Lab Exercise #3

객체와 클래스 실습

2005 봄학기

고급 프로그래밍

김 영 국
충남대 전기정보통신공학부

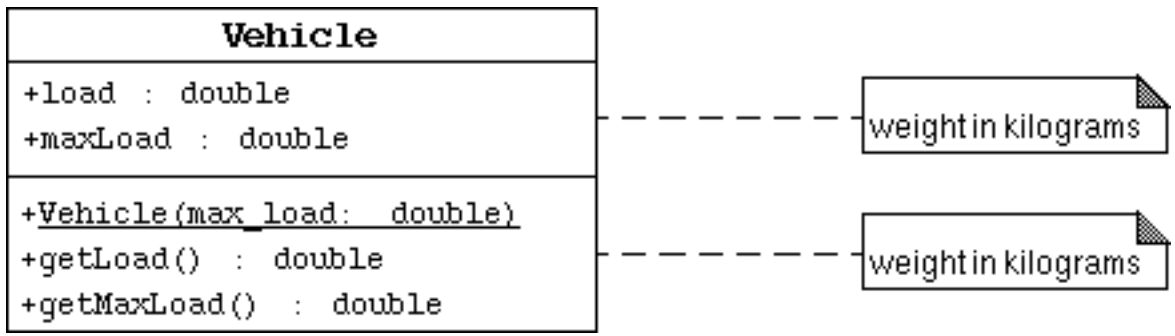
실습 내용

- 실습과제 3.1: 캡슐화(정보은닉)
- 실습과제 3.2: 은행 관련 클래스 만들기
  - Banking 패키지와 Account 클래스
  - Customer 클래스
  - Bank 클래스

실습과제 3.1 – 정보은닉

- Version 1: No Information Hiding
- Version 2: Basic Information Hiding
- Version 3: Change Internal Representation of Weight

Version 1: No Information Hiding (1)



- 테스트 프로그램 TestVehicle이 Vehicle의 애트리뷰트들을 직접 액세스할 수 있도록 접근자를 *public*으로 한다.

Version 1: No Information Hiding (2)

- 위의 UML 다이어그램을 구현하는 Vehicle 클래스를 만든다.
  - 애트리뷰트 : `load` – 현재 짐의 무게
    `maxLoad` – 최대한 실을 수 있는 짐의 무게
  - 생성자 : `maxLoad`를 초기화
  - 메소드 : `load`와 `maxLoad`값을 얻어 오는 `getLoad()`, `getMaxLoad()`

[참고] UML의 기호

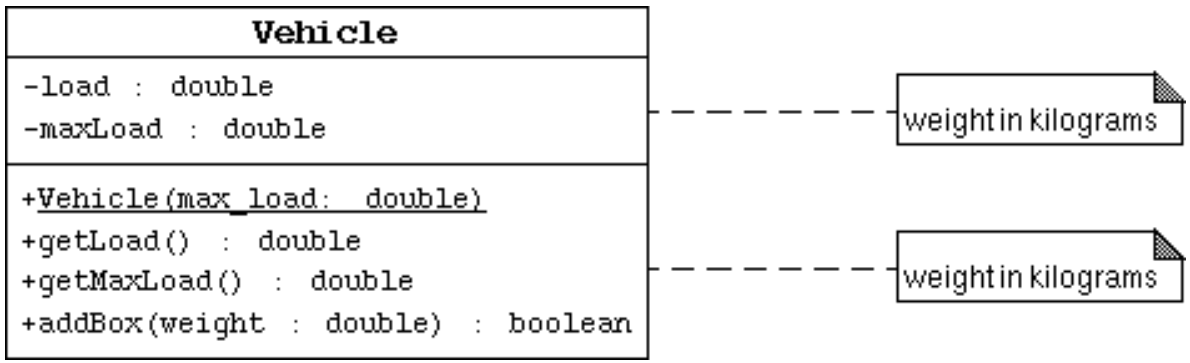
+: public, -: private, #: protected

Version 1: No Information Hiding (3)

- 아래와 같이 출력하는 TestVehicle 클래스를 만든다.

```
Creating a vehicle with a 10,000kg maximum load.
Add box #1 (500kg)
Add box #2 (250kg)
Add box #3 (5000kg)
Add box #4 (4000kg)
Add box #5 (300kg)
Vehicle load is 10050.0 kg
```

Version 2: Basic Information Hiding (1)



- 적재초과를 막기 위해 내부 클래스 데이터를 숨기고(`private`), 적재 초과 여부를 확인할 수 있는 `addBox()` 메소드를 제공한다.

