Abstract Class

경복대학교
소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Abstract

- 추상화란?
  - 물체의 필요 부분만 부각시키는 것(Attribute+Behavior)
  - “추상”의 사전적 의미로 "여러가지 사물이나 개념에 공통되는 특성이나 속성 따위를 추출하여 파악하는 작용" 이라는 의미
  - Class간의 공통점을 찾아내서 공통의 부모를 설계하는 작업
- 구체화
  - 상속을 통해 Class를 설계, 확장하는 작업
- 여러 Object에 공통되는 사항을 추출하여 부모 Class에 선언 및 구현을 하고 자식 Class에는 특징을 구현하여 기능을 확장
- 이 내용만 보면 Inheritance의 개념과 동일하게 보일 것
- Abstract Class는 단순 기능 확장에 개발의 표준화 정도를 올려줌

Abstract

- 정의
  - 실체가 없는 추상적인(구현되지 않은) 의미
  - 내용은 없고 껍데기만 있는 것을 의미
  - 기존의 Class가 “설계도”라면, Abstract Class는 “미완성 설계도”라고 할 수 있음

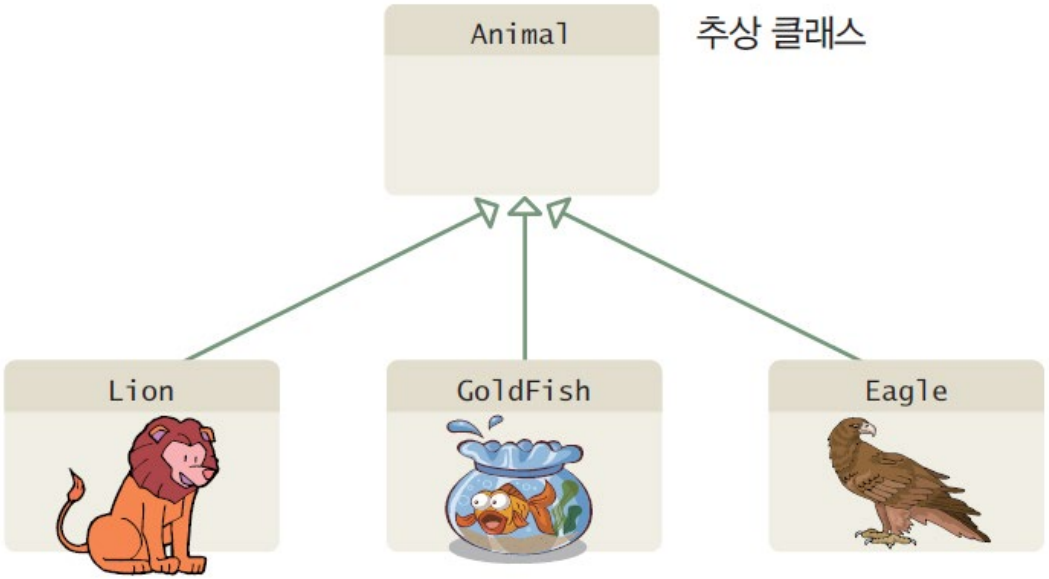
- 용도
  - Class – Abstract Class
  - Member Method – Abstract Method

Abstract

- Abstract Method(미완성 Method)
  - 구현되지 않은 Method, 즉 Body가 없는 Method
  - 선언부만 있고, 구현부가 없는 Method
  - Method 앞에 abstract라는 Keyword 붙음
  - 예) abstract public void sum();
- Concrete Method(구현 Method)
  - 구현된 Method, 즉 Body가 있는 Method
  - 예) public void run() { … }
- Abstract Class(미완성 설계도)
  - Abstract Method를 포함하고 있는 Class
  - class앞에 abstract를 붙여 줌

Abstract Class

- 개념



- Abstract Method를 하나라도 가진 Class로 Object를 생성할 수 없음
  - Class 앞에 반드시 abstract라고 선언해야 함
- Abstract Method가 하나도 없지만 Class 앞에 abstract로 선언한 경우

