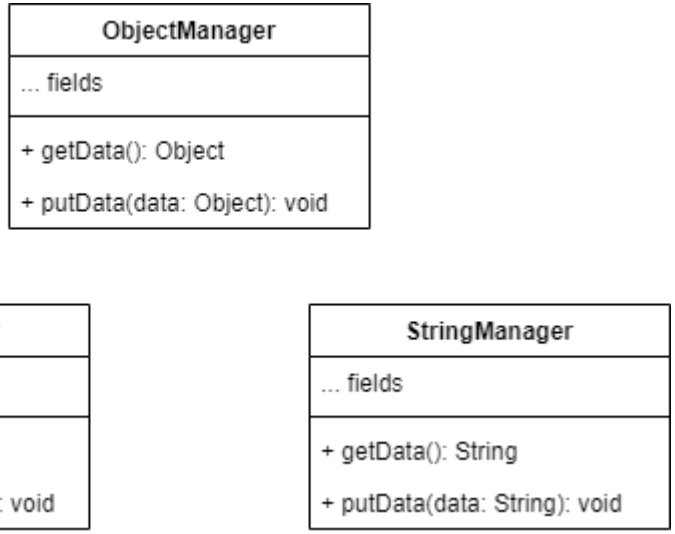
객체지향 프로그래밍 및 실습

14주차. Generic Classes and Methods

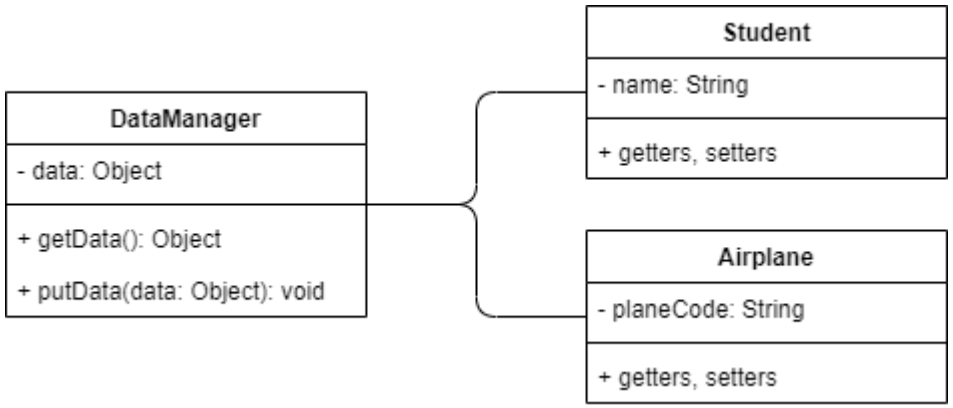
1. About Generic Programming

- Generic Programming
  - 자바는 강타입 언어이기 때문에, 프로그래밍 할 때 클래스의 타입에 대해 신중히 고려해야 한다.
  - 그렇다면 각각 다른 타입을 다루는 클래스를 만들기 위해서는 아래 구조를 적용해야 하는가?



1. About Generic Programming

- **실습 문제 1**
  - UML을 따라 클래스를 작성하고, GenericTest 클래스에 main메서드를 만들어 DataManager 객체에 Student와 Airplane 객체를 get, put 해보자.



2. Generic

- Generic Programming
  - Generic type을 이용함으로써
    잘못된 type이 사용될 수 있는 문제를 컴파일 과정에서 제거
  - Casting을 제거함으로써 프로그램 성능 향상

```
List list = new ArrayList();
list.add("hello");

// casting 필요
String result = (String) list.get(0);
```

```
// generic type 지정
List<String> list = new ArrayList<String>();
list.add("hello");

// casting 불필요
String result = list.get(0);
```

