객체지향 프로그래밍 및 실습

5주차. 복잡한 Class and Object

1. 접근 제어자

1. 접근 제어자

■ 접근 제어자

- 접근 제어자는 멤버 또는 클래스에 사용되어, 해당하는 멤버 또는 클래스를 외부에서 접근하지 못하도록 제어하는 역할을 한다.
- 클래스나 멤버변수, 메서드, 생성자에 접근 제어자가 지정되어 있지 않다면, 접근 제어자가 default 임을 뜻한다.

제어자	같은 Class	같은 Package	자손 Class	전체
public				
protected				
(default)				
private				

대상	사용가능한 접근 제어자
클래스	public, (default)
메서드	public, protected, (default), private
멤버변수	
지역변수	없음

1. 접근 제어자

```java
package access.one;

public class One {
    private String privateField;
    protected String protectedField;
    String defaultField;
    public String publicField;

    public void accessTest() {
        privateField = "test";
        protectedField = "test";
        defaultField = "test";
        publicField = "test";
    }
}
```

접근 가능

```java
package access.one;

public class Three {
    public void accessTest() {
        One o = new One();
        o.privateField = "test";
        o.protectedField = "test";
        o.defaultField = "test";
        o.publicField = "test";
    }
}
```

접근 가능

1. 접근 제어자

```java
package access.two;

import access.one.One;

public class Two {
    public void accessTest() {
        One o = new One();
        o.privateField = "test";
        o.protectedField = "test";
        o.defaultField = "test";
        o.publicField = "test";
    }
}
```

접근 가능

2. 생성자 오버로딩

2. 생성자 오버로딩

- Overloaded Constructor
  - String id, String name
  - Int id, String name
  - 인자 타입을 달리하여 오버로딩

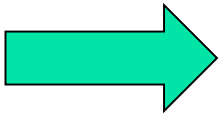
  - this()로 클래스 내부에서 생성자 호출 가능

```java
package constructor;

public class Student {
    private String id;
    private String name;

    public Student(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public Student(int id, String name) {
        this(Integer.toString(id), name);
        // 또는
        this.id = Integer.toString(id);
        this.name = name;
    }
}
```

2. 생성자 오버로딩

- Default Constructor
  - 클래스 내부에 생성자를 명시하지 않을 경우, 컴파일 단계에서 Default Constructor를 생성함

```
1 package constructor;
2
3 public class DefaultConstructor {
4     private String val;
5 }
```



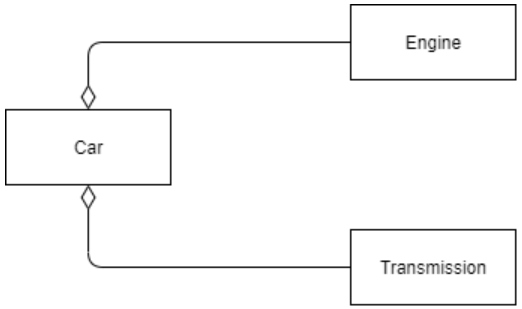
```
1 package constructor;
2
3 public class DefaultConstructor {
4     private String val;
5
6     public DefaultConstructor() { }
7 }
```

3. Composition

3. Composition

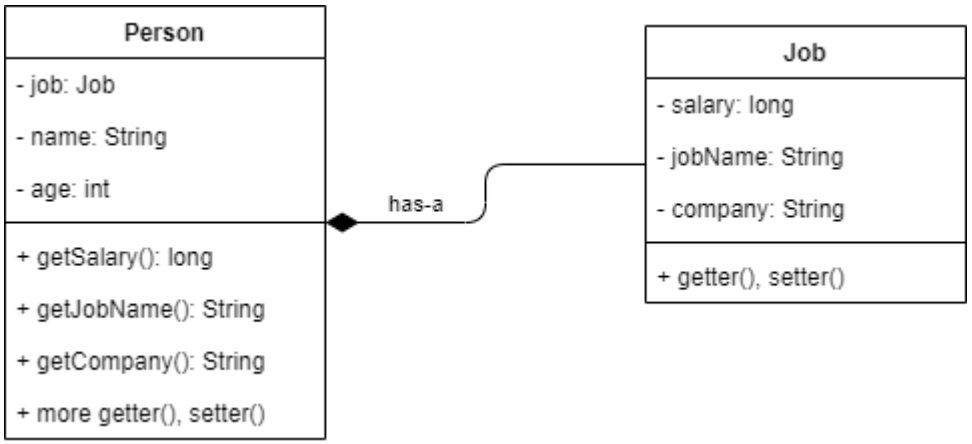
- Has-a 관계
  - Composition을 has-a 관계라고 부름
  - 다른 클래스의 객체를 멤버로 가지는 것을 composition(합성 관계)라고 말함
  - Car has a Engine
  - Car has a Transmission

```java
package composition;

public class Car {
    private Engine engine;
    private Transmission transmission;

    public Car(Engine engine, Transmission transmission) {
        this.engine = engine;
        this.transmission = transmission;
    }
}
```



