The University of British Columbia

CPSC 210

Quiz Practice Question Solutions

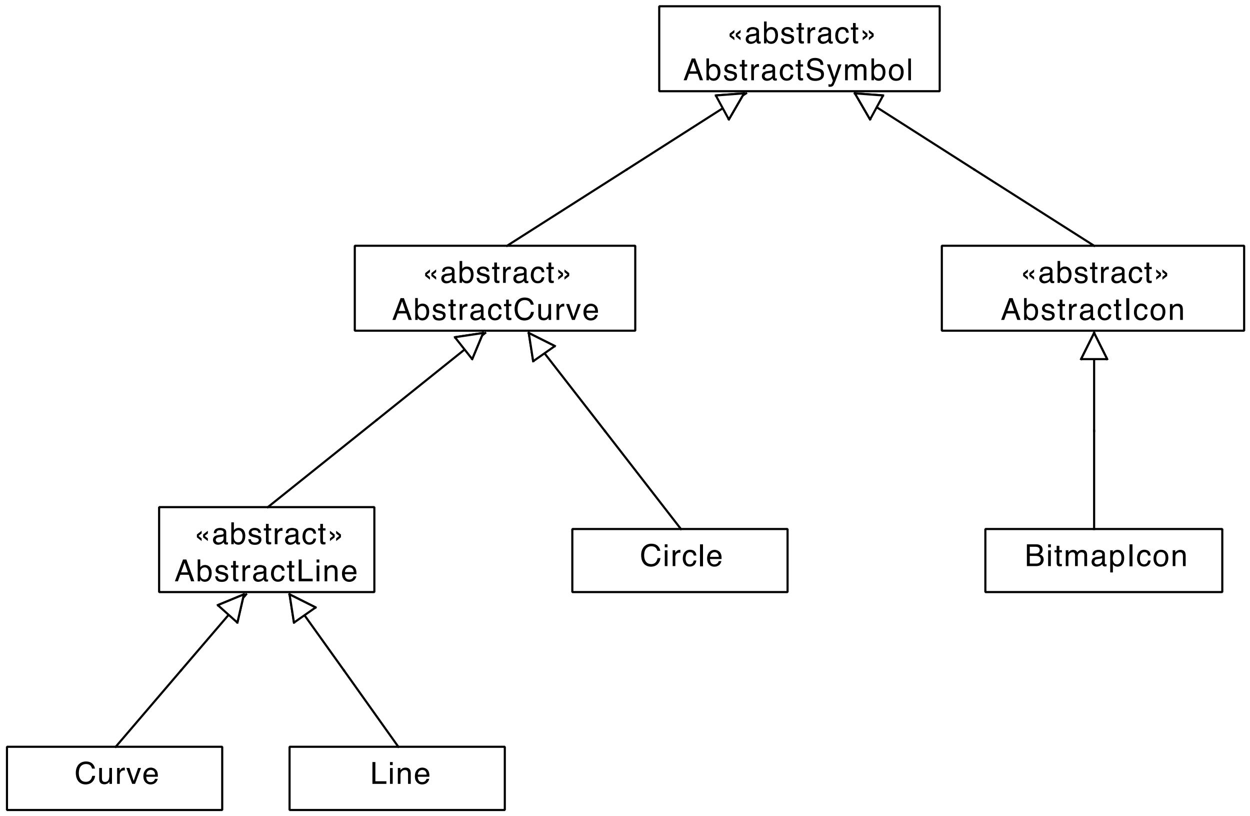
Please don't look at these solutions until you have put significant effort into constructing your own.

Name Student No

IMPORTANT: Questions 1 to 3 apply to the `JDrawing` system provided in the specified repository.

Question 1. Type Hierarchy

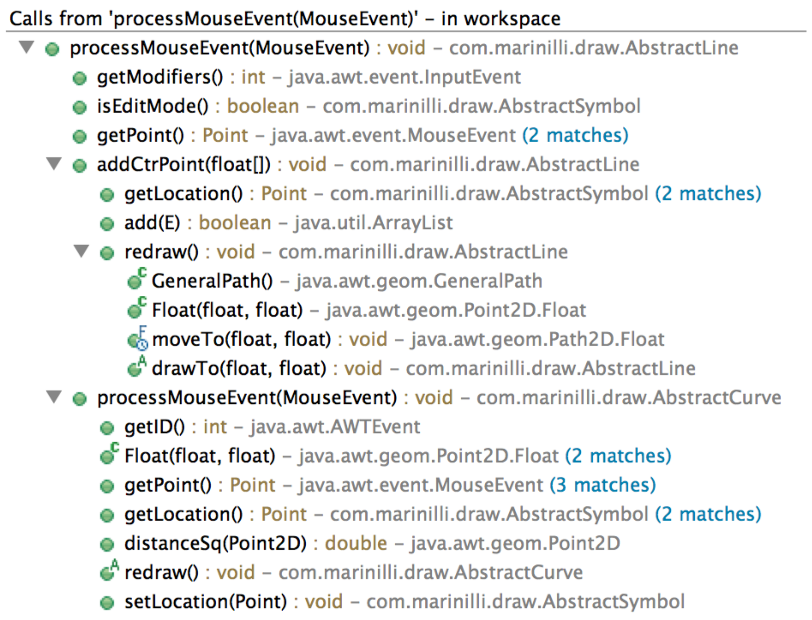
Draw a type hierarchy that includes all subtypes of AbstractSymbol declared in the `com.marinilli.draw` package. Do not include any class(es) or interface(es) declared in the Java library.