Abstract Class

- abstract 수정자와 Abstract Class
  - Abstract Class는 일반적인 Attribute와 Behavior를 추출하여 구현을 하지 않은 Method들로 정의된 Class로서 Instance를 만들 수 없는 Class이고 다음의 형식과 같이 abstract 수정자를 class 앞에 사용

```
abstract class 클래스 이름 {
    // 변수 선언, 생성자, 메소드
   abstract [접근 제한자]반환 형 메소드 이름([매개 변수....]);
}
```

  - Abstract Class는 일반 Class와 동일하게 변수와 생성자, 여러 개의 Method를 포함할 수 있음
  - class 앞부분에 abstract를 나타낸 Class는 Abstract Class로 Super Class를 의미하며 Instance는 만들 수 없고 반드시 Sub Class로 상속이 이루어져야 함

Abstract Class

- abstract 수정자와 Abstract Class
  - Method의 앞 부분에 abstract를 나타내면 Abstract Method가 되며 몸체를 가질 수 없고 Sub Class에서 Overriding하여 해당 Method를 작성해야 함
  - Abstract Method를 갖는 Class는 자동으로 Abstract Class가 되므로 class 앞에 abstract를 생략해도 됨

Abstract Class

- 하위 Class에서 구현될 추상적인 기능만을 Abstract Method로 선언
- Abstract Method는 기능이 무엇(What)인지 만을 선언하고 구현 부분이 없는 Method
- Abstract Method는 하위 Class에서 Overriding되어 구현
- Abstract Class는 Abstract Method 외에 일반적인 Attribute와 Method를 가질 수 있음

```
abstract class 클래스 이름 {
    ……    // 일반 속성과 메소드 기술

    abstract void 추상 메소드 이름();  // Abstract Method 선언
        ……..
}
```

Abstract Class

- Abstract Class를 상속받은 Class는 반드시 Abstract Method를 Overriding(재정의) 해야 함
- Abstract Class는 미완성 Class이기 때문에, Instance(객체)를 생성할 수 없음
  - Abstract Class는 반드시 상속 관계를 통해서만 사용할 수 있음

Abstract Class

- 정의
  - Class 선언 시 abstract 지정자로 선언된 Class
  - Object 생성이 불가능
  - Member : Member 변수, 생성자, Concrete Method, Abstract Method 등 모두 가능함
  - Abstract Method가 없어도 Abstract Class로 선언 가능
  - Abstract Method가 있으면 반드시 Abstract Class로 선언해야 함
  - 자식(하위) Class를 이용하여 Member를 재사용
  - Type으로 사용 가능
  - 자식(하위) Class에 강제성과 통일성을 줄 수 있음

Abstract Class

```
// 추상 메소드를 가진 추상 클래스
public abstract class DObject {
        public DObject next;
        public DObject() {
            next = null;
        }
        public abstract void draw();
}
```

추상 메소드

```
// 추상 메소드 없는 추상 클래스
public abstract class Person {              // 추상 클래스 선언
      public String name;
      public Person(String name) {
            this.name = name;
      }
      public String getName() {
            return name;
      }
}
```

Abstract Class

- 용도
  - 실체 Class들의 공통된 Field와 Method의 이름을 통일할 목적
    - 여러 사람이 실체 Class를 설계하는 경우, 실체 Class마다 Field와 Method가 제각기 다른 이름을 가질 수 있음
    - 이를 방지하기 위해 Abstract Class에서 Field와 Method 이름을 정하고 이를 상속 받음으로서 공통된 Field와 Method를 통일할 수 있음
  - 실체 Class를 작성할 때 시간을 절약
    - 공통되는 Field와 Method는 Abstract Class에 모두 선언해 두고, 실체 Class마다 다른 점만 실체 Class에 선언한다면 실체 Class를 작성하는데 시간을 절약할 수 있음

Abstract Class

- 용도
  - 자식(하위) Class에서 반드시 구현해야 하는Method를 부모(상위) Class에서 정의할 때 부모(상위) Class에서 Method 구현하지 않고 실제 구현은 자식(하위) Class에서 하도록 함
  - 이때 부모(상위) Class에서 Method를 Abstract Method로 선언하여 강제성과 통일성을 줄 수 있음

