03-b. 추상형

Java 객체지향

추상 클래스

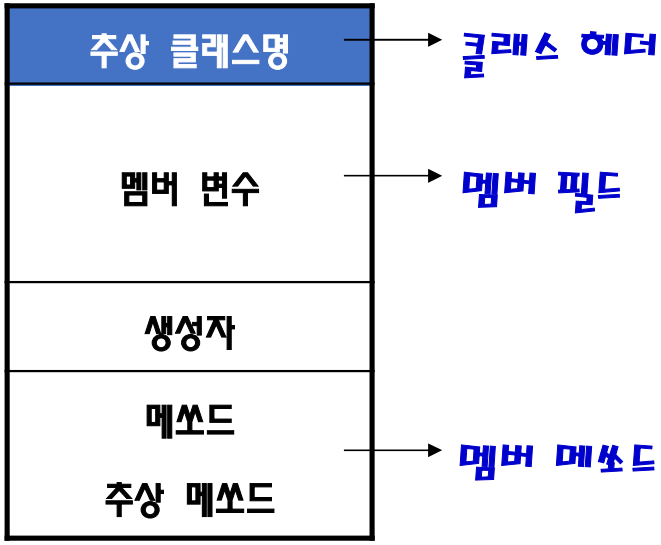
- 추상 클래스
  - **abstract 키워드 사용**
  - **확장 만을 위한 용도로 정의되는 클래스**
  - **하나 이상의 추상 메쏘드를 가짐**
  - **일반 메쏘드를 가질 수 있음**
  - **추상 클래스는 객체화 할 수 없음(컴파일 에러)**

```
abstract class Aclass {
  public abstract void Amethod();
}

…
  Aclass a = new Aclass(); // Error!!
…
```

- 추상 메쏘드
  - **추상 메쏘드는 메쏘드에 대한 구현을 가지지 않음**
  - **추상 클래스의 자식 클래스가 해당 메쏘드를 구현하도록 강요하기 위함**
  - **추상 메쏘드는 추상 클래스에만 존재할 수 있음**

추상 클래스




```
abstract Car
String name
int speed
boolean  isSedan
Car()
setName(String n)
setSpeed (int s)
getSpeed()
abstract boolean isSedan()
```

추상 클래스

- 추상클래스의 상속
  - **추상 클래스의 상속에도 extends 키워드 사용**
  - **추상 클래스를 상속하는 클래스는 반드시 추상 클래스의 추상 메서드를 구현해야 함**
  - **추상 클래스 간의 상속에서는 추상 메서드를 구현하지 않아도 됨**

```
abstract class Aclass {
  public abstract void Amethod(); //추상 메서드
}
abstract class Bclass extends Aclass { //추상 클래스를 상속받은 추상클래스
  public abstract void Bmethod();
}
public class Cclass extends Bclass {
  public void Amethod() { … } //반드시 구현해야 함
  public void Bmethod() { … } //반드시 구현해야 함
}
```

추상 클래스

- 추상클래스의 활용
  - **여러 클래스들이 상당수 공통점을 가지고 있으나 부분적으로 그 처리 방식이 다를 경우 부모 클래스를 추상 클래스로 정의하여 자식 클래스들이 각각 해당 메쏘드를 구현**

```
abstract class Shape {
  public abstract double calculateArea();
}
class Circle extends Shape {
  public double calculateArea() {
        return (double)r*r*Math.PI;
  }
}
class Rectangle extends Shape {
  public double calculateArea() {
        return (double)w*h;
  }
}
```

추상 클래스 – 예제

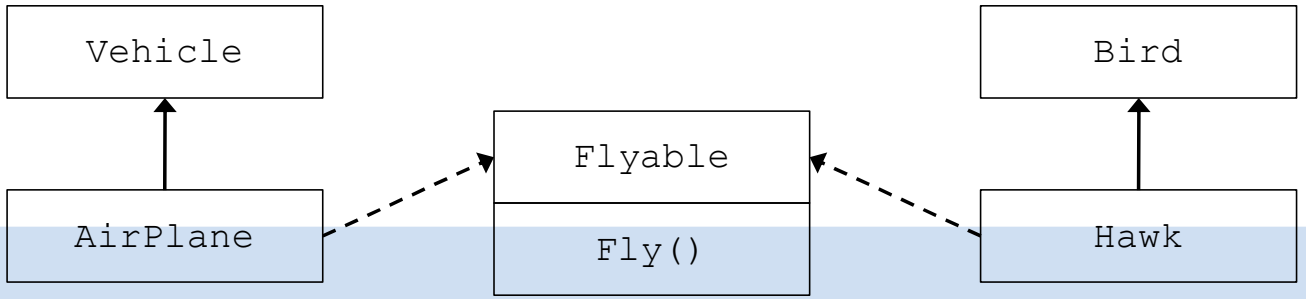
```
public abstract class List {
  protected int size;
  public int length() {
    return size;
  }
  public abstract void
    insertFront(Object item);
}
List myList;     // 문제없음
myList = new List();// 에러
```

```
public class SList extends List {
  protected SListNode head;
  public void insertFront(Object item){
      head = new SListNode(item, head);
      size++;
  }
}

List myList = new SList(); // 문제없음
myList.insertFront(obj);   // 문제없음
```

