



추상클래스

미완성 클래스. 즉, 클래스 내에 메서드의 구현부가 없고 선언만 되어 있을 경우, 이를 추상메서드라 부르고 **abstract** 를 앞에 붙여서 error가 나지 않는다. 또한, 추상 메서드를 하나 이상 가지는 클래스를 추상클래스라고 부른다.

```
abstract class Player {
    int currentPos;           // 현재 Play되고 있는 위치를 저장하기 위한 변수

    Player() {                // 추상클래스도 생성자가 있어야 한다.
        currentPos = 0;
    }

    abstract void play(int pos); // 추상메서드
    abstract void stop();        // 추상메서드

    void play() {
        play(currentPos);       // 추상메서드를 사용할 수 있다.
    }
    ...
}
```

추상클래스는 스스로 인스턴스(객체)화가 될 수 없다. 미완성 클래스이기 때문에.

실습예제1

다음 추상클래스 Calculator를 상속받은 GoodCalc 클래스를 구현하라

```
abstract class Calculator {
    public abstract int add(int a, int b); // 두 정수의 합을 구하여 리턴
    public abstract int subtract(int a, int b); // 두 정수의 차를 구하여 리턴
    public abstract double average(int[] a); // 배열에 저장된 정수의 평균을 구해 실수로 반환
}

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;
    }
    public static void main(String [] args) {
        Calculator c = new GoodCalc();
        System.out.println(c.add(2,3));
        System.out.println(c.add(2,-3));
        System.out.println(c.average(new int [] {2,3,4 }));
    }
}
```

```
5
-1
3.0
```

아래 추상클래스를 실습해 보세요

```
abstract class Figure {
    abstract void area(int a, int b);
}

class Tetragon extends Figure {
    void area(int a, int b) {
        System.out.println("사각형의 넓이 : " + (a * b));
    }
}

class Triangle extends Figure {
    void area(int a, int b) {
        System.out.println("삼각형의 넓이 : " + (a * b / 2));
    }
}

class Simple {
    public static void main(String args[]) {
        Tetragon tetragon = new Tetragon();
        tetragon.area(4, 5);

        Triangle triangle = new Triangle();
        triangle.area(12, 5);
    }
}
```

사각형의 넓이 : 20

삼각형의 넓이 : 30

인터페이스

class 대신에 interface를 사용하는 것 외에 클래스 작성과 동일하다.

실제로 추상클래스보다 인터페이스는 많이 사용되어 지기 때문에 원리를 이해해야 한다.

```
interface 인터페이스이름 {
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수목록);
}
```

- 모든 멤버변수는 `public static final` 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 `public abstract` 이어야 하며, 이를 생략할 수 있다.

```
interface PlayingCard {
    public static final int SPADE = 4;
    final int DIAMOND = 3;      // public static final int DIAMOND = 3;
    static int HEART = 2;       // public static final int HEART = 2;
    int CLOVER = 1;             // public static final int CLOVER = 1;

    public abstract String getCardNumber();
    String getCardKind(); // public abstract String getCardKind();
}
```

실습예제2

```
interface InterTest {
    final int MAX = 100;
    void show(); // 비정상적인 메서드
}

class Child implements InterTest {
    public void show() {
        System.out.println("Interface Test");
    }
}

class ChildTest {
    public static void main(String[] args) {
        Child t = new Child();
        t.show();
    }
}
```

Interface Test

인터페이스 내에 선언된 변수는 무조건 `public static final`로 선언이 되기 되므로, 아래 둘은 완전히 동일한 의미를 갖는다.

```
public class Week
{
    public static final int MON = 1;
    public static final int TUE = 2;
    public static final int WED = 3;
    public static final int THU = 4;
    public static final int FRI = 5;
    public static final int SAT = 6;
    public static final int SUN = 7;
}
```

```
public interface Week
{
    int MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7;
}
```

실습예제3

아래 3가지의 java 파일을 만들어 각 class별 상호관계를 이해하시오.

```
< Stack.java>
interface Stack {
    int length();
    Object pop();
    boolean push(Object ob);
}
```

```
< StringStack.java>
public class StringStack implements Stack {
    private String[] element;
    private int index;
    public StringStack(int capacity)
    {
        element = new String[capacity];
        index = 0;
    }
    @Override
    public int length() {
        // TODO Auto-generated method stub
        return element.length;
    }
    @Override
    public Object pop() {
        if(index == 0) // 스택이 비었음
            return null;
        index--; // 스택 포인터 감소
        return element[index];
    }
    @Override
    public boolean push(Object ob) {
        if(index == element.length)
            return false; // 스택이 다 찼음
        element[index++] = (String)ob; // 요소 스택에 저장 후 스택 포인터 증가
        return true;
    }
}
```

```
<StackManager.java>
public class StackManager {
    public static void main (String[] args) {
        Stack s = new StringStack(10);
        for (int i=0; i<s.length(); i++)
            s.push("문자열"+i);
        for (int i=0;i<s.length(); i++)
            System.out.println(s.pop());
    }
}
```

문자열9

문자열8

문자열7
문자열6
문자열5
문자열4
문자열3
문자열2
문자열1
문자열0

다중상속

인터페이스를 이용하면 다중상속이 가능하다

실습예제4

```
class Tv {
    public void onTv() {
        System.out.println("영상 출력 중");
    }
}

interface Computer {
    public void dataReceive();
}

class ComputerImpl {
    public void dataReceive() {
        System.out.println("영상 데이터 수신 중");
    }
}

class IPTV extends Tv implements Computer {
    ComputerImpl comp = new ComputerImpl();

    public void dataReceive() {
        comp.dataReceive();
    }

    public void powerOn() {
        dataReceive();
        onTv();
    }
}

class Test {
    public static void main(String[] args) {
        IPTV iptv = new IPTV();
        iptv.powerOn();

        Tv tv = iptv;
        Computer comp = iptv;
    }
}
```

