Lab 1: Hệ thống game mô phỏng SimUDuck

I) DUCK SIMULATOR 1

A> Mô tả:

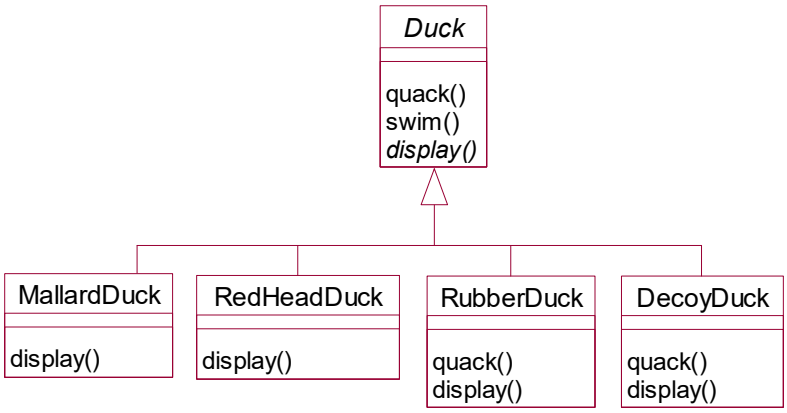
Tạo một hệ thống mô phỏng loài vịt được miêu tả như sau:

- 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

B> Nhiệm vụ

Mở Eclipse, tạo một Java Project có tên **DuckSimulator1**.
Tạo 4 lớp **MarlardDuck**, **RedHeadDuck**, **RubberDuck**, **DecoyDuck** kế thừa từ lớp cha **Duck** và các phương thức sau:



1) Hiển thị **"I am swimming"** khi gọi phương thức **swim()**.
2) Hiển thị **"Quack, quack …"** khi gọi phương thức **quack()** method. Vịt cao su **Rubber** hiển thị **"Squick, squick …"** thay cho **"Quack, quack …"** và vịt mồi **Decoy** không hiển thị gì cả.
3) Hiển thị 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.

```
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();
}
```

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

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

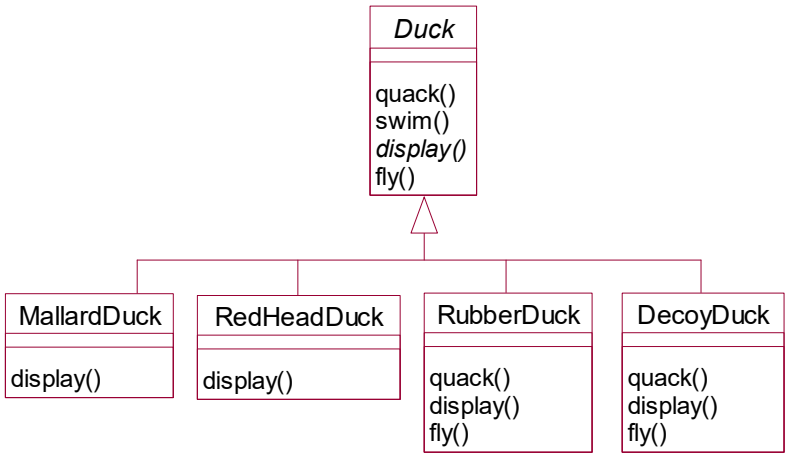
```
public class RubberDuck extends Duck {
   public void quack() {
      System.out.println("Squick, squick ...");
   }

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

```
public class DecoyDuck extends Duck {
   public void quack() {
      System.out.println("...");
   }

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

```
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();
   }
}
```

5) Yêu cầu mới của chương trình: tạo các con vịt có thể bay. **Rubber Duck** và **Decoy Duck** không thể bay được.

- Tạo phương thức **fly()** trong lớp cha **Duck** và in **"I am flying"**
- Cài đặt lại phương thức **fly()** trong **RubberDuck** và **DecoyDuck** không làm gì cả.
- Viết test kiểm tra ứng xử bay của các loại vịt.




```
public abstract class Duck {
   ...

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

```
public class RubberDuck extends Duck {
   ...

   public void fly() { }
}
```

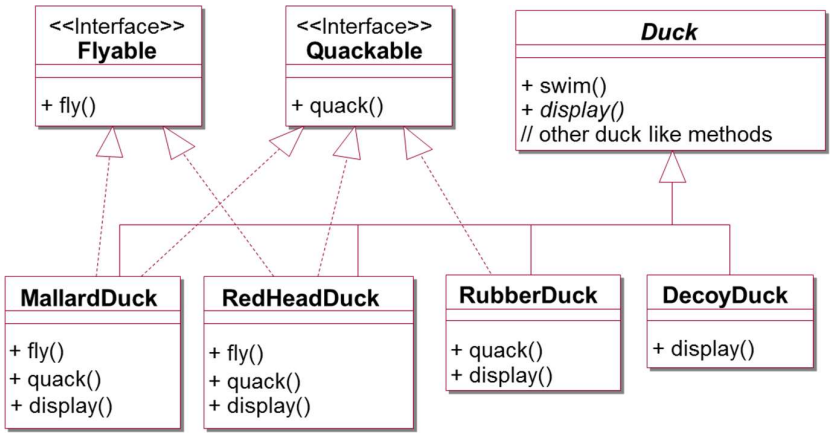
```
public class DecoyDuck extends Duck {
   ...