  - 컬렉션, 람다식, 스트림, NIO에서 널리 사용
  - API 문서에도 수많은 Generic expression 존재

2. Generic

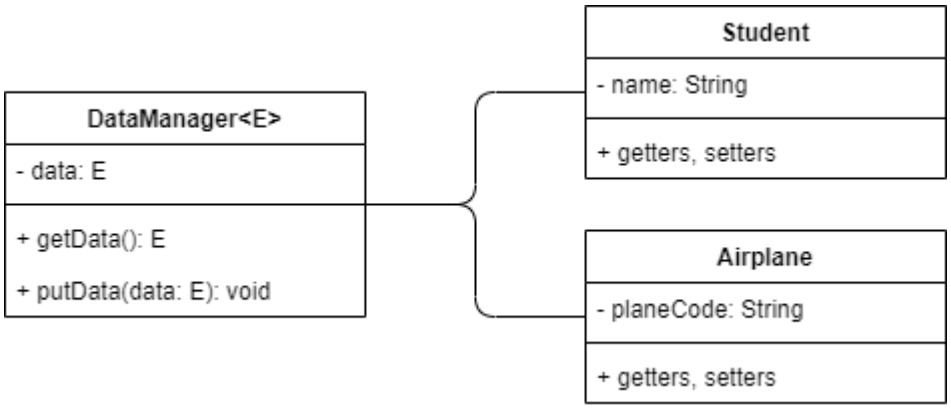
- Generic Programming
  - 타입을 parameter로 가지는 클래스와 인터페이스
  - 사용하는 데이터 타입을 미리 정의하지 않고, 런타임 시점에 타입을 정의
  - 여러 개의 타입 파라미터를 사용할 경우 콤마를 사용하여 구분 〈T, M, K …〉

```java
public class Box<T> {
    private T t;

    public T getT() {
        return t;
    }

    public void setT(T t) {
        this.t = t;
    }
}
```

```java
public class BoxTest {
    public static void main(String[] args) {

        // 사용 type을 String으로 지정
        Box<String> box1 = new Box<>();
        box1.setT("Hello");

        // 사용 type을 Integer로 지정
        Box<Integer> box2 = new Box<>();
        box2.setT(20);

        System.out.println(box1.getT());
        System.out.println(box2.getT());
    }
}
```

2. Generic

- **실습 문제 2**
  - 실습문제 1에 제네릭을 적용해보자.



3. Generic Method

- Generic Method
  - 제네릭 키워드가 클래스에 붙는 것이 아닌, 메서드에 붙을 수도 있다.
  - 제네릭의 Scpoe: 클래스에 붙느냐, 메서드에 붙느냐
    - 제네릭 키워드가 클래스에 붙으면, 클래스 전체에서 영향을 가진다.
    - 제네릭 키워드가 메서드에 붙으면, 해당 메서드에서만 영향을 가진다.

  - 제네릭이 메서드에 붙을 경우에는 타입 추론이 가능하다. 따라서 아래 예시(line 7)처럼 제네릭 인자를 넣을 필요는 없음

```
1  package genericMethods;
2
3  public class GenericMethodTest {
4      public static void main(String[] args) {
5          GenericMethod gm = new GenericMethod();
6          System.out.println(gm.<Integer>getMiddle(2, 3, 45, 2, 1, 234, 43, 5, 2));
7      }
8  }
9
10 class GenericMethod {
11     public <T> T getMiddle(T... a) {
12         return a[a.length / 2];
13     }
14 }
```

3. Generic Method

- **실습 문제 3**
  - 오른쪽 코드의 오버로딩 되어있는 printArray() 메서드를 제네릭을 이용하여 코드를 간략화 해보자

```java
package genericMethods;

public class PrintArrayTest {
    public static void main(String[] args) {
        printArray(new String[] { "a", "b", "c", "d", "e" });
        printArray(new int[] { 2, 3, 5, 5, 2, 1 });
        printArray(new boolean[] { true, false, false, true });
        printArray(new double[] { 5.1, 35.1, 138.8, 385., 35.41 });
    }

    public static void printArray(int[] array) {
        for (int e : array) {
            System.out.printf("%d ", e);
        }
        System.out.println();
    }

    public static void printArray(String[] array) {
        for (String e : array) {
            System.out.printf("%s ", e);
        }
        System.out.println();
    }

    public static void printArray(boolean[] array) {
        for (boolean e : array) {
            System.out.printf("%b ", e);
        }
        System.out.println();
    }

    public static void printArray(double[] array) {
        for (double e : array) {
            System.out.printf("%f ", e);
        }
        System.out.println();
    }
}
```

3. Bounded Type Parameter

- 제네릭 타입을 지정할 때 타입에 제한을 둘 필요가 있다.

