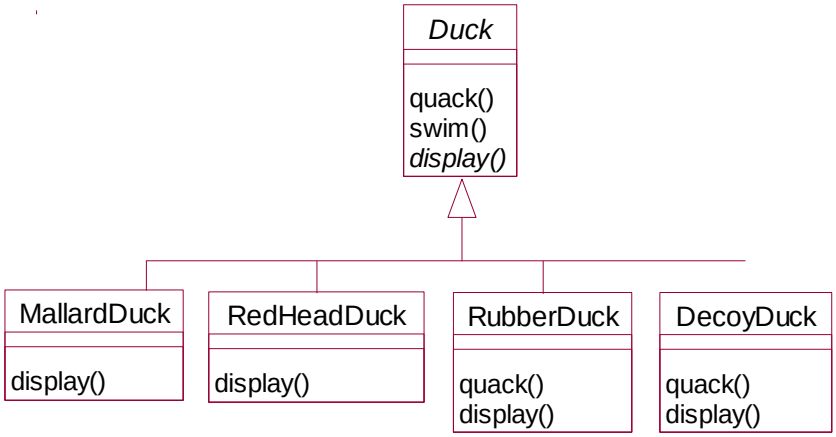
Hệ thống game mô phỏng SimUDuck

Tạo một hệ thống mô phỏng loài vịt được miêu tả như sau:

- Có nhiều loại vịt khác nhau được tích hợp vào game: "Mallard Duck", "Red Head Duck", "Rubber Duck", "Decoy Duck".
- Các con vịt có thể bơi
- Các con vịt có thể bay, kêu khác nhau, và hiển thị màu của nó (mỗi vịt có màu khác nhau)

Hệ thống phải có thể mở rộng: có thể tạo ra các con vịt với các ứng xử kêu, bay khác như mong muốn của ứng dụng

1. Thiết kế ban đầu của hệ thống dùng kế thừa:



Nhiệm vụ:
Tạo một project có tên **DuckSimulator1**, cài đặt các lớp trên và có các phương thức như sau:
1) In **"I am swimming"** khi gọi phương thức **swim()**.
2) In **"Quack, quack …"** khi gọi phương thức **quack()** method. Vịt cao su **Rubber** in **"Squick, squick …"** thay cho **"Quack, quack …"** và vịt mồi **Decoy** không in gì cả.
3) In tên loại vịt khi gọi phương thức **display()**.
4) Tạo test cho mô phỏng trong đó tạo các loại vịt, cho các con vịt hiển thị, bơi và kêu.

```
package duck1;
public abstract class Duck {
   public void swim() {
      System.out.println("I'm swimming");
   }
   public void quack() {
      System.out.println("Quack, quack ...");
   }
   public abstract void display();
}
```

```
package duck1;
public class MarllardDuck extends Duck {

   public void display() {
      System.out.println("I'm a marllard duck");
   }
}
```

```
package duck1;
public class RedHeadDuck extends Duck {
```

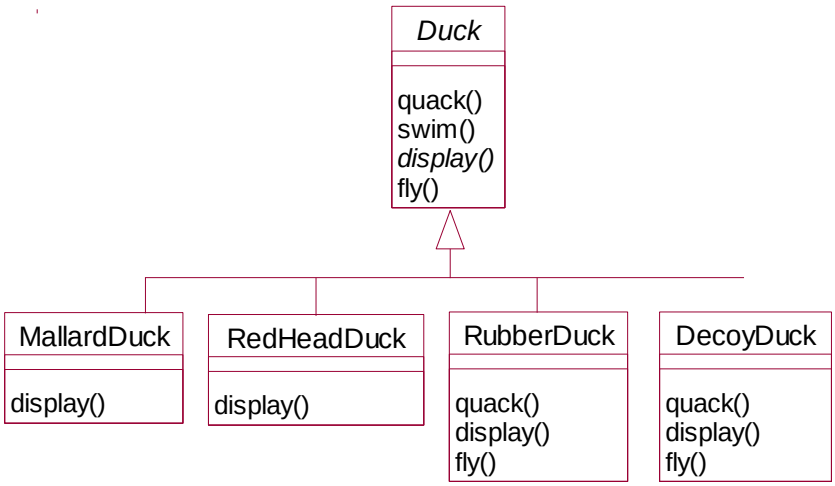
```
    public void display() {
        System.out.println("I'm a red headed duck");
    }
}
```

```
package duck1;
public class RubberDuck extends Duck {
    public void quack() {
        System.out.println("Squick, squick ...");
    }

    public void display() {
        System.out.println("I'm a rubber duck");
    }
}
```

```
package duck1;
public class DecoyDuck extends Duck {
    public void quack() {
        System.out.println("Nothing");
    }

    public void display() {
        System.out.println("I'm a decoy duck");
    }
}
```