인터페이스

- 자바 인터페이스(Java Interface; 이하 인터페이스)
  - **개념**
    - **서로 관계가 없는 물체들이 상호 작용을 하기 위해서 사용하는 장치나 시스템**
    - **클래스 구조상의 관계와 상관 없이 클래스들에 의해 구현되어질 수 있는 규약**
  - **목적**
    - **클래스들 사이의 유사한 특성을 부자연스러운 상속 관계를 설정하지 않고 얻어냄**
  - **활용**
    - **하나 또는 그 이상의 클래스들에서 똑같이 구현되어질 법한 메쏘드를 선언하는 경우**
    - **클래스 자체를 드러내지 않고 객체의 프로그래밍 인터페이스를 제공하는 경우**



인터페이스

- 인터페이스의 구현
  - **인터페이스는 다중 상속을 지원하지 않는 자바에서 다중 상속의 장점을 활용하기 위해 도입**
  - **한 클래스는 하나의 부모클래스를 가지며 하나 이상의 인터페이스를 구현할 수 있음**
  - **인터페이스 사이에서는 다중 상속이 가능**
  - **인터페이스는 interface 키워드로 선언하고 implements 키워드로 사용**
  - **인터페이스를 구현한 클래스에서 추상 메쏘드를 반드시 정의해야 함**

```
public interface Drivable extends Steerable, Acceleratable {
}


public class Sedan extends Car implements Drivable, Breakable {
}
```

인터페이스

```
interface Drawable {
  public void draw(); //abstract 키워드로 지정하지 않더라도 자동으로 추상화
}
abstract class Shape {
   public abstract double calculateArea();
}
class Circle extends Shape implements Drawable {
  public double calculateArea() {
        return (double)r*r*Math.PI;
  }
  public void draw() {
        System.out.println("이 객체는 원 입니다.");
  }
}
```

인터페이스

- 인터페이스를 구현한 객체에 대한 instanceof 연산자의 동작

```
> Circle c = new Circle();
> c instanceof Circle
true
> c instanceof Drawable
true
> c instanceof Rectangle
false
> c instanceof Shape
true
```

인터페이스

- 필드 선언
- 컴파일 시 필드 선언부에 다음의 세 가지제어자가 자동으로 추가됩니다.

- public → 누구나 접근하여 사용할 수 있음
- static → 인스턴스 생성 없이 사용할 수 있음
- final → 초기화된 값을 변경할 수 없음.

```
public interface Messenger {
  int MIN_SIZE = 1;
  int MAX_SIZE = 104857600;
}
```

```
public interface Messenger {
  public static final int MIN_SIZE = 1;
  public static final int MAX_SIZE = 104857600;
}
```

인터페이스

- 메서드 선언
- 인터페이스에 선언하는 메서드는 실제 내용을 구현할 목적이 아니라 인터페이스 통일을 목적으로 선언하는 것이므로 본문을 구현하지 않습니다.
- 컴파일 시 public abstract 제어자가 자동으로 추가됩니다.

```
public String getMessage()
public void setMessage(String msg);
```

```
public abstract String getMessage()
public abstract void setMessage(String msg);
```

인터페이스

- Default 메서드 선언
- 자바 8부터 지원하며 인터페이스를 사용하는 모든 클래스에서 공통으로 갖는 기능을 구현할 목적으로 사용 합니다.

【default 메서드 선언】

```
public default 리턴 타입 메서드명(매개변수) {
  실행문;
}
```

```
public defalut void setLogin(boolean login) {
  if(login) {
    System.out.println("로그인 처리합니다.");
  } else {
    System.out.println("로그아웃 처리합니다");
  }
}
```

인터페이스

- static 메서드 선언
- 자바 8부터 지원하며 프로그램 시작 시 메모리에 사용 준비가 완료되므로 인스턴스 생성과 상관없이 바로 사용할 수 있습니다.
- static 메서드 선언 시 public 접근 제한자를 생략하면 컴파일 시 자동으로 추가됩니다.
- static으로 선언한 메서드는 "인터페이스명.메서드명( )"으로 호출합니다.

```
public static void getConnection() {
  System.out.println("network에 연결합니다.");
}
```

```
Messenger.getConnection();
```

인터페이스

- private 메서드 선언
- 자바 9에서 추가되었으며 동일한 인터페이스에 선언된 default 메서드에서만 사용할 목적으로 본문을 구현하는 메서드입니다

```
private void log() {
  System.out.println("start job!");
}
```

```
default void setLogin(boolean login) {
  log();
}
```

```
private static void log() {
  System.out.println("start job!");
}
```

```
static void getConnection() {
  log();
  System.out.println("network에 연결합니다.");
}
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare