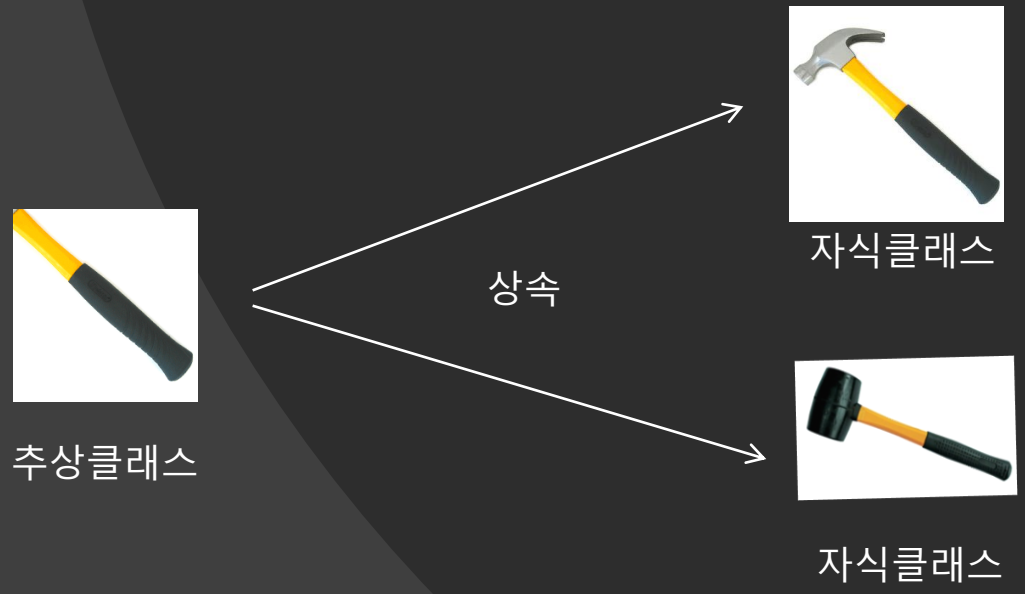
Java 기초

추상클래스, 인터페이스, 어노테이션

추상클래스

- 추상클래스란
  - 미완성 메서드(추상메서드)를 갖고 있는 클래스
  - 추상메서드 : 선언부만 있고 구현부가 없는 메서드
  - 공통된 기능은 일부 구현하고 선언만 된 기능은 상속받는 클래스 내에서 따로 구현이 된다.



추상클래스

- 추상 클래스 작성 방법

추상 클래스일 경우 접근제한자 옆에 abstract를 붙인다. 메서드도 이와 동일

Ex)

```
public abstract class AbstractClass {
    public abstract void abstractMethod();
    public void initMethod(){
        abstractMethod();
    }
}
```

상속

```
public class AbstractChild extends AbstractClass{
    @Override
    public void abstractMethod(){}
}
```

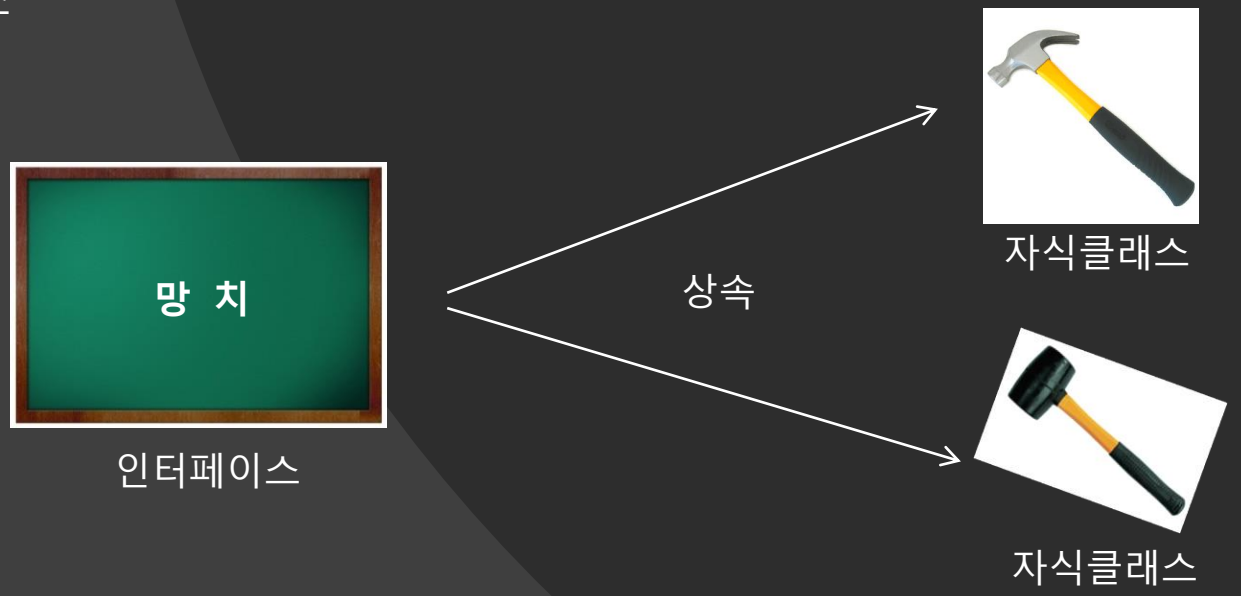
상속을 받는 경우 추상메서드의 리턴값, 메서드이름, 매개변수가 전부 같아야 한다.

추상클래스

- 추상클래스 특징
  - 추상 클래스 내에 일반 메서드가 추상 메서드 호출이 가능하다.
  - 추상 클래스로 인스턴스 생성이 불가능하다.
  - 필요한 기능이지만 자식 클래스에서 저마다 다르게 구현될 경우에 사용한다
  - 추상클래스를 상속받은 자식 클래스는 반드시 추상 메서드의 구현부를 완성 해야한다.
  - 추상클래스를 추상클래스가 상속받았을 경우 굳이 추상 메서드를 구현하지 않아도 된다.

인터페이스

- 인터페이스란
  - 구현된 것이 아무것도 없는 완전 추상 클래스
  - 상수와 추상 메서드만 존재하는 클래스



인터페이스

- 작성 방법

```java
public interface InterfaceClass {
    static final int x = 1;
    int y = 2;
    public void a();
    public void b();
}
```

상속

```java
public class InterfaceChild implements InterfaceClass{
    @Override
    public void a() {}
    @Override
    public void b() {}
}
```

인터페이스

- 작성 방법
  - 인터페이스는 일반 클래스에서는 implements로 상속이 가능하다
  - 인터페이스를 인터페이스가 상속할 때는 extends로 상속한다.
  - 인터페이스는 다중 상속이 가능하다.
  - 인터페이스를 통한 상속과 클래스를 통한 상속을 동시에 할 수도 있다

Ex)

```
public interface InterfaceClass extends InterfaceClass1, InterfaceClass2{
```

```
public class ChildClass1 implements InterfaceClass1, InterfaceClass2{
```

```
public class ChildClass2 extends AbstractClass
    implements InterfaceClass1, InterfaceClass2{
```

인터페이스

- default 메소드
  - 인터페이스는 default로 선언된 메서드를 선언할 수 있다.
  - 해당 메서드는 자식객체에서 오버라이드 하지 않는 이상 해당 디폴트 기능을 수행한다.

```
public interface RemoteControl {
    default void setMute(boolean mute){
        if(mute){
            System.out.println("무음 처리합니다");
        }else{
            System.out.println("무음 해제합니다");
        }
    }
}
```

인터페이스

- 정적 메소드
  - 인터페이스 내에서는 정적 메서드 선언이 가능하다..

```
public interface RemoteControl {
    public static void changeBettery(){
        System.out.println("건전지를 교체합니다.");
    }
```

인터페이스

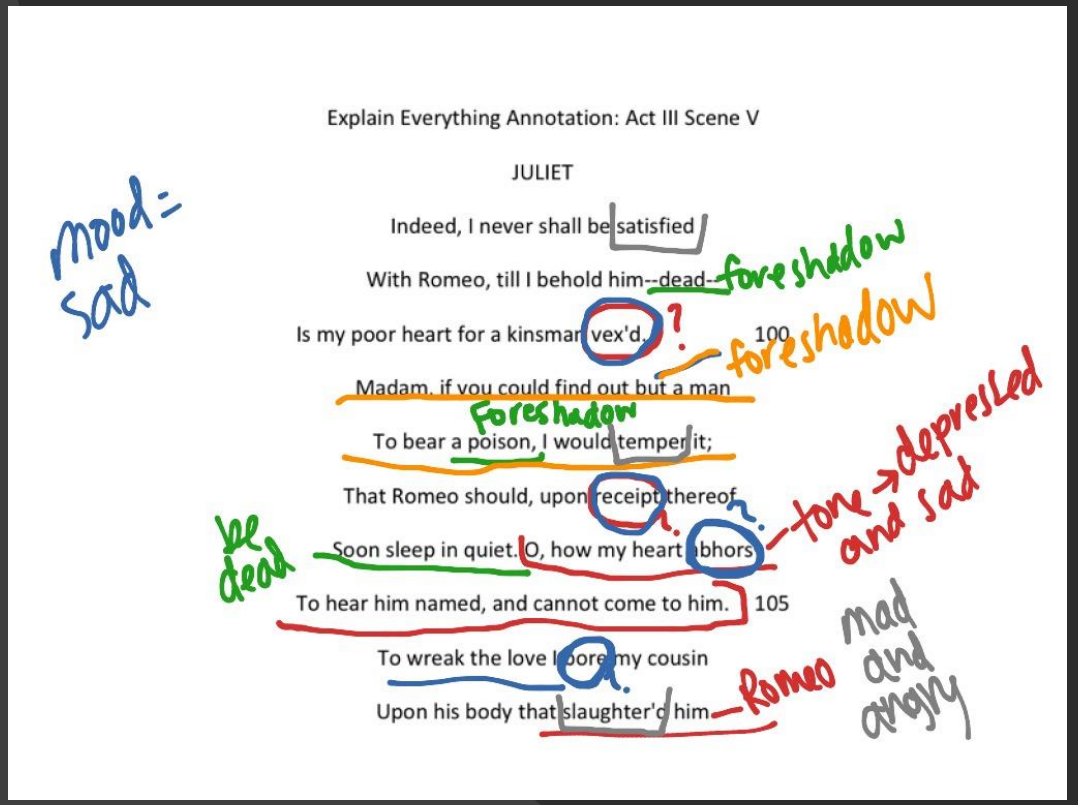
- 특징
  - 인터페이스는 Object와 같은 최상위 클래스가 존재하지 않는다.
  - 인터페이스의 맴버변수는 어떻게 작성을 하던지 전부 상수가 된다.
  - 인터페이스의 맴버변수는 전부 초기화를 해 주어야 한다.
  - 인터페이스의 메서드를 클래스가 상속 받았을 경우 추상메서드를 전부 완성해야 한다.
  - 인터페이스를 추상 메서드나 인터페이스가 상속 받았을 경우 해당 인터페이스의 추상메서드를 완성할 필요가 없다.

인터페이스

- 장점
  - 개발시간을 단축시킬 수 있다.
    - 일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다. 그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.
  - 표준화가 가능하다.
    - 프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.
  - 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.
    - 서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.
  - 독립적인 프로그래밍이 가능하다.
    - 인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다. 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

어노테이션(Annotation)

- 어노테이션이란?
  - 주석의 의미를 담고 있다. 즉 알림 용도
  - 버전이 올라가면서 어노테이션에 기능이 생기기 시작



어노테이션(Annotation)

- 어노테이션의 용도
  - 컴파일러에게 코드 문법 에러를 체크하도록 정보를 제공
  - 소프트웨어 개발 툴이 빌드나 배치 시 코드를 자동으로 생성할 수 있도록 정보를 제공
  - 실행 시(런타임 시) 특정 기능을 실행하도록 정보를 제공

```
@AnnotationName
```

어노테이션(Annotation)

- 어노테이션 생성
  - 기본적으로 어노테이션 생성은 인터페이스 생성과 유사하다.
  - 단 추상메서드 형태가 아닌 엘리먼츠라는 형태를 가진다.

```
public @interface AnnotationName {
    //타입 elementName() [default 값];
}
```

어노테이션(Annotation)

- 어노테이션 생성
  - 엘리먼츠는 복수개를 가질 수도 있으며 아래와 같이 표기가 가능하다.

```
public @interface AnnotationName {
    String elementName1();
    int elementName2() default 5;
}
```

  - 사용방법은 아래와 같다.

```
@AnnotationName(elementName1 = "값1")
int value1;
@AnnotationName(elementName1 = "값2",elementName2=3)
double value2;
```

어노테이션(Annotation)

- 어노테이션 생성
  - 디폴트 값이 없는 엘리먼츠는 반드시 값을 기술해야 한다.
  - 어노테이션은 기본 엘리먼트인 value를 가질 수 있다.

```
public @interface AnnotationName {
    String value();
    int elementName2() default 5;
}
```

```
@AnnotationName("값")
int value;
```

어노테이션(Annotation)

- 어노테이션 적용 대상
  - 어노테이션 적용 대상은 java.lang.annotation.ElementType 열거 상수로 아래와 같이 정의되어 있다.

ElementType 열거 상수	적용 대상
TYPE	클래스, 인터페이스, 열거 타입
ANNOTATION_TYPE	어노테이션
FIELD	필드
CONSTRUCTOR	생성자
METHOD	메서드
LOCAL_VARIABLE	로컬 변수
PACKAGE	패키지

어노테이션(Annotation)

- 어노테이션 적용 대상 예제

```java
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
public @interface AnnotationName {
    String value() default "";
    int elementName2() default 5;
}
```

```java
@AnnotationName
public class AnnotationClass {
    @AnnotationName
    int value;
    @AnnotationName //에러
    public AnnotationClass(){}
    @AnnotationName
    public void method(){}
```

어노테이션(Annotation)

- 어노테이션 유지 정책
  - 사용 용도에 따라 어노테이션을 어느 시점까지 유지할 것인지 지정할 수 있다.
  - 어노테이션 유지 정책은 java.lang.annotation.RetentionPolicy 열거 상수로 다음과 같이 정의된다.