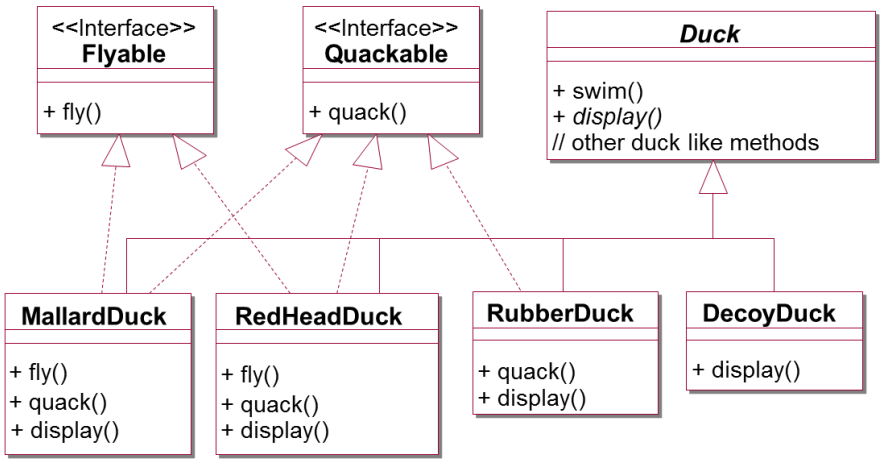
```
package duck1;
public class TestDrive {
    public static void main(String[] args) {
        Duck marllard = new MarllardDuck();
        Duck redHead = new RedHeadDuck();
        Duck rubber = new RubberDuck();
        Duck decoy = new DecoyDuck();

        marllard.display();
        marllard.swim();
        marllard.quack();

        redHead.display();
        redHead.swim();
        redHead.quack();

        rubber.display();
        rubber.swim();
        rubber.quack();

        decoy.display();
        decoy.swim();
        decoy.quack();
    }
}
```

2. **Yêu cầu mới:**
   Chúng ta phải tạo các con vịt có thể bay. **Rubber Duck** và **Decoy Duck** không thể bay được.



Nhiệm vụ:
1) Tạo một project tên **DuckSimulator2**, chép từ project **DuckSimulator1**.
2) Tạo phương thức **fly()** trong lớp cha **Duck** và in **"I am flying"**
3) Cài đạt lại phương thức fly() trong **RubberDuck** và **DecoyDuck** không làm gì cả.
4) Viết test kiểm tra ứng xử bay mới của các loại vịt.

3. **Hiệu chỉnh thiết kế của hệ thống tách ứng xử quake và fly thành interface, và dùng kế thừa.**



Nhiệm vụ:
1) Create a new Java Project called DuckSimulator3, which is a copy of the project DuckSimulator1.
2) Create two interfaces: Flyable and Quackable and their methods fly() and quack(), respectively.
3) Delete quack() from the super-class Duck (we don't need it anymore).
4) The Mallard Duck, Rubber Duck and Red Head Duck must now implement Quackable interface. The Mallard Duck and Red Head Duck implements Flyable interface.
5) Run the test again.