Version 2: Basic Information Hiding (2)

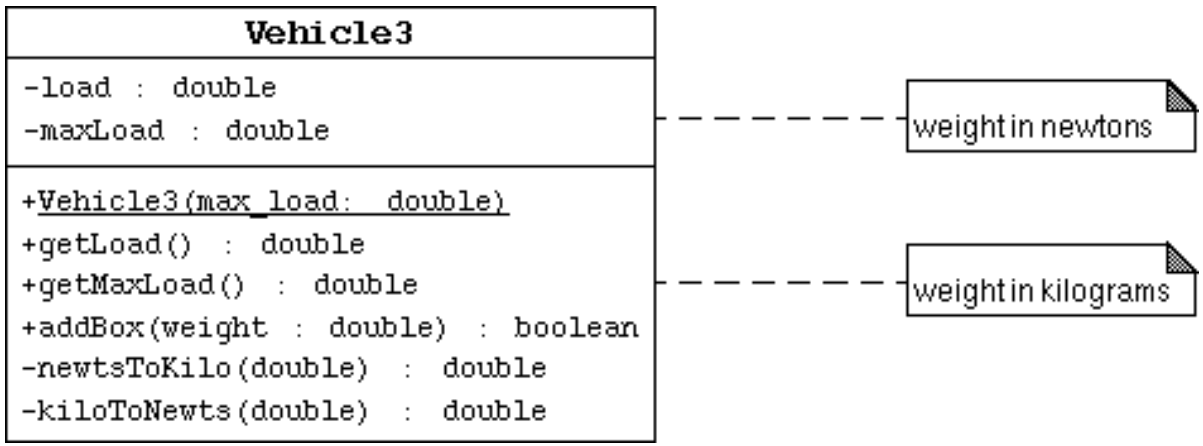
- 위의 UML 다이어그램을 구현하는 Vehicle 클래스를 만든다.
  - 애트리뷰트 : `load, maxLoad`의 접근자를 `private`으로 수정
  - 메소드 : 적재량을 매개변수로 받아서 적재량에 더하는 `addBox()`를 추가한다.
    최대적재량을 초과하면 `false`, 초과하지 않으면 `true`를 반환한다.

Version 2: Basic Information Hiding (3)

- 아래와 같이 출력하는 TestVehicle 클래스를 만든다.

```
Creating a vehicle with a 10,000 kg maximum load.
Add box #1 (500kg) : true
Add box #2 (250kg) : true
Add box #3 (5000kg) : true
Add box #4 (4000kg) : true
Add box #5 (300kg) : false Vehicle load is 9750.0 kg
```

Version 3: Change Internal Representation of Weight (1)



- 클래스 내부 데이터의 단위는 Newton이고, 외부 데이터의 단위는 kilogram이다. 서로 다른 단위로 값을 변환하기 위한 메소드를 추가한다.

Version 3: Change Internal Representation of Weight (2)

- 위의 UML 다이어그램을 구현하는 Vehicle 클래스를 만든다.
  - 단위를 변환할 수 있는 메소드를 추가한다.

```
private double kiloToNewts(double weight)
{ return (weight * 9.8); }
private double newtsToKilo(double weight)
{ return (weight / 9.8); }
```