   public void fly() { }
}
```

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

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

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

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

II> DUCK SIMULATOR 2

A> Mô tả

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



B> Nhiệm vụ:

1) Tạo một project tên **DuckSimulator2**, chép từ project **DuckSimulator1**.
2) Tạo interface **Flyable** có phương thức **fly()** và interface **Quackable** với phương thức **quack()**.
3) Xóa các phương thức **quack**() từ lớp super-class **Duck** (chúng ta không cần nó nữa).
4) Các lớp **MallardDuck**, **RubberDuck** and **RedHeadDuck** bây giờ sẽ cài đặt interface **Quackable**. Các lớp **MallardDuck** and **RedHeadDuck** sẽ cài đặt interface **Flyable**.
5) Chạy lại test.

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

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

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

```
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 ...");
   }
}
```

```
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 ...");
   }
}
```

```
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 ...");
   }
}
```

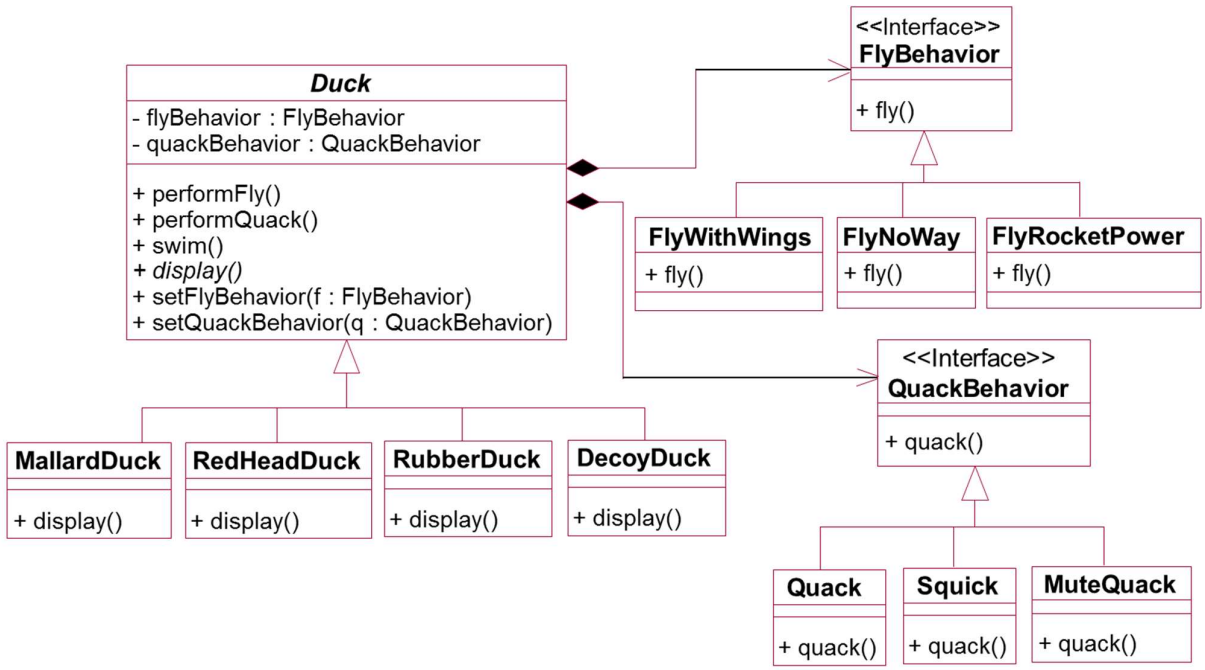
```
public class DecoyDuck extends Duck {
   public void display() {
      System.out.println("I'm a decoy duck");
   }
}
```