Nhận xét:
Interfaces are dynamic and are easy to use. However, we lost code-reuse.

```
package duck2;
```

```
public abstract class Duck {
   public void swim() {
      System.out.println("I'm swimming");
   }

   public abstract void display();
}
```

```
package duck2;
public interface Flyable {
   void fly();
}
```

```
package duck2;
public interface Quackable {
   void quack();
}
```

```
package duck2;
public class MarllardDuck extends Duck implements Flyable, Quackable {
   public void display() {
      System.out.println("I'm a mallard duck");
   }

   public void fly() {
      System.out.println("I'm flying");
   }

   public void quack() {
      System.out.println("Quack, quack ...");
   }
}
```

```
package duck2;
public class RedHeadDuck extends Duck implements Flyable, Quackable {
   public void display() {
      System.out.println("I'm a red head duck");
   }

   public void fly() {
      System.out.println("I'm flying");
   }

   public void quack() {
      System.out.println("Quack, quack ...");
   }
}
```

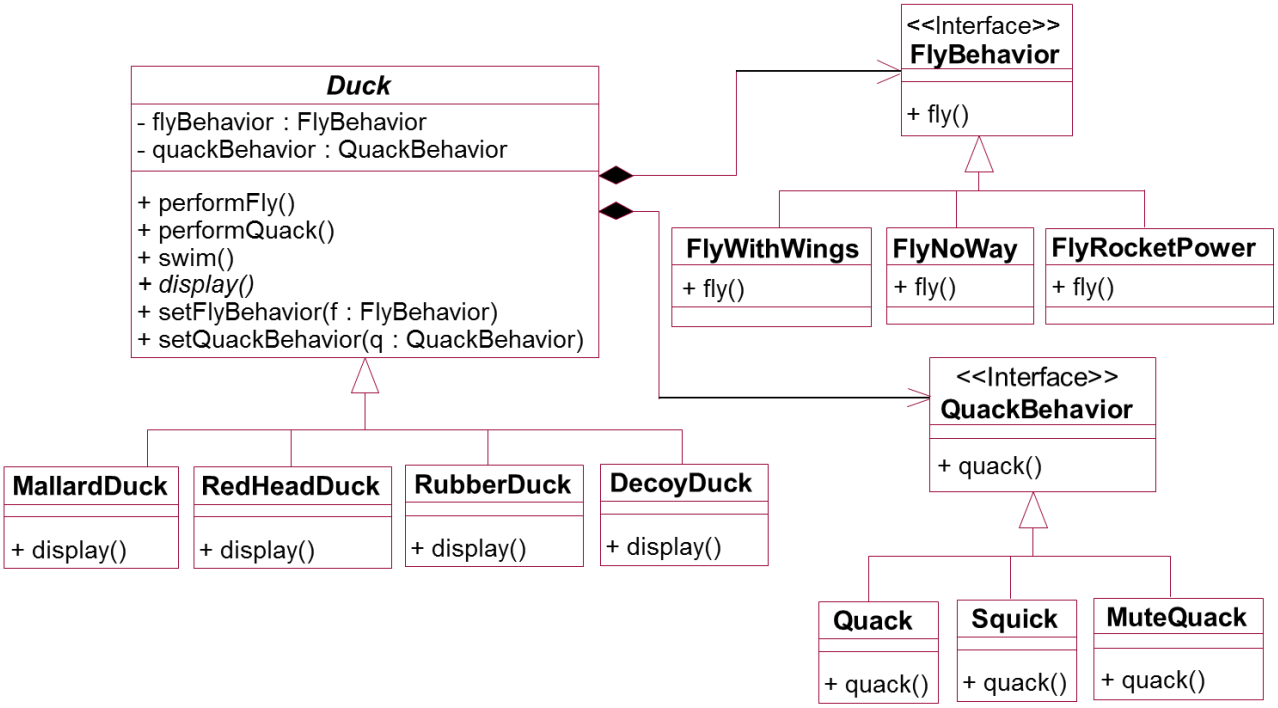
```
package duck2;
public class RubberDuck extends Duck implements Quackable {
   public void display() {
      System.out.println("I'm a rubber duck");
   }

   public void quack() {
      System.out.println("Squick, squick ...");
   }
}
```

```java
package duck2;
public class DecoyDuck extends Duck {
   public void display() {
      System.out.println("I'm a decoy duck");
   }
}
```

```java
package duck2;
public class TestDrive {
   public static void main(String[] args) {
      Duck marllard = new MarllardDuck();
      Duck rubber = new RubberDuck();
      Duck decoy = new DecoyDuck();

      marllard.display();
      ((Quackable) marllard).quack();
      ((Flyable) marllard).fly();

      rubber.display();
      ((Quackable) rubber).quack();

      decoy.display();
      decoy.swim();
   }
}
```

