CPSC 210

Sample Midterm Exam Questions

The source code for use with these sample questions can be found at in the lectures repository at: `/SampleMidterm1`

Checkout the project named `PaymentSystem`.

This question set does not constitute an actual exam although many of these questions have appeared on past exams. The set is representative of the kinds of questions that could be asked on your midterm exam but the questions included here are by no means exhaustive.

During your exam, you will be allowed to access the Java 6 API and to reference any code that has been committed to the /lectures repository.

Questions 1 through 5 apply to the `PaymentSystem` provided in the specified repository.

Question 1. Type Hierarchy

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

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.

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.

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. Debugging.

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?

Question 6. Specification

Suppose you are designing a new data type to represent a fare box on a bus. The fare box accepts pre-paid tickets and cash (in the form of coins only). When a ticket is inserted into the machine, the value of the ticket is read and that amount is added to the total fare collected. The amount of the fare is deducted from the ticket. When coins are inserted, their value is added to the total fare collected. Write the specification for the `payByTicket` and `payByCash` methods:

```
public class FareBox {
    private int totalFareCollected;   // in cents

    public void payByTicket(Ticket t) {
        …
    }

    public void payByCash(int coinValue) {
        …
    }
}
```

Question 7. Unit Tests

Consider the specification for the `reportWeather` method shown below. Describe a good set of unit cases for testing this operation using a white-box testing approach. For each test case, provide the input and the expected output.

```
// Get a weather report
// EFFECTS:  returns a brief weather report based on the
//           given temperature.
public String reportWeather(int temperature) {
   String report;

   if (temperature < 0) {
      report = "Chilly…";
   }
   else if (temperature < 25) {
      report = "Pleasant…";
   }
   else {
      report = "Hot…";
   }

   return report;
}
```

Input	Output

Question 8. Type Substitution

i) Consider the following code. Examine the specification of the methods in each class, in the context of the Liskov Substitution Principle, to determine if an object of type `EvenSet` is substitutable for an object of the supertype `Set`. Explain why or why not.

```
// represents a set of integers
public class IntSet {

    // Requires: nothing
    // Modifies: this
    // Effects: adds Integer i to the set, if it's not already there
    public void insert(Integer i) {
       //…
    }
}


// represents a set that contains even integers
public class EvenSet extends IntSet {

    // Requires: nothing
    // Modifies: this
    // Effects: adds Integer i to the set, if it's not already there
    //          and if i is an even integer, otherwise does nothing
    public void insert(Integer i) {
       //…
    }
}
```

iii) Do the same to the following classes to determine if `SortedIntSet` is substitutable for an object of the supertype `IntSet`.

```
// represents a set of integers
public class IntSet {

    // Requires: nothing
    // Modifies: this
    // Effects: adds Integer i to the set, if it's not already there
    public void insert(Integer i) {
       //…
    }
}


// represents a sorted set of integers
public class SortedIntSet extends IntSet {

    // Requires: nothing
    // Modifies: this
    // Effects: adds Integer i to the set, if it's not already
    //          there, so that items in set remain sorted in
    //          ascending order
    public void insert(Integer i) {
       //…
    }
}
```

iii) Do the same to the following classes to determine if `SpecialCurrentAccount` is substitutable for an object of the supertype `CurrentAccount`. (This example is inspired by an article by Samudra Gupta (http://javaboutique.internet.com/tutorials/JavaOO/):

```
public class CurrentAccount {

    //Requires: nothing
    //Modifies: nothing
    //Effects:  returns true if getBalance() >= 0
    //          false otherwise
    public boolean closeAccount() {
        //…
    }

    //Requires: nothing
    //Modifies: nothing
    //Effects:  returns balance on the account (in cents)
    public int getBalance() {
        //…
    }

    //Requires: nothing
    //Modifies: nothing
    //Effects: returns number of days that account has been open
    public int period() {
        //…
    }
}


public class SpecialCurrentAccount extends CurrentAccount {

    //Requires: period() > 60
    //Modifies: nothing
    //Effects:  returns true if getBalance() >= 0,
    //          false otherwise
    public boolean closeAccount() {
        //…
    }
}
```

iv) What if the specification for `SpecialCurrentAccount.closeAccount` is as follows:

```
public class SpecialCurrentAccount extends CurrentAccount {

    //Requires: nothing
    //Modifies: nothing
    //Effects:  returns true if getBalance() >= 0 and
    //          period() > 60, false otherwise
    public boolean closeAccount() {
        //…
    }
}
```

Is an object of type `SpecialCurrentAccount` substitutable for an object of type `CurrentAccount`?