추상화
(클래스와 메소드)

추상화

- **추상화(abstract)**

구체적인 사실들을 일반화시켜 기술하는 개념으로써 필요한 공통점을 추출하고
불필요한 공통점을 제거하는 과정

1. 복잡한 시스템을 단순화하고 모델링이 가능

추상 클래스 - 1

- **추상 클래스(abstract class)**

  미완성된 클래스로써, 구현부 {} 가 없는(미완성인) 메소드를 가지고 있음

- **추상 클래스와 메소드 예시**

```
abstract class Play {
    abstract void attack();
    abstract void stop();
}
```

추상 클래스 - 2

- **추상 클래스(abstract class)**

  미완성된 클래스

- **추상 메소드(abstract method)**

  구현부 {} 가 없는(미완성인) 메소드

- **추상 클래스와 메소드 예시**

```
abstract class Play {
    abstract void attack();
    abstract void stop();
}
```

추상 클래스 - 3

- **추상 클래스를 사용하는 이유**

  똑같은 부모 클래스(추상 클래스)를 상속 받지만, 자식 클래스들의 메소드가
  모두 다른 기능을 수행할 것으로 예상되는 경우

- **다음 슬라이드 예시**

  1. 생물(Organism) 이라는 추상 클래스를 만들고 식물(Plant)과 동물(Animal) 클래스에서 상속 받으려고 함.
  2. 먹고(eat), 숨쉬는(breathe)것은 식물과 동물 모두 똑같으나 서로 다름
     → 동물은 먹이를 먹고, 호흡 기관을 통해 호흡함
     → 식물은 광합성을 통해 영양분을 보충하고, 교환 기능을 통해 호흡함

추상 클래스 - 4

- 특징 - 1

추상 클래스는 객체 생성 불가
→ 상속을 통해 추상 메소드의 구현부를 완성해야 객체 생성 가능

- 추상 클래스와 메소드 예시

```
// 생물 추상화 클래스
abstract class Organism  {
    public abstract void eat();
    public abstract void breathe();
}
```

추상 클래스 - 5

- 특징 - 2

추상 클래스 안에는 추상 메소드뿐만 아니라 멤버 변수와 구현이 된 메소드들도 사용이 가능함

- 추상 클래스와 메소드 예시

```
// 생물 추상화 클래스
abstract class Organism  {
    public String name;

    public abstract void eat();
    public abstract void breathe();

    public void organism() {
        System.out.println("생물체입니다.");
    }
}
```

추상 클래스 - 6

- 설명

아래와 같이 추상화 클래스를 상속 받고, 메소드의 구현부 {} 까지 다 만들어주면
객체 생성 및 사용 가능

```
// 생물 추상화 클래스
abstract class Organism  {
    public abstract void eat();
    public abstract void breathe();
}

// 동물 클래스
class Animal extends Organism {
    @Override
    public void eat() {
        System.out.println("동물이 먹이를 먹습니다.");
    }

    @Override
    public void breathe() {
        System.out.println("동물이 숨을 쉽니다.");
    }
}
```

```
public class Abstract {
    public static void main(String[] args) {
        Animal a = new Animal();
        a.eat();
        a.breathe();
    }
}
```

추상 클래스 - 7

- 특징 - 3

추상 클래스를 상속 받을 경우 추상 메소드들을 모두 구현해줘야 사용 가능
아래의 케이스는 eat 메소드의 구현부를 만들어 줬지만, breathe 메소드는 여전히
추상 메소드이기 때문에 에러 발생

```
// 생물 추상화 클래스
abstract class Organism  {
    public abstract void eat();
    public abstract void breathe();
}

//식물 클래스
class Plant extends Organism {
    @Override
    public void eat() {
        System.out.println("식물은 광합성을 통해 영양을 얻습니다.");
    }

}
```