```
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();
   }
}
```

C> Nhận xét: Dùng interface cho phép động và dễ dùng. Tuy nhiên trong trường hợp này, code có thể lặp lại ở nhiều nơi, chúng ta mất việc tái sử dụng lại code.

IV> DUCK SIMULATOR 3

A> <u>Mô tả:</u> Thiết kế của hệ thống mô phỏng thế giới loài vịt dùng **Strategy pattern**
Các ứng xử **quack** và **fly** bây giờ được bao bọc trong các đối tượng.



B> <u>Nhiệm vụ</u>

1) Tạo một project tên **DuckSimulator3**, chép từ project **DuckSimulator1**.
2) Tạo 2 interface cho 2 kiểu ứng xử kêu và bay: **FlyBehavior** có phương thức **fly()** và interface **QuackBehavior** với phương thức **quack()**.
3) Tạo các lớp **FlyWithWings** and **FlyNoWay** cài đặt interface **FlyBehavior**.
Tạo các lớp **Quack**, **Squeak** and **MuteQuack** cài đăt interface **QuackBehavior**.
4) Mỗi con vịt bây giờ có một ứng xử bay và một ứng xử kêu, nên ta đặt các đối tượng **flyBehavior** và **quackBehavior** có kiểu tương ứng vào lớp cha **Duck**.
5) Tạo 2 phương thức **performQuack()** và **performFly()** trong lớp cha **Duck**. Các phương thức này kích hoạt **quack()** và **fly()** trong các đối tượng **quackBehavior** và **flyBehavior** tương ứng.
6) Trong contructor của từng kiểu **Duck**, tạo các ứng xử bay và kêu mặc nhiên của chúng.
7) Chạy lại test.
8) Thử thay đổi ứng xử bay của **RubberDuck** ở thời gian chạy bằng cách cho người dùng chọn các chiến lược: **FlyWithWings** hoặc **FlyNoWay**.

C> <u>Nhận xét</u>

Thay vì dùng kế thừa inheritance (quan hệ is-a), chúng ta dùng composition (quan hệ has-a) để tái sử dụng code. Composition cung cấp cho ta nhiều linh động hơn inheritance.

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

```
public interface FlyBehavior {
   public void fly();
}
```

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

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

```java
public interface QuackBehavior {
   public void quack();
}
```

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

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

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

```java
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!");
   }
}
```

```
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");
   }
}
```

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

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

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

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

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

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

Chạy test

```
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

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

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

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

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

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

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

Lab 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:

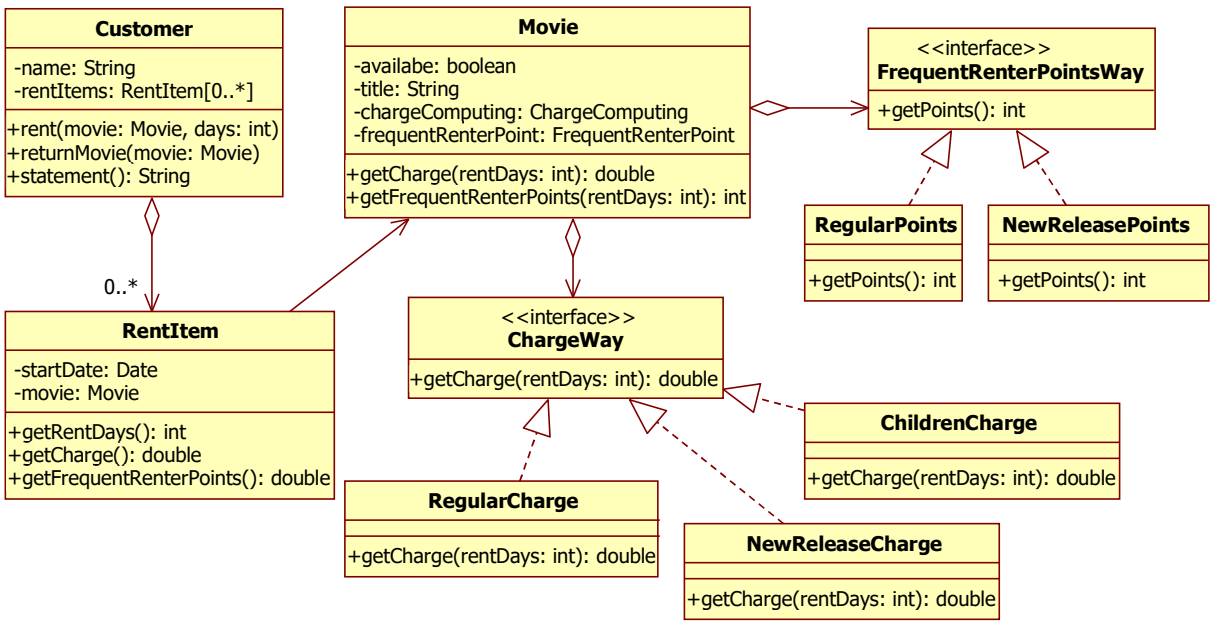
- Với mỗi phim thuê cộng 1 điểm cho khách hàng, ngoài ra với các phim mới và số ngày thuê của khách hàng > 1 thì cộng thêm 1 điểm nữa.