  ❖ ex

  숫자들 (Integer, Long, Double)을 목적으로 코드를 작성했을 때
  String type이 들어오는 것을 제한하기 위해

```
public <T extends Number> void calcNumber(T t1, T t2){
        // code
}
```

4. Wildcard

- List에 담긴 요소들의 평균을 구해보자

```
1 public static double getAverage(List<Integer> list) {
2     return list.stream().mapToInt(e -> e).average().getAsDouble();
3 }
```

  - List<Integer>, List<Double>, … 등등
  - List<숫자타입> 자료형의 리스트를 받으면 되는데…
  - List<Number> list를 타입으로 하면 될까?

  - 제네릭 타입의 불공변 때문에 불가능 하다.
    - 공변: Object[] a = new Long[2]; Transprotation a = new Car();
    - 불공변: ArrayList<Object> a = new ArrayList<Integer>(); // 컴파일 불가

  - 그렇다면 오버로딩은 가능할까?
    - List<Integer> list, List<Double> list…
  - 오버로딩은 Eraser 때문에 불가능하다. 컴파일 단계에서 제네릭 타입이 소거되고 Object타입으로 대치되기 때문

4. Wildcard

- List에 담긴 요소들의 평균을 구해보자
  - 이 때 사용할 수 있는 것은?
  - Wildcard를 사용해보자

```java
package wildcard;

import java.util.ArrayList;
import java.util.List;

public class WildcardTest {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            list1.add(i);
        }

        List<Double> list2 = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            list2.add(i * 2.1);
        }

        System.out.println(getAverage(list1));
        System.out.println(getAverage(list2));

    }

    public static double getAverage(List<? extends Number> list) {
        return list.stream().mapToDouble(e -> e.doubleValue()).average().getAsDouble();
    }
}
```

4. Wildcard

- 타입 파라미터에 지정되는 타입을 제한할 필요가 있을 때 사용

〈?〉: Unbounded Wildcards (제한 없음)

모든 클래스나 인터페이스 타입이 사용 가능.

〈? extends 상위타입〉: Upper Bounded Wildcards (상위 클래스 제한)

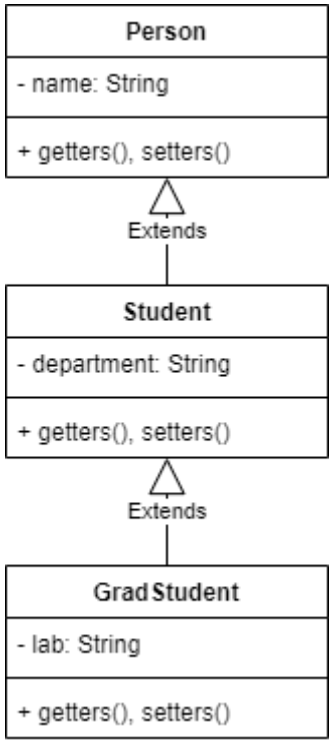
상위 타입을 상속한 클래스들만 사용 가능

〈? super 하위타입〉: Under Bounded Wildcards (하위 클래스 제한)

하위 타입을 가질 수 있는 클래스들만 사용 가능

4. Wildcard

- **실습 문제 4**
  - 오른쪽 구조의 클래스를 작성한다.
  - WildcardTest 클래스에 main메서드를 위치시킨다.
  - WildcardTest 클래스에 아래 메서드를 구현한다.
  - sayName(List<타입> list) -> 요소들의 name필드를 출력
  - sayDepartment(List<타입> list) -> 요소들의 department필드를 출력
  - sayToString(List<타입> list) -> 요소들의 toString()값을 출력
  - 이때, wildcard를 사용하여 가능한 많은 타입을 받도록 한다.



4. Wildcard

- 제네릭 타입도 부모 클래스가 될 수 있다.
- 자식 제네릭 타입은 추가적으로 제네릭 타입 파라미터를 가질 수 있다.

```
public class Child<T, M> extends Parent<T,M>{

}
```

```
public class Child<T, M, C> extends Parent<T,M>{

}
```