2023. 07. 06. ~ 2023. 11. 30.

[부산광역시] 클라우드 플랫폼 기반 웹서비스 개발자 양성 과정

2023. 08. 08. 화요일

데이터 활용 저장을 위한 프로그래밍 및 DBMS　　이름 :

1. 다음 코드 조각이 어떻게 동작하는지 설명하고, 상속, 캡슐화, 다형성, 추상화의 각 원칙이 이 예제에서 어떻게 적용되는지 설명하십시오. (30점)

```
abstract class Vehicle {
    private String color;

    Vehicle(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    abstract void run();
}

class Car extends Vehicle {
    Car(String color) {
        super(color);
    }

    void run() {
        System.out.println("The "+getColor() +" car runs.");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle vehicle =new Car("red");
        vehicle.run();
    }
}
```

가이드라인:

- 코드의 동작 설명: 프로그램이 무엇을 하는지, 어떻게 동작하는지에 대한 전반적인 설명을 제공해야 합니다.
- 상속: 여기에서 상속이 어떻게 사용되었는지, Car가 Vehicle 클래스로부터 어떤 속성과 메서드를 상속받았는지 설명해야 합니다.
- 캡슐화: 캡슐화가 어떻게 이루어졌는지, 이를 통해 어떤 데이터가 보호되는지 설명해야 합니다.
- 다형성: Vehicle의 참조를 통해 Car의 run 메서드를 호출하는 것이 다형성의 예인지, 그 이유는 무엇인지 설명해야 합니다.
- 추상화: 추상 클래스 Vehicle과 추상 메서드 run이 어떤 역할을 하는지, 이것이 추상화에 어떻게 기여하는지 설명해야 합니다.

이 코드의 기본적인 동작은 Vehicle라는 추상 클래스를 정의하고, Car라는 구체적인 클래스를 Vehicle로부터 상속받아 Vehicle의 추상 메소드 run()을 구현합니다. 메인 함수에서는 Vehicle 타입의 변수로 Car 인스턴스를 참조하고, run() 메소드를 호출합니다. 이때 출력되는 메시지는 "The red car runs." 입니다.

상속: 이 코드에서 Car 클래스는 Vehicle 클래스를 상속합니다. 이는 Car가 Vehicle의 속성과 메서드를 상속받을 수 있음을 의미하며, Vehicle의 getColor() 메서드와 color 필드가 이에 해당합니다.

캡슐화: Vehicle 클래스에서 color 필드는 private로 선언되어 있습니다. 이는 직접 접근이 제한되며, 외부에서는 getColor() 메서드를 통해 이 필드의 값을 가져올 수 있습니다. 이런 방식으로 데이터를 보호하며, 이것이 캡슐화의 한 예입니다.

다형성: 이 코드에서 Vehicle 타입의 변수 vehicle로 Car 인스턴스를 참조하고 있습니다. vehicle.run()을 호출하면 실제 인스턴스인 Car의 run() 메서드가 실행됩니다. 이는 다형성의 한 예로, 부모 클래스 타입의 참조를 통해 자식 클래스의 메서드를 실행할 수 있습니다.

추상화: 이 코드에서 Vehicle은 추상 클래스로 정의되어 있습니다. Vehicle 클래스는 color라는 속성과 run이라는 추상 메서드를 가지고 있습니다. Vehicle 클래스는 자체적으로 인스턴스화할 수 없으며, 구체적인 하위 클래스인 Car에서 run 메서드를 구체적으로 구현해야 합니다. 이런 방식으로 Vehicle 클래스는 자동차의 주요 행동인 "달리기"를 정의하지만, 그 구체적인 동작 방식은 하위 클래스에게 맡기는 추상화를 구현하고 있습니다.

2. 아래의 설명에 따라 Person 클래스와 PersonDatabase 클래스를 작성하십시오. PersonDatabase 클래스는 Person 객체에 대한 CRUD(Create, Read, Update, Delete) 연산을 모두 구현해야 합니다. (40점)

1. Person 클래스는 name (String)과 id (int) 필드를 가지며, 이는 각각 사람의 이름과 식별자를 나타냅니다. 생성자를 포함하여 필요한 getter, setter 메서드를 구현하십시오.

2. PersonDatabase 클래스는 Person 객체의 리스트를 저장하는 List<Person> 타입의 private 필드를 가지며, 이는 Person 객체의 데이터베이스를 나타냅니다.

3. PersonDatabase 클래스는 다음 메서드를 구현해야 합니다:

addPerson(Person person): 새로운 Person 객체를 리스트에 추가합니다. (Create)
getPerson(int id): 주어진 ID를 가진 Person 객체를 리스트에서 찾아 반환합니다. 만약 해당 ID를 가진 Person이 없다면 null을 반환합니다. (Read)
updatePerson(int id, String newName): 주어진 ID를 가진 Person 객체의 이름을 새 이름으로 수정합니다. 만약 해당 ID를 가진 Person이 없다면 아무런 동작도 수행하지 않습니다. (Update)
deletePerson(int id): 주어진 ID를 가진 Person 객체를 리스트에서 삭제합니다. 만약 해당 ID를 가진 Person이 없다면 아무런 동작도 수행하지 않습니다. (Delete)