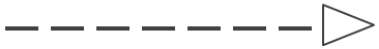
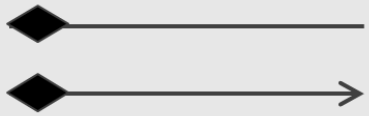
3. Composition

- **실습 문제 1**
  - 아래 Class Diagram을 참고해서 코드를 작성해보자
  - Person has-a Job
  - Person 및 Job에 대한 알맞은 생성자를 작성한다
  - CompositionTest클래스에 main메소드를 생성하여 Person과 Job이 예상대로 잘 작동하지는 확인한다



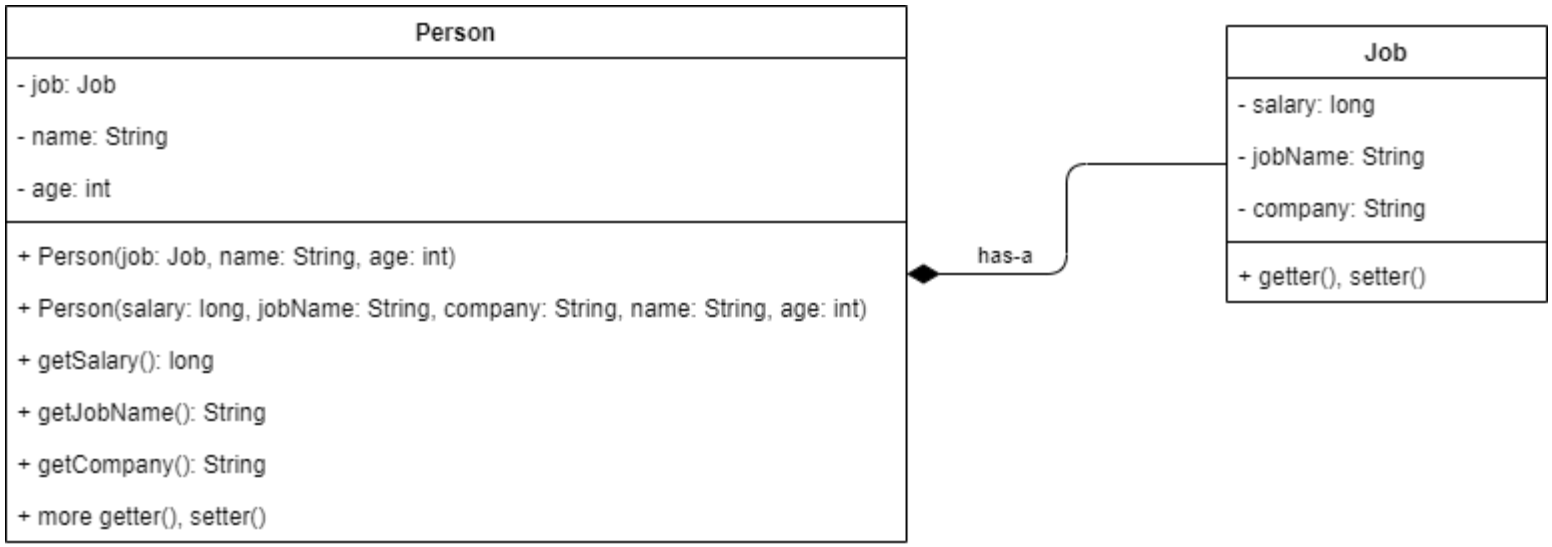
3. Composition

- 참고사항
  - https://gmlwjd9405.github.io/2018/07/04/class-diagram.html

관계	UML 표기
Generalization (일반화)	
Realization (실체화)	
Dependency (의존)	
Association (연관)	
Directed Association (직접연관)	
Aggregation (집합, 집합연관)	
Composition (합성, 복합연관)	

3. Composition

- **실습 문제 2**
  - 실습 문제 1의 코드에 Overloaded 생성자를 적용해보자



4. Static class members

4. Static class members

- static 키워드
  - 정적(static) 멤버는 클래스(객체가 아님!)에 소속되어있다.
  - 쉽게 말하면 static 키워드가 붙은 변수, 메서드가 이에 해당

```java
package org.statickey;

public class Circle {
    private static final double PI = 3.14;
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double getArea() {
        return radius * radius * PI;
    }

    public static double getPI() {
        return PI;
    }
}
```