Abstract Class

- Abstract Method를 선언하는 이유
  - 설계자가 특정 Method를 각 Class 별로 재 구현을 원하지만 부모 Class에서 일반 Method로 구현하면 자식 Class에서 구현을 하지 않는 경우가 발생할 수 있음
  - 이런 Method를 Abstract Method로 선언하면 자식 Class는 재 구현을 강요 받음
  - 설계와 구현 분리
    - Sub Class마다 목적에 맞게 Abstract Method를 다르게 구현하여 다형성 실현
    - Super Class에서는 개념 정의
      - Sub Class마다 다른 구현이 필요한 Method는 Abstract Method로 선언
    - 각 Sub Class에서 구체적 행위 구현
  - 계층적 상속 관계를 갖는 Class 구조를 만들 때 사용

Abstract Class

- Abstract Class의 장점
  - 부모 Class에서 공통 부분을 구현과 설계가 완료되면 자식 Class에서 상속받아 기능을 확장 시 편리함
  - 자식 Class에서 Abstract Method를 반드시 구현하도록 강요
  - 이는 Program의 표준화 정도를 높임
  - 공통 사항이 한곳에서 관리되어 개발 및 유지 보수에 용이함

Abstract Class

- Abstract Class는 Instance를 생성할 수 없음

```
public abstract class DObject { // 추상 클래스 선언
    public DObject next;
    public DObject() { next = null; }
    public abstract void draw();  // 추상 메소드 선언
}
public class AbstractError {
    public static void main(String [] args) {
        DObject obj;          // 오류 아님 (추상 클래스의 레퍼런스 변수 선언은 가능)
        obj = new DObject();
                    // 컴파일 오류, 추상 클래스 DObject의 인스턴스를 생성할 수 없다
        obj.draw();  // 컴파일 오류
    }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type DObject

        at chap5.AbstractError.main(AbstractError.java:11)
```

Abstract Class

```
class DObject {
    public DObject next;

    public DObject() { next = null;}
     public void draw() {
        System.out.println("DObject draw");
    }
}
```

추상 클래스로 수정

```
abstract class DObject {
    public DObject next;

    public DObject() { next = null;}
    abstract public void draw();
}
```

```
class Line extends DObject {
    public void draw() {
        System.out.println("Line");
    }
}
```

추상 클래스를 상속받아 추상 메소드 draw()를 구현한 클래스

```
class Rect extends DObject {
    public void draw() {
        System.out.println("Rect");
    }
}
```

```
class Circle extends DObject {
    public void draw() {
        System.out.println("Circle");
    }
}
```

Abstract Class

- Abstract Class 구현 및 활용 예

```
public abstract class DObject {
   public DObject next;

   public DObject() { next = null;}
   public abstract void draw();
}
```

```
class Line extends DObject {
    public void draw() {
        System.out.println("Line");
    }
}
```

```
class Rect extends DObject {
    public void draw() {
        System.out.println("Rect");
    }
}
```

```
class Circle extends DObject {
    public void draw() {
        System.out.println("Circle");
    }
}
```

추상 클래스를 상속받아 추상 메소드 draw()를 구현한 클래스

Abstract Method

- Sub Class에서 반드시 Overriding해야 만 사용할 수 있는 Method를 의미
- JAVA에서 Abstract Method를 선언하여 사용하는 목적은 Abstract Method가 포함된 Class를 상속받는 Sub Class가 반드시 Abstract Method를 구현하도록 하기 위함
- 예) Module처럼 중복되는 부분이나 공통적인 부분은 미리 다 만들어진 것을 사용하고, 이를 받아 사용하는 쪽에서는 자신에게 필요한 부분만을 Overriding하여 사용함으로써 생산성이 향상되고 배포 등이 쉬워지기 때문임
- 이러한 Abstract Method는 선언부만이 존재하며, 구현부는 작성되어 있지 않음
- 바로 이 작성되어 있지 않은 구현부를 Sub Class에서 Overriding하여 사용하는 것임