가이드라인:

Person 클래스의 필드는 캡슐화 원칙을 따라야 합니다. 즉, 직접 접근이 불가능하도록 private로 선언되어야 하며, 필요한 getter와 setter 메서드를 통해 접근할 수 있도록 해야 합니다.
PersonDatabase 클래스의 메서드에서는 리스트를 순회하면서 ID가 일치하는 Person 객체를 찾아야 합니다. 이때, 'Read'와 'Delete' 연산에서는 일치하는 객체를 찾지 못하면 null을 반환하거나 아무런 동작도 수행하지 않는 등의 예외 처리를 해야 합니다.

```java
class Person {
    private String name;
    private int id;

    Person(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }
}

class PersonDatabase {
    private List<Person> persons =new ArrayList<>();

    public void addPerson(Person person) {
        persons.add(person);
    }

    public Person getPerson(int id) {
        for (Person person : persons) {
            if (person.getId() == id) {
                return person;
            }
        }
        return null;
    }

    public void updatePerson(int id, String newName) {
        Person person =getPerson(id);
        if (person !=null) {
            person.setName(newName);
        }
    }

    public void deletePerson(int id) {
        Person toRemove =null;
        for (Person person : persons) {
            if (person.getId() == id) {
                toRemove = person;
                break;
            }
        }
        if (toRemove !=null) {
            persons.remove(toRemove);
        }
    }
}
```

3. MySQL을 사용하여 다음 작업을 수행하는 SQL 쿼리를 작성하십시오. (30점)

##

Employee라는 이름의 테이블을 생성합니다. 테이블은 id (정수형, 기본 키, 자동 증가), name (문자열, 최대 길이 100), departmentId (정수형) 라는 세 개의 필드를 포함해야 합니다. (DDL)

##

Department라는 이름의 테이블을 생성합니다. 테이블은 id (정수형, 기본 키, 자동 증가), name (문자열, 최대 길이 100)라는 두 개의 필드를 포함해야 합니다. (DDL)

##

Employee 테이블에 다음 데이터를 추가합니다: {name: 'John Doe', departmentId: 1} 그리고 Department 테이블에 {id: 1, name: 'Sales'} 데이터를 추가합니다. (DML)

##

name이 'John Doe'인 직원의 departmentId를 2로 업데이트합니다. 그리고 Department 테이블에 {id: 2, name: 'HR'} 데이터를 추가합니다. (DML)

##

Employee 테이블과 Department 테이블을 JOIN하여 직원 이름과 해당 직원의 부서 이름을 조회합니다. (JOIN)

##

Employee 테이블에 있는 각각의 departmentId에 대해 해당 ID를 가진 직원 수를 조회합니다. (GROUP BY)

##

DBUser라는 사용자를 생성하고, Employee 테이블과 Department 테이블에 대해 SELECT, INSERT, UPDATE 권한을 부여합니다. 이 사용자의 비밀번호는 'password'입니다. (DCL)

##

가이드라인:

테이블 생성은 CREATE TABLE 문을 사용하며, 필드 선언에는 필드명, 데이터 타입, 필수 여부 등을 포함해야 합니다.
데이터 추가는 INSERT INTO 문을 사용하며, 대응되는 필드명과 값을 명시해야 합니다.
데이터 수정은 UPDATE 문을 사용하며, 조건에 맞는 데이터를 선택하여 필드의 값을 변경해야 합니다.
JOIN 연산은 JOIN 키워드를 사용하며, 두 테이블이 어떤 필드를 기준으로 조인되는지 명시해야 합니다.
그룹화는 GROUP BY 키워드를 사용하며, 어떤 필드를 기준으로 데이터를 그룹화할지 명시해야 합니다.
사용자 생성과 권한 부여는 CREATE USER 문과 GRANT 문을 사용해야 합니다. 이때, 사용자명, 비밀번호, 권한 등을 명시해야 합니다.

##

```sql
CREATE TABLE Employee (
    id INT AUTO_INCREMENT,
    name VARCHAR(100),
    departmentId INT,
    PRIMARY KEY(id)
);

CREATE TABLE Department (
    id INT AUTO_INCREMENT,
    name VARCHAR(100),
    PRIMARY KEY(id)
);
```

##

```sql
INSERT INTO Employee (name, departmentId)
VALUES ('John Doe', 1);
INSERT INTO Department (id, name)
VALUES (1, 'Sales');
```

##

```sql
UPDATE Employee SET departmentId =2
WHERE name ='John Doe';
INSERT INTO Department (id, name)
VALUES (2, 'HR');
```

##

```sql
SELECT E.name, D.name
FROM Employee E
JOIN Department D ON E.departmentId = D.id;
```

##

```sql
SELECT departmentId, COUNT(*)
FROM Employee
GROUP BY departmentId;
```

##

```sql
CREATE USER 'DBUser'@'localhost'
IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE ON
database.Employee TO 'DBUser'@'localhost';
GRANT SELECT, INSERT, UPDATE ON
database.Department TO 'DBUser'@'localhost';
```