4. Thiết kế của hệ thống dùng Strategy pattern

Các ứng xử quack và fly bây giờ được bao bọc trong các đối tượng (strategy pattern)



B> Tasks

1) Create a new Java Project called DuckSimulator4, which is a copy of the project DuckSimulator1.
2) Create two interfaces for our two types of behaviors: **QuackBehavior** and **FlyBehavior**. **quack()** and **fly()** are two methods of these interfaces respectively.
3) **FlyWithWings** and **FlyNoWay** are two classes that implement the **FlyBehavior** interface. **Quack**, **Squeak** and **Mute** are three classes that implement the **QuackBehavior** interface.
4) Every duck now should have a fly behavior and a quack behavior. Therefore, we put these objects in the **Duck** super-class.
5) Create two methods **performQuack()** and **performFly()** in the **Duck** super-class . These methods invoke the **quack()** and **fly()** methods in the quack behavior and fly behavior objects respectively.
6) In the constructor of each type of **Duck**, we should instantiate the their default fly and quack behavior.
7) Run the test again.
8) Try to change the **Rubber** duck fly behavior at run-time by allowing user to choose either strategy: **FlyWithWings** and **FlyNoWay**.

C> Remarks

Instead of using inheritance (is-a relation) we now use composition (has-a relation) to achieve code reuse. Composition provides us more flexibility than inheritance.

D> Questions

1) What are the advantages of using the Strategy Pattern over a simple if-else-if chain or case statement?

Cài đặt code và bổ sung các phần còn thiếu (// TODO)

```
package duck3;
public interface FlyBehavior {
```

```
    public void fly();
}
```

```
package duck3;
public class FlyWithWings implements FlyBehavior {
    public void fly() {
        System.out.println("I'm flying!!");
    }
}
```

```
package duck3;
public class FlyNoWay implements FlyBehavior {
    public void fly() {
        System.out.println("I can't fly");
    }
}
```

```
package duck3;
public interface QuackBehavior {
    public void quack();
}
```

```
package duck3;
public class Quack implements QuackBehavior {
    public void quack() {
        System.out.println("Quack");
    }
}
```

```
package duck3;
public class Squeak implements QuackBehavior {
    public void quack() {
        System.out.println("Squeak");
    }
}
```

```
package duck3;
public class MuteQuack implements QuackBehavior {
    public void quack() {
        System.out.println("<< Silence >>");
    }
}
```

```
package duck3;
public abstract class Duck {
    FlyBehavior flyBehavior;
    QuackBehavior quackBehavior;

    public Duck() {
    }

    public void setFlyBehavior(FlyBehavior fb) {
        // TODO1
    }

    public void setQuackBehavior(QuackBehavior qb) {
        // TODO2
```

```
    }

    abstract void display();

    public void performFly() {
        // TODO3
    }

    public void performQuack() {
        // TODO4
    }

    public void swim() {
        System.out.println("All ducks float, even decoys!");
    }
}
```

```
package duck3;
public class MallardDuck extends Duck {
    public MallardDuck() {
        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();
    }

    public void display() {
        System.out.println("I'm a real Mallard duck");
    }
}
```

```
package duck3;
public class RedHeadDuck extends Duck {
    public RedHeadDuck() {
        // TODO5
    }

    public void display() {
        System.out.println("I'm a real Red Headed duck");
    }
}
```

```
package duck3;
public class RubberDuck extends Duck {
    public RubberDuck() {
        // TODO6
    }

    public void display() {
        System.out.println("I'm a rubber duckie");
    }
}
```

```
package duck3;
public class DecoyDuck extends Duck {
    public DecoyDuck() {
        // TODO7
    }

    public void display() {
```

```
        System.out.println("I'm a duck Decoy");
    }
}
```

Chạy test

```
package duck3;
public class DuckSimulator {
    public static void main(String[] args) {
        MallardDuck mallard = new MallardDuck();
        RubberDuck rubberDuckie = new RubberDuck();
        DecoyDuck decoy = new DecoyDuck();

        mallard.display();
        mallard.performQuack();
        mallard.performFly();

        rubberDuckie.display();
        rubberDuckie.performQuack();
        rubberDuckie.performFly();

        decoy.display();
        decoy.performQuack();
        decoy.performFly();

        // TODO8
        decoy.performFly();
    }
}
```

Thêm các ứng xử bay và kêu mới, và loại vịt mới. Thay đổi ứng xử ở thời gian chạy

```
package duck3;
public class FlyRocketPowered implements FlyBehavior {
    public void fly() {
        System.out.println("I'm flying with a rocket");
    }
}
```

```
package duck3;
public class FakeQuack implements QuackBehavior {
    public void quack() {
        System.out.println("Qwak");
    }
}
```

```
package duck3;
public class ModelDuck extends Duck {
    public ModelDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Quack();
    }

    public void display() {
        System.out.println("I'm a model duck");
    }
```

```
}
```

```
package duck3;
public class MiniDuckSimulator {
   public static void main(String[] args) {
      ModelDuck model = new ModelDuck();

      model.display();
      model.performFly();
      model.setFlyBehavior(new FlyRocketPowered());
      model.performFly();
   }
}
```

Bài 2: Video rental

Một cửa hàng cho thuê video. Mỗi phim video có các thông tin tựa phim, hãng sản xuất. Mỗi khách hàng được phép thuê tối đa 5 phim. Với mỗi phim thuê, nhân viên sẽ ghi nhận lại ngày bắt đầu thuê.

Tiền thuê phim được tính phụ thuộc vào thời gian thuê phim và được tính như sau:

- Với phim bình thường (REGULAR) mỗi lượt thuê là 3000đ, nhưng nếu giữ phim từ ngày thứ 3 trở đi mỗi ngày tính 1000đ.
- Với phim mới (NEW RELEASE) mỗi ngày thuê tính 4000đ
- Với phim trẻ em (CHILDRENS) mỗi lượt thuê là 2500đ, nhưng nếu giữ phim từ ngày thứ 4 trở đi mỗi ngày tính 1500đ.

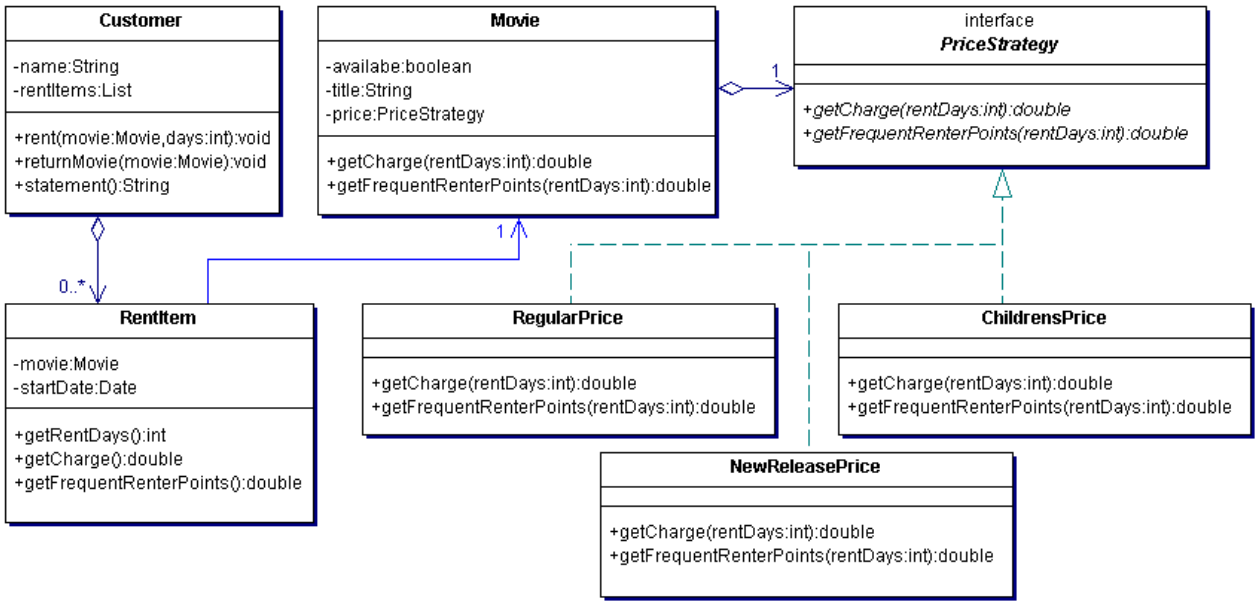
Ngoài ra, cùng với việc tính tiền thuê, cửa hàng cũng tính điểm thưởng cho khách hàng thuê thường xuyên (frequent renter points) như sau:

- Với mỗi phim thuê cộng 1 điểm cho khách hàng, ngoài ra với các phim mới và số ngày thuê của khách hàng > 1 thì cộng thêm 1 điểm nữa.

Cách tính tiền thuê phim và tính điểm thưởng theo các loại phim được thuê như trên có thể thay đổi và có thể có cách cách tính khác.

Yêu cầu:

1) Thiết kế và cài đặt các lớp cho bài toán trên.
2) Thiết kế và cài đặt các xử lý tính toán sau:
   a) Tính tiền thuê phim và tính điểm thưởng cho từng lần thuê phim.
   b) Thêm một cách tính tiền thuê phim mới cho bài toán: Đối với phim bộ (nhiều tập - SERIAL) thì mỗi lượt thuê tính 2000đ, nhưng nếu giữ phim từ ngày thứ 3 trở đi mỗi ngày tính 1000đ.
   c) In biên lai tính tiền thuê cho từng khách hàng.



Customer

```
package videorental;
import java.util.*;
```

```
class Customer {
   private String name;
   private List<RentItem> rentals = new ArrayList<RentItem>();

   public Customer(String name) {
      this.name = name;
   };

   public String getName() {
      return name;
   };

   public List<RentItem> getRentals() {
      return rentals;
   }

   public boolean rent(Movie movie, Date startDate) {
      RentItem rental = new RentItem(movie, startDate);
      if (rentals.size() < 5) {
         rentals.add(rental);
         rental.getMovie().setAvailable(false);
         return true;
      } else
         return false;
   }

   public boolean returnMovie(String title) {
      RentItem r = null;
      for (RentItem rental : rentals) {
         if (rental.getMovie().getTitle().equals(title)) {
            r = rental;
            break;
         }
      }
      if (r != null) {
         rentals.remove(r);
         r.getMovie().setAvailable(true);
         return true;
      } else
         return false;
   }

   public String statement() {
      StringBuffer result = new StringBuffer();
      result.append("Hoa don cua " + getName() + "\n");
      for (RentItem each : rentals) {
         result.append("\t" + each.getMovie().getTitle() + "\t"
               + each.getCharge() + "\n");
      }

      result.append("Tien tra " + getTotalCharge() + "\n");
      result.append("Diem thuong " + getTotalFrequentRenterPoints());
      return result.toString();
   }

   private double getTotalCharge() {
      double result = 0;
      for (RentItem each : rentals) {
         result += each.getCharge();
      }
```

```
        return result;
    }

    private double getTotalFrequentRenterPoints() {
        double result = 0;
        for (RentItem each : rentals) {
            result += each.getFrequentRenterPoints();
        }
        return result;
    }
}
```