또 다른 다중 상속을 아래 코드를 통해 실습해 보세요

```
interface aExample {
    public void sayHello();
}

interface bExample {
    public void sayBye();
}

class JavaInterfaceExample implements aExample, bExample {
    public void sayHello(){
        System.out.println("안녕!");
    }
    public void sayBye(){
        System.out.println("잘가!");
    }
}

public static void main(String args[]) {
    JavaInterfaceExample javaInterfaceExample = new JavaInterfaceExample();

    javaInterfaceExample.sayHello();
    javaInterfaceExample.sayBye();
}
```

```
안녕!
잘가!
```

내부클래스

클래스 안에 클래스가 정의된 형태를 내부클래스라고 부른다.

실습예제5

```
class OuterClass {
    OuterClass() {
        InnerClass nst = new InnerClass();
        nst.simpleMethod();
    }






    static class InnerClass {
        public void simpleMethod() {
            System.out.println("Inner Instance method");
        }
    }
}

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

<pre> OuterClass.InnerClass tt = new OuterClass.InnerClass(); tt.simpleMethod(); } } </pre>	
Inner Instance method	Inner Instance method

다형성 (Polymorphism)

다형성은 개념이다. 아래 예제코드를 이해해 보자. 코드는 아래와 같이 각각 파일로 만들어 보자.

- ▶  Fish.java
- ▶  Human.java
- ▶  Mermaid.java
- ▶  Polimorphismex1.java
- ▶  Shark.java

```

< Fish.java >
public interface Fish {
    public void breatheInWater();
}

```

```

< Human.java >
public class Human {
    public void speaks() {
        System.out.println("Human is speaking");
    }
}

```

```

< Mermaid.java >
public class Mermaid extends Human implements Fish {
    @Override
    public void breatheInWater() {
        System.out.println("Mermaid is breathing in the water");
    }

    @Override
    public void speaks() {
        System.out.println("Mermaid is speaking");
    }
}

```

```

< Shark.java >
public class Shark implements Fish {

    @Override
    public void breatheInWater() {
        System.out.println("Shark is breating in the water");
    }
}

```

```

< Polymorphismex1.java >
public class Polimorphismex1 {
    public static void main(String[] args) {
        Mermaid m1 = new Mermaid();
    }
}

```

<pre> Shark s1 = new Shark(); doWork(m1); doWork(s1); doWork(new Fish() { @Override public void breatheInWater() { System.out.println("Unknown Type of fish is breathing."); } }); static void doWork(Fish f) { f.breatheInWater(); if (f instanceof Human) { ((Human)f).speaks(); } } </pre>	
Mermaid is breathing in the water Mermaid is speaking Shark is breathing in the water Unknown Type of fish is breathing.	

싱글톤 (Singleton)

객체를 하나만 생성시켜 같이 사용하고자 할 경우에 사용함.

<pre> public class SingletonEx1 { public static void main(String[] args) { AAA a1 = AAA.getInstance(); AAA a2 = AAA.getInstance(); System.out.println(a2 == a1); } } class AAA { private static AAA a = new AAA(); private AAA() {} public static AAA getInstance() { return a; } } </pre>	
true	

실습예제6 (선택사항)

목적 : 클래스의 상속에 대한 개념을 실습을 통해서 학습한다.

과제 : 직원 클래스를 조상 클래스로 하여, 정직원 클래스와 시간제 직원 클래스를 만들고, 메소드 오버라이딩을 통해서 급여를 계산하여 출력한다.

과정 :

1. 직원 클래스를 **abstract** 클래스로 선언한다.

2. 정직원 클래스는 직원 클래스를 상속한다. 직원 클래스에서 **abstract** 로 된 메소드의 실제 코드를 구현한다.
3. 시간제 직원 클래스는 직원 클래스를 상속한다. 직원 클래스에서 **abstract** 로 된 메소드의 실제 코드를 구현한다.
4. Department 클래스를 만들고 직원 목록을 배열 객체로 하여 인스턴스 변수를 만든다.
5. Department 클래스에는 showList 라는 메서를 만들어서 모든 직원의 목록을 출력한다.
6. EmployeeManager 클래스에서는 위에서 만든 클래스들을 시험하는 main() 메서드를 작성한다.

```
public class EmployeeManager {
    public static void main(String[] args)
    {
        Department department = new Department();

        department.addEmployee(new Permanent("KIM", 1000));
        department.addEmployee(new Permanent("LEE", 1500));
        department.addEmployee(new Temporary("HAN", 10, 200));
        department.addEmployee(new Temporary("JANG", 12, 300));

        department.showList();
    }
}

abstract class Employee
{
    protected String name;

    Employee(String _name)
    {
        name = _name;
    }

    String getName()
    {
        return name;
    }

    abstract int getPay();
}

class Permanent extends Employee
{
}

class Temporary extends Employee
{
}

class Department
{
}
```

```
name: KIM | salary: 1000
name: LEE | salary: 1500
name: HAN | salary: 2000
name: JANG | salary: 3600
```

수고했습니다. 이제 `interface`를 만들어서 아래 문제를 코딩 해보세요
클래스를 3개 만들어야 합니다. (`interface`, 자식클래스, `main()` 포함하는 클래스)
공통: 1042, 1046, 1604, 1706, 1707, 1708, 1709, 1120, 실습문제 1번, 3번, 5번

