**The Starting Point**

The sample program is very simple. It is a program to calculate and print a statement of a

customer's charges at a video store. The program is told which movies a customer rented and for

how long. It then calculates the charges, which depend on how long the movie is rented, and

identifies the type movie. There are three kinds of movies: regular, children's, and new releases.

In addition to calculating charges, the statement also computes frequent renter points, which vary

depending on whether the film is a new release.

Several classes represent various video elements. Here's a class diagram to show them (Figure

1.1).

Figure 1.1. Class diagram of the starting-point classes. Only the most important features

are shown. The notation is Unified Modeling Language UML [Fowler, UML].



I'll show the code for each of these classes in turn.

**Movie**

Movie is just a simple data class.

public class Movie {

public static final int CHILDRENS = 2;

public static final int REGULAR = 0;

public static final int NEW\_RELEASE = 1;

private String \_title;

private int \_priceCode;

public Movie(String title, int priceCode) {

\_title = title;

\_priceCode = priceCode;

}

public int getPriceCode() {

return \_priceCode;

}

public void setPriceCode(int arg) {

\_priceCode = arg;

}

public String getTitle (){

return \_title;

};

}

**Rental**

The rental class represents a customer renting a movie.

class Rental {

private Movie \_movie;

private int \_daysRented;

public Rental(Movie movie, int daysRented) {

\_movie = movie;

\_daysRented = daysRented;

}

public int getDaysRented() {

return \_daysRented;

}

public Movie getMovie() {

return \_movie;

}

}

**Customer**

The customer class represents the customer of the store. Like the other classes it has data and

accessors:

class Customer {

private String \_name;

private Vector \_rentals = new Vector();

public Customer (String name){

\_name = name;

};

public void addRental(Rental arg) {

\_rentals.addElement(arg);

}

public String getName (){

return \_name;

};

Customer also has the method that produces a statement. Figure 1.2 shows the interactions for

this method. The body for this method is on the facing page.

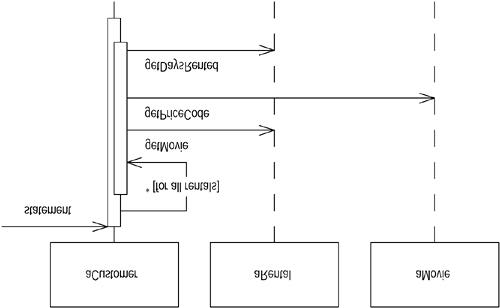


Figure 1.2. Interactions for the statement method

public String statement() {

double totalAmount = 0;

int frequentRenterPoints = 0;

Enumeration rentals = \_rentals.elements();

String result = "Rental Record for " + getName() + "\n";

while (rentals.hasMoreElements()) {

double thisAmount = 0;

Rental each = (Rental) rentals.nextElement();

//determine amounts for each line

switch (each.getMovie().getPriceCode()) {

case Movie.REGULAR:

thisAmount += 2;

if (each.getDaysRented() > 2)

thisAmount += (each.getDaysRented() - 2) \* 1.5;

break;

case Movie.NEW\_RELEASE:

thisAmount += each.getDaysRented() \* 3;

break;

case Movie.CHILDRENS:

thisAmount += 1.5;

if (each.getDaysRented() > 3)

thisAmount += (each.getDaysRented() - 3) \* 1.5;

break;

}

// add frequent renter points

frequentRenterPoints ++;

// add bonus for a two day new release rental

if ((each.getMovie().getPriceCode() == Movie.NEW\_RELEASE)

&&

each.getDaysRented() > 1) frequentRenterPoints ++;

//show figures for this rental

result += "\t" + each.getMovie().getTitle()+ "\t" +

String.valueOf(thisAmount) + "\n";

totalAmount += thisAmount;

}

//add footer lines

result += "Amount owed is " + String.valueOf(totalAmount) +

"\n";

result += "You earned " + String.valueOf(frequentRenterPoints)

+

" frequent renter points";

return result;

}