest {  
    public static void main(String[] args) {  
        ArrayList<Car> carList = new ArrayList<Car>();  
  
        carList.add(new Sonata());  
        carList.add(new Grandeur());  
        carList.add(new Avante());  
        carList.add(new Genesis());  
  
        for(Car car : carList) {  
            car.run();  
            System.out.println("=====");  
        }  
    }  
}
```

실행결과

```
Sonata 시동을 켭니다.  
Sonata 달립니다.  
Sonata 멈춥니다.  
Sonata 시동을 끕니다.  
=====  
Grandeur 시동을 켭니다.  
Grandeur 달립니다.  
Grandeur 멈춥니다.  
Grandeur 시동을 끕니다.  
=====  
Avante 시동을 켭니다.  
Avante 달립니다.  
Avante 멈춥니다.  
Avante 시동을 끕니다.  
=====  
Genesis 시동을 켭니다.  
Genesis 달립니다.  
Genesis 멈춥니다.  
Genesis 시동을 끕니다.  
=====
```

Quiz 템플릿 메서드와 추상 클래스

- 이전 Quiz에서 구현한 차는 모두 공통으로 washCar() 메서드를 호출할 수 있습니다. 차를 주행한 후 세차를 하도록 메서드를 추가해 프로그램을 구현해 보세요.



```
Problems @ Javadoc Declaration Console Call Hierarchy
<terminated> CarTest (4) [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\
Sonata 시동을 켭니다.
Sonata 달립니다.
Sonata 멈춥니다.
Sonata 시동을 끕니다.
세차를 합니다.
=====
Grandeur 시동을 켭니다.
Grandeur 달립니다.
Grandeur 멈춥니다.
Grandeur 시동을 끕니다.
세차를 합니다.
=====
Avante 시동을 켭니다.
Avante 달립니다.
Avante 멈춥니다.
Avante 시동을 끕니다.
세차를 합니다.
=====
Genesis 시동을 켭니다.
Genesis 달립니다.
Genesis 멈춥니다.
Genesis 시동을 끕니다.
세차를 합니다.
=====
```