Abstract Method

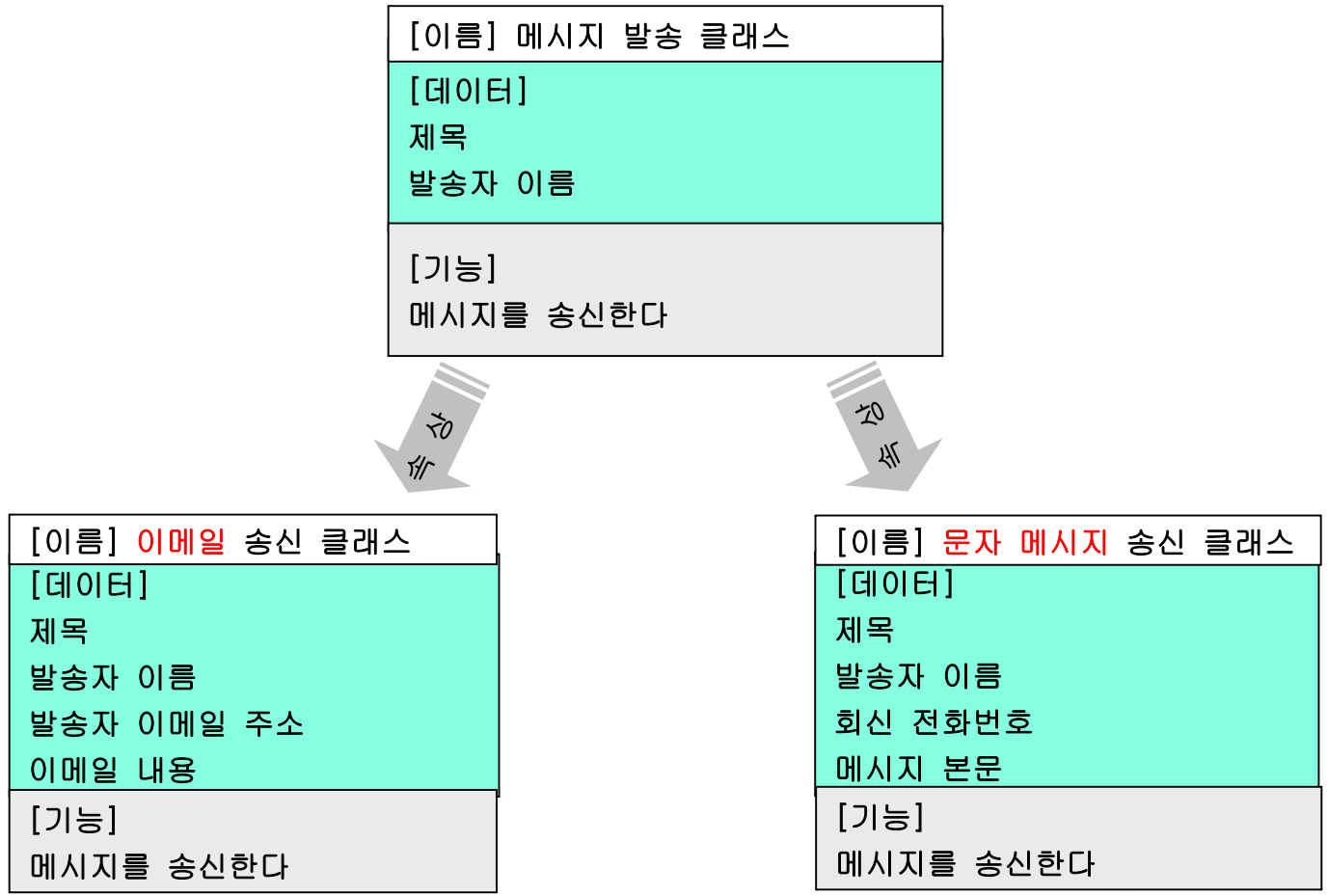
- 선언되어 있으나 Code가 구현되어 있지 않은 Method
  - abstract Keyword로 선언
  - Method Type, 이름, 매개 변수 List만 선언
  - 예)

```
public abstract int getValue();

public abstract void setName(String s);

public abstract Sting fail() { return “Good Bye”; }   (x)
```

