Write a Java Program to implement Strategy Pattern for Duck Behavior. Create

instance variable that holds current state of Duck from there, we just need to handle

all Flying Behaviors and Quack Behavior

Below is a Java program implementing the \*\*Strategy Pattern\*\* to manage duck behaviors (flying and quacking). The Strategy Pattern allows us to define a family of algorithms (flying and quacking behaviors), encapsulate each one, and make them interchangeable without changing the client class.

---

### \*\*Code Implementation\*\*

#### \*\*1. Interface: FlyBehavior\*\*

Defines the behavior for flying.

```java

package strategy;

public interface FlyBehavior {

void fly();

}

```

#### \*\*2. Interface: QuackBehavior\*\*

Defines the behavior for quacking.

```java

package strategy;

public interface QuackBehavior {

void quack();

}

```

#### \*\*3. Implementations of FlyBehavior\*\*

Different flying behaviors:

```java

package strategy;

public class FlyWithWings implements FlyBehavior {

@Override

public void fly() {

System.out.println("I'm flying with wings!");

}

}

public class FlyNoWay implements FlyBehavior {

@Override

public void fly() {

System.out.println("I can't fly.");

}

}

```

#### \*\*4. Implementations of QuackBehavior\*\*

Different quacking behaviors:

```java

package strategy;

public class Quack implements QuackBehavior {

@Override

public void quack() {

System.out.println("Quack! Quack!");

}

}

public class Squeak implements QuackBehavior {

@Override

public void quack() {

System.out.println("Squeak! Squeak!");

}

}

public class MuteQuack implements QuackBehavior {

@Override

public void quack() {

System.out.println("...");

}

}

```

#### \*\*5. Duck Class\*\*

Defines the Duck with behavior instances:

```java

package strategy;

public abstract class Duck {

FlyBehavior flyBehavior;

QuackBehavior quackBehavior;

public void performFly() {

flyBehavior.fly();

}

public void performQuack() {

quackBehavior.quack();

}

public void setFlyBehavior(FlyBehavior fb) {

flyBehavior = fb;

}

public void setQuackBehavior(QuackBehavior qb) {

quackBehavior = qb;

}

public abstract void display();

}

```

#### \*\*6. Specific Duck Types\*\*

Different duck types with default behaviors:

```java

package strategy;

public class MallardDuck extends Duck {

public MallardDuck() {

flyBehavior = new FlyWithWings();

quackBehavior = new Quack();

}

@Override

public void display() {

System.out.println("I'm a Mallard Duck!");

}

}

public class RubberDuck extends Duck {

public RubberDuck() {

flyBehavior = new FlyNoWay();

quackBehavior = new Squeak();

}

@Override

public void display() {

System.out.println("I'm a Rubber Duck!");

}

}

```

#### \*\*7. Main Class\*\*

Test the Strategy Pattern:

```java

package strategy;

public class DuckSimulator {

public static void main(String[] args) {

// Create a Mallard Duck

Duck mallard = new MallardDuck();

mallard.display();

mallard.performFly();

mallard.performQuack();

System.out.println("\n--- Changing Mallard's Quack Behavior ---");

mallard.setQuackBehavior(new MuteQuack());

mallard.performQuack();

// Create a Rubber Duck

Duck rubberDuck = new RubberDuck();

System.out.println("\n--- Rubber Duck ---");

rubberDuck.display();

rubberDuck.performFly();

rubberDuck.performQuack();

}

}

```

---

### \*\*Program Output\*\*

```

I'm a Mallard Duck!

I'm flying with wings!

Quack! Quack!

--- Changing Mallard's Quack Behavior ---

...

--- Rubber Duck ---

I'm a Rubber Duck!

I can't fly.

Squeak! Squeak!

```

---

### \*\*Explanation\*\*

1. \*\*Strategy Pattern\*\*:

- Encapsulates the flying and quacking behaviors (`FlyBehavior`, `QuackBehavior`).

- Makes behaviors interchangeable during runtime using setter methods (`setFlyBehavior`, `setQuackBehavior`).

2. \*\*Duck Class\*\*:

- Abstract class holding the current state (flying and quacking behaviors).

- Defines `performFly` and `performQuack` to delegate behavior execution to strategy classes.

3. \*\*Behavior Implementation\*\*:

- `FlyWithWings` and `FlyNoWay` for flying.

- `Quack`, `Squeak`, and `MuteQuack` for quacking.

4. \*\*Runtime Behavior Changes\*\*:

- Modify behavior dynamically using `setFlyBehavior` and `setQuackBehavior`.

This implementation demonstrates the flexibility and modularity of the Strategy Pattern, making it easy to extend or modify behaviors without changing the Duck class or its subclasses.