The University of British Columbia

CPSC 210

Sample Midterm Examination

This examination has 10 pages.

Check that you have a complete paper.

Ensure you have access to the source repository at:
svn+ssh://westham.ugrad.cs.ubc.ca/home/c/cs210/repositorectures/lectureMidTermOne-2009W2

We refer to the code in this repository as `PaymentSystem` in this exam.

This is an open computer examination. You may use any reference on the internet but you must not collaborate with any other individual through personal contact or electronic contact in anyway during the exam.

Answer all the questions on this paper.

Give **short but precise** answers.

Work fast and do the easy questions first. Leave some time to review your exam at the end.

The marks for each question are given in []. Use this to manage your time.

Good Luck

Question	**Marks**
1	/6
2	/6
3	/4
4	/4
5	/4
6	/6
7	omitted
8	/6
9	/6
10	omitted
11	/4

Questions 1 through 10 apply to the `PaymentSystem` provided in the specified repository.

Question 1. Type Hierarchy [6 points]

Draw the type hierarchy for all types declared in the `ca.ubc.cpsc210.payment.model` package. Use directional arrows to relate subtypes to supertypes in the drawing (i.e., lines between types should have an arrowhead only at one end; lines should go from the subtype to the supertype with the arrowhead at the supertype).

Question 2. Call Graph [6 points]

Draw a call graph starting from the `generateCreditCardPayments(AuditTrail auditTrail)` function defined in the `Main` class (Main.java). Stop following method calls for any method defined in a class outside of `ca.ubc.cpsc210.payment.model`. You might want to sketch the call graph on a scrap piece of paper before placing it on this sheet. You can also rotate the paper and write in lansdscape mode for more space. If you abbreviate any names, please provide a legend.

Question 3. Types. [4 points]

Consider the following code:

```
(1) Payment p;
(2) p = new DebitCard(3, 4);
(3) InternetPayment i = new PalPay();
```

i) What is the actual type of the variable `p` at the statement numbered (2) after the statement executes?

ii) What is the apparent type of the variable `p` at the statement numbered (2) after the statement executes?

iii) What is the apparent type of the variable `i` at the statement numbered (3) after the statement executes?

iv) What is the actual type of the variable `i` at the statement numbered (3) after the statement executes?

Question 4. Overridding and Overloading. [4 points]

i) Consider the following code:

```
(1) CreditCard c = new ASIVCard(3, "Me", "02/10");
(2) PaymentRecord record = c.processPayment(4.0);
```

State the name of the class and the full method signature of the `processPayment` method executed at statement (2).

ii) Consider the following code:

```
(3) CreditCard c = new ASIVCard(3, "Me", "02/10");
(4) PaymentRecord record = c.processPayment(new String(4.0));
```

State the name of the class and the full method signature of the `processPayment` method executed at statement (4).

Question 5. Exceptions. [4 points]

In the function `generateInternetPayments(AuditTrail auditTrail)` in the `Main` class (Main.java), the last statement of the `for` loop is

```
URL url = internetPayment.retrieveURLForProvider();
```

Assuming that execution of this statement throws a `MalformedURLException`, describe what code will execute until the system terminates.

Question 6. Unit Tests [6 points]

Consider the specification for the `processPayment(double amount)` operation in the `Payment` interface. Describe a good set of unit cases for testing this operation using a black-box testing approach. For each test case, provide the input and the expected output.

Input	Output

Question 7. Omitted

Question 8. Specifications [6 points].

For each of the public constructors and methods listed on the `PaymentRecord` and `AuditTrail` classes below, indicate whether the constructor or method is an observer (mark with O) or a mutator (mark with M).

Constructors and Methods on PaymentRecord	M or O?	Constructors and Methods on AuditTrail	M or O?
PaymentRecord(String, double)		AuditTrail()	
PaymentRecord(String, double, int)		void addPaymentRecord(PaymentRecord)	
PaymentRecord()		Iterator<PaymentRecord> iterator()	
boolean inError()			
Date getTimeOfPayment()			
String getTypeOfPayment()			
double getAmountOfPayment()			
int getPaymentNumber()			
int getTransactionNumber()			
String toString()			

Question 9. Debugging. [6 points]

If you run the Main class as a Java application, the output will include the following:

```
Payment[ num=15, type=PalPay, amt=0.724302501394058, txNum=15]
Payment[ num=16, type=PalPay, amt=1.2554252514453499, txNum=16]
Payment[ num=-83, type=Cash, amt=0.0]
Payment[ num=-82, type=Cash, amt=0.3682269387159234]
```

Note that the last two lines of this output have a negative payment number, which is illegal according to the specification of the PaymentRecord data abstraction. Generate two hypotheses about what might be causing this error.

(Extra credit.) What is actually causing the error in the output shown above? (3 points)

Question 10. Omitted

Question 11. Type Substitution [6 points]

i) Consider the following code. Is an object of type `SmallFigure` substitutable for an object of the supertype `Figure`? Explain why or why not.

```
class Figure{
   ...
   // REQUIRES: width and height are positive integers
   // MODIFIES: Graphics object
   // EFFECTS:  returns a Graphics object that has the figure drawn
   //           onto it
   public Graphics drawFigure(int width, int height) {...}
   …
}



class SmallFigure extends Figure{
   ...
   // REQUIRES: 0 < width < 100 and 0 < height < 250
   // MODIFIES: Graphics object
   // EFFECTS: returns a Graphics object that has the small figure
   //          drawn onto it
   public Graphics drawFigure(int width, int height) {...}
   …
}
```

ii) Consider the following code. Is an object of type `FlatPanelTVHighRes` substitutable for an object of the supertype `FlatPanelTVLowRes`? Explain why or why not.

```
class FlatPanelTVLowRes {

     private int horizontalRes;
     private int verticalRes;

     /**
      * Return the resolution in shorthand integer form
      * REQUIRES: nothing
      * MODIFIES: nothing
      * EFFECTS: returns a value >= 720
      */
     void getResolution() {...}
     . . .
}

class FlatPanelTVHighRes extends FlatPanelTVLowRes {

     /**
      * REQUIRES: nothing
      * MODIFIES: nothing
      * EFFECTS: returns a value >= 1080
      */
     int getResolution() {...}
     . . .
}
```