Movie

```
package videorental;
public class Movie {
    private String title;
    private PriceStrategy price;
    private boolean available;

    public Movie(String title, PriceStrategy price) {
        this.title = title;
        this.price = price;
        available = true;
    }

    public PriceStrategy getPrice() {
        return price;
    }

    public void setPrice(PriceStrategy arg) {
        price = arg;
    }

    public String getTitle() {
        return title;
    }

    public double getCharge(int daysRented) {
        return price.getCharge(daysRented);
    }

    public int getFrequentRenterPoints(int daysRented) {
        return price.getFrequentRenterPoints(daysRented);
    }

    public boolean isAvailable() {
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }

    public String toString() {
        return title + ", State: "
              + (available ? " available:" : " not available");
    }
}
```

RentItem

```
package videorental;
import java.util.Date;

class RentItem {
   private Movie movie;
   private Date startDate;

   public RentItem(Movie movie, Date startDate) {
      this.movie = movie;
      this.startDate = startDate;
   }

   public int getRentedDays() {
      Date now = new Date();
      long duration = now.getTime() - startDate.getTime();
      long rentedDays = duration / (24 * 3600000);
      if (duration % (24 * 3600000) > 2 * 3600000)
         rentedDays++;
      return (int) rentedDays;
   }

   public Movie getMovie() {
      return movie;
   }

   public double getCharge() {
      return movie.getCharge(getRentedDays());
   }

   public int getFrequentRenterPoints() {
      return movie.getFrequentRenterPoints(getRentedDays());
   }
}
```

PriceStrategy

```
package videorental;
public abstract class PriceStrategy {
   public static final PriceStrategy CHILDRENS = new ChildrensPrice();
   public static final PriceStrategy REGULAR = new RegularPrice();
   public static final PriceStrategy NEW_RELEASE = new NewReleasePrice();

   abstract public double getCharge(int daysRented);

   public int getFrequentRenterPoints(int daysRented) {
      return 1;
   }
}
```

RegularPrice

```
package videorental;
public class RegularPrice extends PriceStrategy {
   public double getCharge(int daysRented) {
      double result = 3000;
      if (daysRented > 2)
         result += (daysRented - 2) * 1000;
      return result;
   }
}
```

NewReleasePrice

```
package videorental;
public class NewReleasePrice extends PriceStrategy {
   public double getCharge(int daysRented) {
      return daysRented * 4000;
   }

   public int getFrequentRenterPoints(int daysRented) {
      if (daysRented > 1)
         return 2;
      else
         return 1;
   }
}
```

ChildrensPrice

```
package videorental;

public class ChildrensPrice extends PriceStrategy {
   public double getCharge(int daysRented) {
      double result = 2500;
      if (daysRented > 3)
         result += (daysRented - 3) * 1500;
      return result;
   }
}
```

TestVideoRental

```
package videorental;

import java.util.Calendar;
import java.util.GregorianCalendar;

import junit.framework.TestCase;

public class TestVideoRental extends TestCase {

   public void testStatement() {

      Movie movies[] = { new Movie("Harry Potter", PriceStrategy.CHILDRENS),
            new Movie("Pretty Woman", PriceStrategy.REGULAR),
            new Movie("Gai nhay", PriceStrategy.NEW_RELEASE),
            new Movie("Nguoi Ha Noi", PriceStrategy.REGULAR),
            new Movie("Lo lem he pho", PriceStrategy.NEW_RELEASE) };

      Customer teo = new Customer("Teo");
      Customer ti = new Customer("Ti");

      teo.rent(movies[0],
            new GregorianCalendar(2008, Calendar.SEPTEMBER, 22).getTime());
      teo.rent(movies[2],
            new GregorianCalendar(2008, Calendar.SEPTEMBER, 23).getTime());
      ti.rent(movies[1],
            new GregorianCalendar(2008, Calendar.SEPTEMBER, 21).getTime());
      ti.rent(movies[3],
            new GregorianCalendar(2008, Calendar.SEPTEMBER, 22).getTime());
      ti.rent(movies[4],
            new GregorianCalendar(2008, Calendar.SEPTEMBER, 24).getTime());
```

```
        System.out.println(teo.statement());
        System.out.println("-----------------");

        System.out.println(ti.statement());
        System.out.println("-----------------");

        teo.returnMovie("Gai nhay");
        System.out.println(teo.statement());
    }
}
```