- Abstract Method는 Sub Class에서 Overriding하여 구현
  - Abstract Method를 사용하기 위해서는 반드시 해당 Method를 Overriding해야만 함

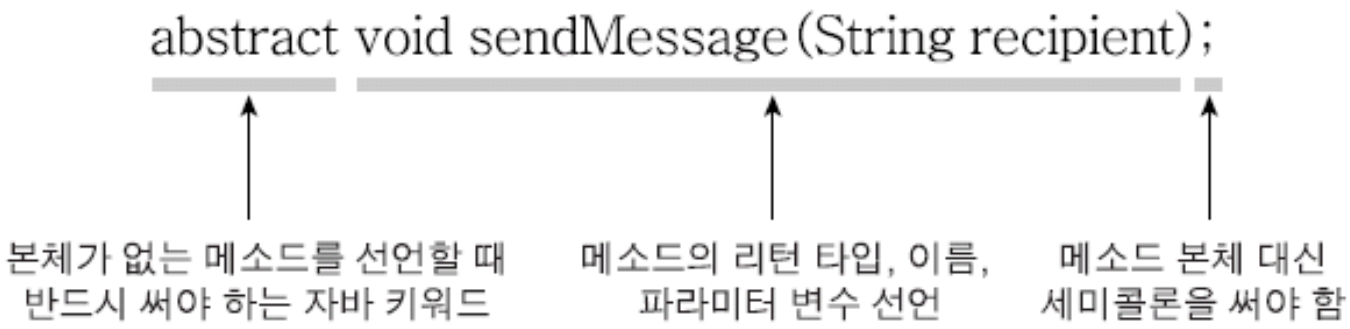
Abstract Method

- Abstract Method가 필요한 경우



Abstract Method

- Abstract Method 선언 방법
  - abstract Keyword로 선언
  - Method 타입, 이름, 매개 변수 리스트만 선언



Abstract Class의 상속

- Abstract Class의 단순 상속
  - Abstract Class를 상속받아, Abstract Method를 구현하지 않으면 Sub Class도 Abstract Class가 됨
  - Sub Class도 abstract로 선언해야 함

```
public abstract class DObject { // 추상 클래스
   public DObject next;
   public DObject() { next = null; }
   public abstract void draw();   // 추상 메소드
}

abstract class Line extends DObject {
                        // draw()를 구현하지 않았기 때문에 추상 클래스
    public String toString()  { return "Line";}
}
```

Abstract Class의 상속

- Abstract Class 구현 상속
  - Sub Class에서 Super Class의 Abstract Method 구현 (Overriding)
  - Sub Class는 Abstract Class 아님

Abstract Class의 상속

```
abstract class Shape {
    int x, y;
    public void move(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public abstract void draw();
};

public class Rectangle extends Shape {
    int width, height;
    public void draw() {      // 추상 메소드 구현
        System.out.println("사각형 그리기 메소드");
    }
};

class Circle extends Shape {
    int radius;
    public void draw() {
        System.out.println("원 그리기 메소드");
    }
};
```

추상 클래스 Shape를 선언한다. 추상 클래스로는 객체를 생성할 수 없다.

추상 클래스라고 하더라도 추상 메소드가 아닌 보통의 메소드도 가질 수 있음을 유의하라.

추상 메소드를 선언한다. 추상 메소드를 하나라도 가지면 추상 클래스가 된다. 추상 메소드를 가지고 있는데도 abstract를 class 앞에 붙이지 않으면 컴파일 오류가 발생한다.

서브 클래스 Rectangle에서 수퍼 클래스의 추상 메소드 draw()가 실제 메소드로 구현한다. 서브 클래스에서 추상 메소드를 구현하지 않으면 컴파일 오류가 발생한다.

추상 메소드 draw()가 실제 메소드로 구현한다.