에러 발생

추상 클래스 실습

추상 클래스 활용하기 - 1

- **Q1-1. 아래 조건에 따라 추상 클래스를 만들고 기능을 구현해보세요.**

1. 물건(Item) 이라는 추상화 클래스가 있습니다.
2. Item 추상화 클래스에는 아래와 같은 멤버 변수가 있습니다.
   → 이름(name)
   → 가격(price) : 실수형
3. Item 추상화 클래스에는 아래와 같은 메소드가 있습니다.
   → 매개변수 있는 생성자(name, price)
   → use 추상화 메소드
4. Book 클래스는 Item 메소드를 상속 받습니다.
5. Book 클래스에는 아래와 같은 멤버 변수가 있습니다.
   → 작가(author)
6. Book 클래스에는 아래와 같은 메소드가 있습니다.
   → 매개변수 있는 생성자(name, price, author)
   → use() 메소드 : “책을 읽습니다.” 출력
   → getAuthor() 메소드 : author 변수의 getter

추상 클래스 활용하기 - 2

- **Q1-2. 아래 조건에 따라 추상 클래스를 만들고 기능을 구현해보세요.**

7. Pen 클래스는 Item 메소드를 상속 받습니다.
8. Pen 클래스에는 아래와 같은 멤버 변수가 있습니다.
   → 색상(color)
9. Pen 클래스에는 아래와 같은 메소드가 있습니다.
   → 매개변수 있는 생성자(name, price, color)
   → use() 메소드 : "펜으로 쓰기를 합니다." 출력
   → getColor() 메소드 : color 변수의 getter

추상화
(인터페이스)

인터페이스 - 1

- 인터페이스(inferface)

인터페이스는 추상 메소드의 묶음이며, 추상 메소드가 아닌 일반 메소드는 가질 수 없음
또한 멤버 변수또한 클래스 변수(상수)만을 가지고 모든 클래스 변수와 추상 메소드의
접근제어자는 public을 가져야 함
→ 외부에서 접근하여 사용할 수 밖에 없기 때문에

```
public interface inferface {
    public static final int num = 10;

    public abstract int getNum();
}
```

인터페이스 - 2

- 특징 - 1

아래와 같이 선언부 부분을 모두 생략해도 사용은 가능함
→ 가능한 이유 : 어차피 인터페이스는 반드시 상수 또는 추상 메소드를 사용하기 때문에

```
public interface inferface {
    public static final int num = 10;
    public abstract int getNum();

    int num2 = 20;
    int getNum2();

}
```

인터페이스 - 3

- **특징 - 2**

1. 다중 상속 가능
2. 인터페이스의 부모 클래스는 인터페이스만 가능
   → 인터페이스는 예외적으로 Object가 부모 클래스를 갖지 않음
3. 여러 개의 추상 메소드 중에서 일부만 구현 가능
   → 클래스 앞에 abstract 붙여줘야 함
   → 추상 클래스가 되므로 객체 생성 불가

인터페이스 - 4

인터페이스의 일부만 구현할 경우

인터페이스 상속

```java
//생물 추상화 클래스
interface Organism2  {

    public abstract void eat();
    public abstract void breathe();

}

//식물 추상화 클래스
abstract class Plant2 implements Organism2 {
    @Override
    public void eat() {
        System.out.println("식물은 광합성을 통해 영양을 얻습니다.");
    }
```

추상화 클래스 vs 인터페이스

- **추상화 클래스 vs 인터페이스의 차이**

1. 추상화 클래스
   → 생성자를 가질 수 있음
   → 클래스 변수, 인스턴스 변수 모두 사용 가능
   → 생성자 가질 수 있음
2. 인터페이스
   → 클래스 변수만 사용 가능
   → 추상 메소드만 가지고 있는 형태

인터페이스

- **인터페이스를 사용하는 이유**

1. 인터페이스를 상속받은 클래스가 반드시 해당 동작을 구현하도록 강제함
   → 표준화가 가능
2. 유연한 설계가 가능하며 개발 시간을 단축할 수 있음
   → 인터페이스를 참조하여 동작을 사용할 수 있으므로 클래스간의 의존성이 줄고 코드의 재사용성이 높아짐
3. 클래스들의 관계를 맺어줌