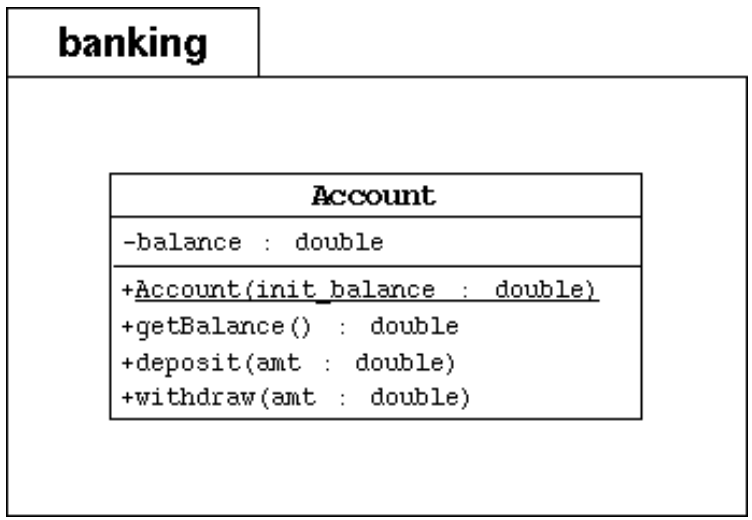
  - 생성자, `getLoad()`, `getMaxLoad()`, `addBox()` 메소드가 위의 메소드를 사용하여 단위를 변환하도록 수정한다.

Version 3: Change Internal Representation of Weight (2)

- Version 2의 TestVehicle 클래스를 사용하여 출력한 결과는 아래와 같다.

```
Creating a vehicle with a 10,000 kg maximum load.
Add box #1 (500kg) : true
Add box #2 (250kg) : true
Add box #3 (5000kg) : true
Add box #4 (4000kg) : true
Add box #5 (300kg) : false Vehicle load is 9750.0 kg
```

실습과제 3.2 – 은행 관련 클래스



- banking 패키지 내에 계좌 정보를 가지고 있는 Account 클래스를 작성한다. Accout 클래스에 간단한 트랜잭션을 실행하는 TestBanking 클래스를 작성한다.

banking 패키지(1)

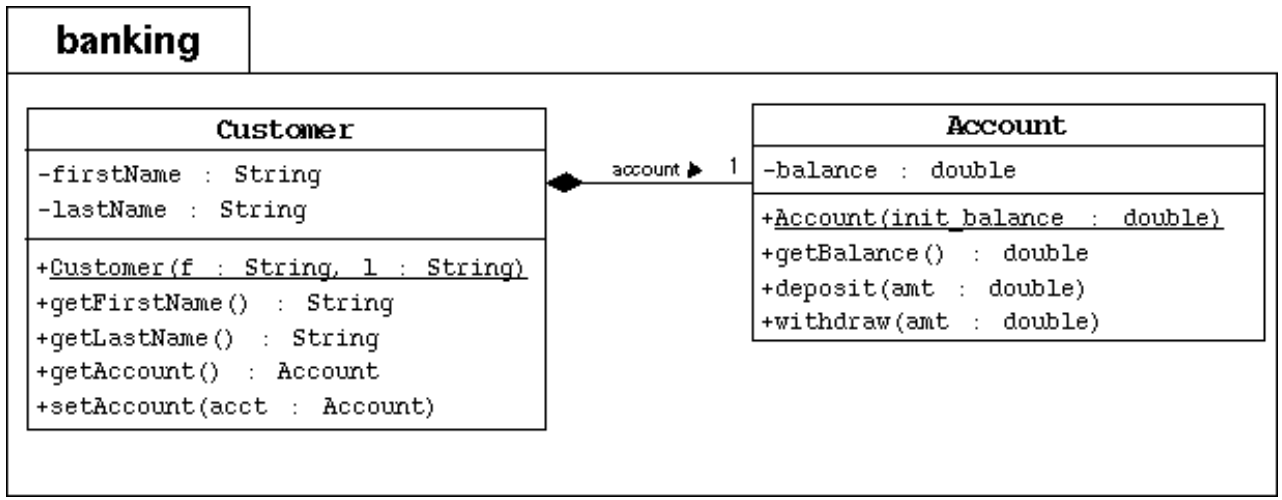
- 디렉토리
  - C:\이름\TestBanking.java
    C:\이름\banking\Account.java
- Account 클래스
  - 파일의 맨 위에 package banking; 추가한다.
  - 애트리뷰트 : balance – 계좌의 현재 잔고
  - 생성자 : balance를 초기화(double형의 파라미터)
  - 메소드 : getBalance() – 현재 잔고를 반환
    deposit() – 입금을 실행
    withdraw() – 출금을 실행

banking 패키지(2)

- TestBanking 클래스
  - 파일의 맨 위에 `import banking.*;` 추가한다.
  - 계좌를 생성하여 입, 출금을 실행하고 계좌의 상태를 출력하는 결과는 아래와 같다

```
Creating an account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
The account has a balance of 324.88
```

Customer 클래스 만들기(1)



- banking 패키지에 Customer 클래스를 추가하여 banking 프로젝트를 확장한다. Customer 클래스는 Account 오브젝트를 가지고 있다.