4. Static class members

- static 멤버
  - 객체에 소속될 이유가 없는 변수들
    - 자연상수e, PI, 빛의 속도 등의 상수 값
    - 모든 객체가 공통의 값을 가지는 경우
  - 클래스에 종속되어야 할 경우
    - 모든 객체가 값을 공유해야 할 경우

```java
public class HansinPocha {
    private static String CEO;   // 대표이사
    private String shopKeeper;   // 점주
    private String location;     // 가게 위치
}
```

  - 한신포차 – 수원인계점, 수원팔달점, 오산점, 동탄점
    - 다양한 위치에 다양한 점주(동탄점의 점주가 바뀌어도 다른 지점은 영향 없음)
    - 하지만 CEO가 바뀌면 모든 지점에 영향이 감

5. Static import

- Math 클래스 메서드의 활용

```java
package staticimport;

public class StaticImportTest {
    public static void main(String[] args) {
        System.out.printf("sqrt(900.0): %f", Math.sqrt(900.0));
    }
}
```

```java
package staticimport;

import static java.lang.Math.sqrt;

public class StaticImportTest {
    public static void main(String[] args) {
        System.out.printf("sqrt(900.0): %f", sqrt(900.0));
    }
}
```

6. final instance variables

- 수정이 불가능한 변수 – 상수
  - final 변수는 선언과 동시에 또는 생성자를 통해서 초기화 될 수 있다
  - 초기화된 이후에는 수정이 불가능

```
1 package finaltype;
2
3 public class FinalType {
4     private final int INCREMENT;
5
6     public FinalType() {
7         INCREMENT = 5;
8     }
9 }
```

```
1 package finaltype;
2
3 public class FinalType {
4     private final int INCREMENT = 5;
5 }
```

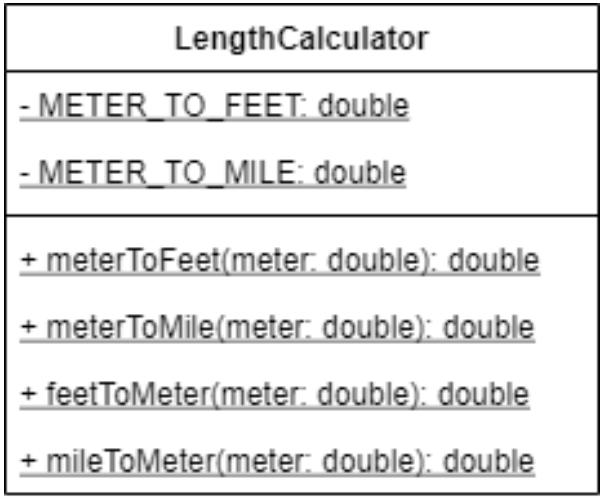
6. final instance variables

- Math 클래스에서의 활용
  - Math클래스의 PI, E등의 클래스변수는 수정이 불가능 하도록 final로 선언되어 있다
  - private -> 은닉화의 차원에서

```
public static final double E = 2.7182818284590452354;

/**
 * The {@code double} value that is closer than any other to
 * <i>pi</i>, the ratio of the circumference of a circle to its
 * diameter.
 */
public static final double PI = 3.14159265358979323846;

/**
 * Constant by which to multiply an angular value in degrees to obtain an
 * angular value in radians.
 */
private static final double DEGREES_TO_RADIANS = 0.017453292519943295;

/**
 * Constant by which to multiply an angular value in radians to obtain an
 * angular value in degrees.
 */
private static final double RADIANS_TO_DEGREES = 57.29577951308232;
```

6. final instance variables

- **실습 문제 3**
  - 직접 utility를 제공하는 클래스를 설계하고, 이를 static import 하여 사용해보자
  - METER_TO_FEET = 3.281
  - METER_TO_MILE = 0.000621371



7. Package Access

7. Package Access

- Package Access란
  - 메서드나 변수의 접근 제어자가 생략된 경우, Package access를 가진다고 말함

```java
package packageaccess.higher;

public class PackageData {
    int number = 0;
    String string = "Hello";

    public String toString() {
        return String.format("number: %d; string: %s", number, string);
    }
}
```

```java
package packageaccess;

import packageaccess.higher.PackageData;

public class PackageAccessTest {
    public static void main(String[] args) {
        PackageData pd = new PackageData();

        pd.number = 5;  // 접근 불가
    }
}
```

```java
package packageaccess.higher;

public class PackageAccessTest {
    public static void main(String[] args) {
        PackageData pd = new PackageData();

        pd.number = 5; // 접근 가능.
    }
}
```

8. 실습 종합 문제

- **실습 문제 4**
  - BookStore has-a Book 관계를 구성해보자
  - BookUtil에 환율계산 기능을 작성하여 static import하여 사용해보자