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.

Thiết kế của chương trình dùng mẫu Stategy cho các cách tính tiền thuê phim và tiền thưởng.



Customer

```
public 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

```
public class Movie {
    private String title;
    private ChargeComputing chargeComputing;
    private FrequentRenterPointComputing frequentRenterPointComputing;
    private boolean available;

    public Movie(String title, ChargeComputing price,
                                FrequentRenterPointComputing frequentRenterPoint) {
        this.title = title;
        this.chargeComputing = price;
        this.frequentRenterPointComputing = frequentRenterPoint;
        available = true;
    }

    public ChargeComputing getChargeComputing() {
        return chargeComputing;
    }

    public void setChargeComputing(ChargeComputing arg) {
        chargeComputing = arg;
    }

    public FrequentRenterPointComputing getFrequentRenterPoint() {
        return frequentRenterPointComputing;
    }

    public void setFrequentRenterPoint(FrequentRenterPointComputing frpComputing) {
        this.frequentRenterPointComputing = frpComputing;
    }

    public String getTitle() {
        return title;
    }

    public double getCharge(int rentDays) {
        return chargeComputing.getCharge(rentDays);
    }

    public int getFrequentRenterPoints(int rentDays) {
        return frequentRenterPointComputing.getPoints(rentDays);
    }

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

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

RentItem

```
public 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 rentDays = duration / (24 * 3600000);
        if (duration % (24 * 3600000) > 2 * 3600000) rentDays++;
        return (int) rentDays;
    }

    public Movie getMovie() {
        return movie;
    }

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

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

ChargeWay

```
public interface ChargeWay {
    public double getCharge(int rentDays);
}
```

RegularCharge

```
public class RegularCharge implements ChargeWay {
    public double getCharge(int rentDays) {
        double result = 3000;
        if (rentDays > 2)
            result += (rentDays - 2) * 1000;
        return result;
    }
}
```

NewReleaseCharge

```
public class NewReleaseCharge implements ChargeWay {
    public double getCharge(int rentDays) {
        return rentDays * 4000;
    }
}
```

ChildrensCharge

```
public class ChildrenCharge implements ChargeWay {
   public double getCharge(int rentDays) {
      double result = 2500;
      if (rentDays > 3)
         result += (rentDays - 3) * 1500;
      return result;
   }
}
```

FrequentRenterPointsWay

```
public interface FrequentRenterPointsWay {
   public int getFrequentRenterPoints(int rentDays);
}
```

RegularPoints

```
public class RegularPoints implements FrequentRenterPointsWay {
   public int getFrequentRenterPoints(int rentDays) {
      return 1;
   }
}
```

NewReleasePoints

```
public class NewReleasePoints implements FrequentRenterPointsWay {
   public int getFrequentRenterPoints(int rentDays) {
      if (rentDays > 1)
         return 2;
      else
         return 1;
   }
}
```

RegularMovie

```
public class RegularMovie extends Movie {
   public RegularMovie(String title) {
      super(title);
      setChargeWay(new RegularCharge());
      setFrequentRenterPointsWay(new RegularPoints());
   }
}
```

ChildrenMovie

```
package videorental; public class ChildrenMovie extends Movie {
   public ChildrenMovie(String title) {
      super(title);
      setChargeWay(new ChildrenCharge());
      setFrequentRenterPointsWay(new RegularPoints());
   }
}
```

NewReleaseMovie

```
package videorental;
public class NewReleaseMovie extends Movie {
   public NewReleaseMovie(String title) {
      super(title);
```

```
        setChargeWay(new NewReleaseCharge());
        setFrequentRenterPointsWay(new NewReleasePoints());
    }
}
```

TestVideoRental

```
public class TestVideoRental extends TestCase {

    public void testStatement() {
        Movie m1 = new ChildrenMovie("Harry Potter");
        Movie m2 = new RegularMovie("Pretty Woman");
        Movie m3 = new NewReleaseMovie("Gai nhay");
        Movie m4 = new RegularMovie("Nguoi Ha Noi");
        Movie m5 = new NewReleaseMovie("Lo lem he pho");

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

        teo.rent(m1, new GregorianCalendar(2017, Calendar.MARCH, 2).getTime());
        teo.rent(m3, new GregorianCalendar(2017, Calendar.MARCH, 6).getTime());
        ti.rent(m2, new GregorianCalendar(2017, Calendar.MARCH, 7).getTime());
        ti.rent(m4, new GregorianCalendar(2017, Calendar.MARCH, 7).getTime());
        ti.rent(m5, new GregorianCalendar(2017, Calendar.MARCH, 10).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());
    }
}
```