Customer 클래스 만들기(2)

- Customer 클래스
  - 파일의 맨 위에 package banking; 추가한다.
  - 애트리뷰트 : firstName, lastName, account
  - 생성자 : String 타입의 f와 l값을 파라미터로 받아 firstName, lastName을 초기화
  - 메소드 : getFirstName(), getLastName()
    - firstName과 lastName값을 단순히 리턴하는 메소드

    setAccount() – account에 값을 할당

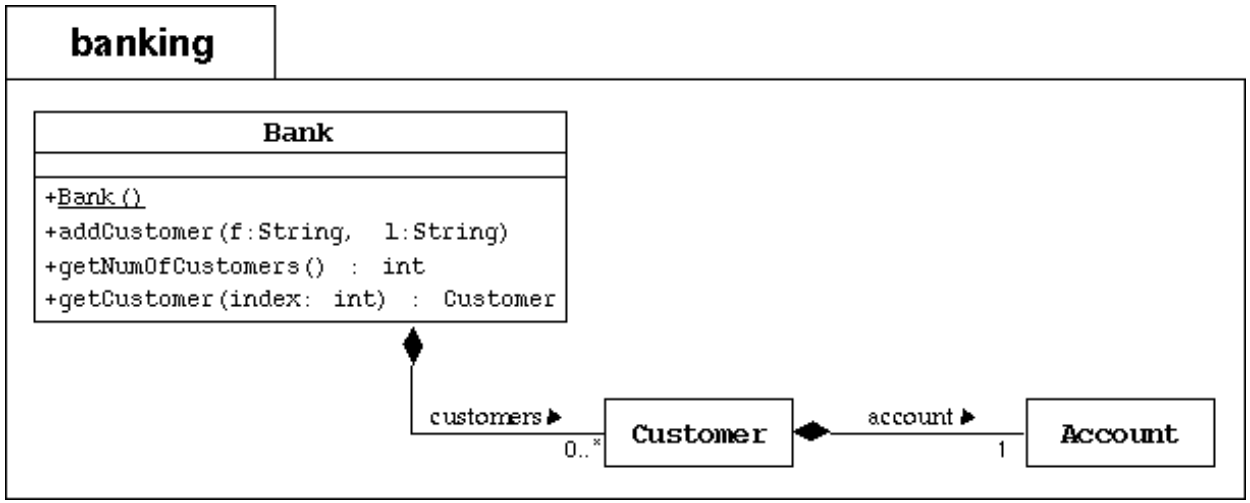
    getAccount() – account의 값을 리턴

Customer 클래스 만들기(3)

- TestBanking 클래스
  - 파일의 맨 위에 **import banking.*;** 추가한다.
  - 고객을 생성하고, 고객의 계좌에 대한 입, 출금을 실행하여 고객의 정보를 출력하는 결과는 아래와 같다

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

Bank 클래스 만들기(1)



- banking 패키지에 Bank 클래스를 추가한다. Bank 클래스는 여러 명의 고객을 가질 수 있도록 구현한다.

Bank 클래스 만들기(2)

- Bank 클래스
  - 애트리뷰트 : customers – Customer[]형
    numberOfCustomers – customers 배열의 다음 index를 유지하기 위한 int형
  - 생성자 : customers의 배열 크기를 10으로 초기화
  - 메소드 : addCustomer() – 새로운 Customer 객체를 받아서 customers에 할당한다. 이 때 numberOfCustomers의 값을 증가시킨다.
    getNumOfCustomers() –numOfCustomers 값을 리턴
    getCustomer() – 주어진 index 파라미터에 해당하는 customer를 리턴

Bank 클래스 만들기(3)

- TestBanking 클래스
  - 파일의 맨 위에 import banking.*; 추가한다.
  - bank 클래스의 customers를 출력하는 결과는 아래와 같다.

```
Customer [1] is Simms, Jane
Customer [2] is Bryant, Owen
Customer [3] is Soley, Tim
Customer [4] is Soley, Maria
```