COMP314-07B Test

11 October 2007

First name: _____ **Last name:** _____

ID Number: _____

Instructions

1. Write your name and ID number into the spaces provided above.
2. There are three questions of equal value.
3. All questions have to be answered.
4. Time allowed is 50 minutes.
5. Write your answers in the spaces provided. Do not use your own paper! There is a blank page at the end of the test that you can use. You can get extra paper from us if necessary.

DO NOT TURN THE PAGE UNTIL YOU ARE ASKED TO DO SO!

Question 1: A Java Generics exercise

Immutable objects (objects that cannot be modified after creation, e.g. Strings in Java) can be very useful, especially with respect to program correctness as well as efficiency in multi-threaded processing: as they only allow read-access, they cannot be accidentally modified, and access can be performed in parallel.

Fill out missing bits below for an immutable `Map` implementation, which is based on a HashMap by way of Composition (also an instance of the Adapter pattern). You need to fill in proper types (use generic types where appropriate) into the marked spaces, as well as appropriate code for all three methods *size*, *get*, and *put*.

```
public final class ImmutableMap<K,V> implements Map<K,V> {

   HashMap<K,V> mapping;

   //
   // constructor which initialises from a given map of appropriate type
   //
   public ImmutableMap<K,V>( Map<? extends K, ? extends V> someMap ) {
       mapping = new HashMap( someMap);
   }

   //
   // return the size of this map.
   //
   public ____ size() {

      ......
   }

   //
   // retrieve the value associated with the given key
   //
   public ____ get( ____ key) {

      ......
   }

   //
   // update the value associated with the given key
   //
   public ____ put( ____ key, ____ value) {

       ......
   }
}
```

Question 2: Refactoring exercise

Assume you are given the following class definitions for a software system managing a video rental business. Inspect the *computeStatement*() method of the *Customer* class overleaf and suggest refactorings to improve the design and code of this method.

```
public class Movie {

  public String title = "";
  public int priceCode = 0;

  public Movie(String title, int priceCode) {
     this.title = title;
     this.priceCode = priceCode;
  }
}

public class Rental {

  public Movie movie = null;
  public int daysRented = 0;

  public Rental(Movie movie, int daysRented) {
      this.movie = movie;
      this.daysRented = daysRented;
  }
}

public class Customer {

  public ArrayList m_Rentals = new ArrayList();
  public String m_Name = "";

  public Customer(String name) {
     m_Name = name;
  }

  // please turn over ...
```

```
public String computeStatement() {
  double totalAmount = 0;
  // frequent renter points, used for loyalty scheme members
  int frp = 0;
  Iterator rentals = m_Rentals.iterator();
  String result = "Rental record for " + m_Name + "\n";

  while ( rentals.hasNext() ) {
    double thisAmount = 0;
    Rental each = (Rental) rentals.next();

    switch(each.movie.priceCode) {
    case 1:                                  // regular rental, good for 8 days
        thisAmount += 5;
        if (each.daysRented > 8) thisAmount += (each.daysRented - 8) * 1.0;
        break;

    case 2:                                  // new release, 8 bucks per day
        thisAmount += each.daysRented * 8;
        break;

    case 3:                                  // recent release, good for 3 days
        thisAmount += 6.0;
        if (each.daysRented > 3) thisAmount += (each.daysRented - 3) * 0.5;
        break;
    }

    frp++;

    // Add bonus point for a two-day new-release rental
    if ((each.movie.priceCode == 2) && (each.daysRented > 1)) {
      frp++;
    }

    // Show figures for this rental
    result += "\t" + each.movie.title + "\t" + thisAmount + "\n";
    totalAmount += thisAmount;
  }

  // Add footer lines
  result += "Amount owed is " + totalAmount + "\n";
  result += "You earned " + frp + " frequent renter points.";
  return result;
}
```

Question 3: Design Patterns

Explain the usefulness of design patterns for software engineering in general:

Also explain which patterns you are using in your project and how. Should you currently not use any pattern in your project, speculate instead on which patterns you could have used, or might use in future versions:

Extra space for answering questions