RetentionPolicy 열거 상수	적용 대상
SOURCE	소스상에서만 어노테이션 정보를 유지한다. 소스코드를 분석할 때만 의미가 있으며 바이트 코드 파일에는 정보가 남지 않는다.
CLASS	바이트 코드 파일까지 어노테이션 정보를 유지한다. 하지만 리플렉션을 이용해서 어노테이션 정보를 얻을 수는 없다.
RUNTIME	바이트 코드 파일까지 어노테이션 정보를 유지하면서 리플렉션을 이용해서 런타임 시에 어노테이션 정보를 얻을 수 있다.

어노테이션(Annotation)

- 어노테이션 유지 정책 예제

```java
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface AnnotationName {
    String value() default "";
    int elementName2() default 5;
}
```

어노테이션(Annotation)

- 런타임 시 어노테이션 정보 사용하기
  - 어노테이션은 리플렉션을 이용해서 어노테이션의 적용 여부와 엘리먼트의 값을 읽고 처리할 수 있다.
  - 클래스에 적용된 어노테이션 정보를 얻으려면 java.lang.Class를 이용하면 된다.
  - 하지만 필드, 생성자, 메소드에 적용된 어노테이션 정보를 얻으려면 Class의 아래의 메서드를 통해 java.lang.reflect 패키지의 Field, Constructor, Method 타입의 배열을 얻어야 한다.

리턴타입	메소드명(매개변수)	설명
Field[]	getFields()	필드 정보를 Field배열로 리턴
Constructor[]	getConstructors()	생성자 정보를 Constructor 배열로 리턴
Method[]	getDeclaredMethods()	메소드 정보를 Method 배열로 리턴

어노테이션(Annotation)

- 런타임 시 어노테이션 정보 사용하기
  - Class, Field, Constructor, Method가 가지고 있는 아래 메서드를 호출하여 적용된 어노테이션 정보를 얻을 수 있다.

리턴타입	메소드명(매개변수)
boolean	isAnnotationPresent(Class<? extends Annotation> annotationClass)
	지정한 어노테이션이 적용되었는지 여부, Class에서 호출했을 때 상위 클래스에 적용된 경우에도 true로 리턴된다.
Annotation	getAnnotation(Class<T> annotationClass)
	지정한 어노테이션이 적용되어 있으면 어노테이션을 리턴하고 그렇지 않다면 null을 리턴한다. Class에서 호출했을 때 상위 클래스에 적용된 경우에도 어노테이션을 리턴한다.
Annotation[]	getAnnotations()
	적용된 모든 어노테이션을 리턴한다. Class에서 호출했을 때 상위 클래스에 적용된 어노테이션도 모두 포함된다. 적용된 어노테이션이 없을 경우 길이가 0인 배열을 리턴한다.
Annotation[]	getDeclaredAnnotations()
	직접 적용된 모든 어노테이션을 리턴한다. Class에서 호출했을 때 상위 클래스에 적용된 어노테이션은 포함되지 않는다.

어노테이션(Annotation)

- 런타임 시 어노테이션 정보 사용하기 예제 - 1

```java
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface PrintAnnotation {
    String value() default "-";
    int number() default 15;
}
```

```java
public class Service {
    @PrintAnnotation
    public void method1(){
        System.out.println("실행내용1");
    }
    @PrintAnnotation("*")
    public void method2(){
        System.out.println("실행내용2");
    }
    @PrintAnnotation(value="#", number=20)
    public void method3(){
        System.out.println("실행내용3");
    }
}
```

어노테이션(Annotation)

- 런타임 시 어노테이션 정보 사용하기 예제 - 2

```
public class PrintAnnotationExample {
    public static void main(String[] args) {
        //Service 클래스로부터 메서드 정보를 얻음
        Method[] declaredMethods = Service.class.getDeclaredMethods();
        for(Method method : declaredMethods){
            if(method.isAnnotationPresent(PrintAnnotation.class)){
                //printAnnotation 객체 얻기
                PrintAnnotation printAnnotation =
                        method.getAnnotation(PrintAnnotation.class);
                //메소드 이름 출력
                System.out.println("["+method.getName()+"]");
                //구분선 출력
                for(int i=0;i<printAnnotation.number();i++){
                    System.out.print(printAnnotation.value());
                }
                System.out.println();

                try {
                    method.invoke(new Service());
                } catch (Exception e){}
                System.out.println();
```