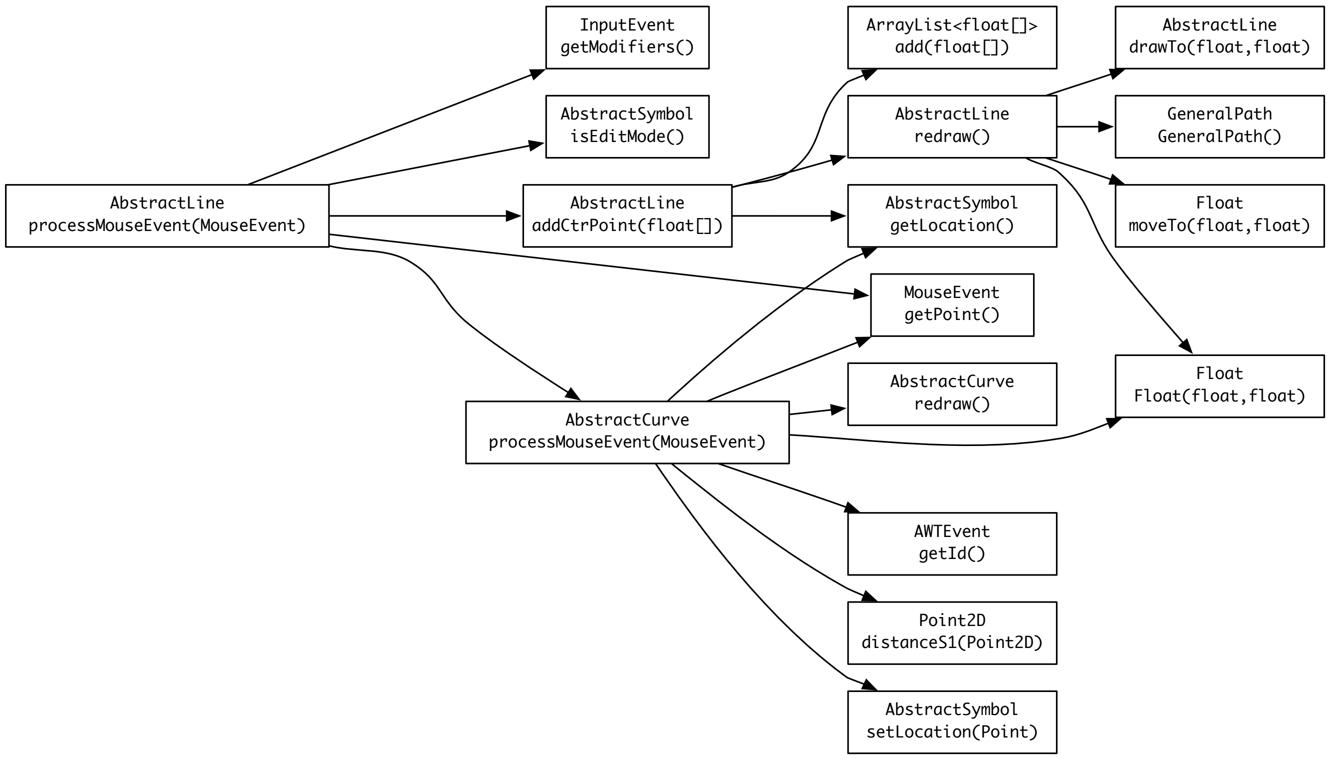
Name Student No

Question 2. Inter-method Control Flow (Call Graph)

Draw a call graph starting from the `processMouseEvent(MouseEvent me)` method defined in the `AbstractLine` class of the `com.marinilli.draw` package. Do not include calls to methods in any Java library. If you abbreviate any names, please provide a legend. You might want to rotate the page and draw your graph in landscape mode.



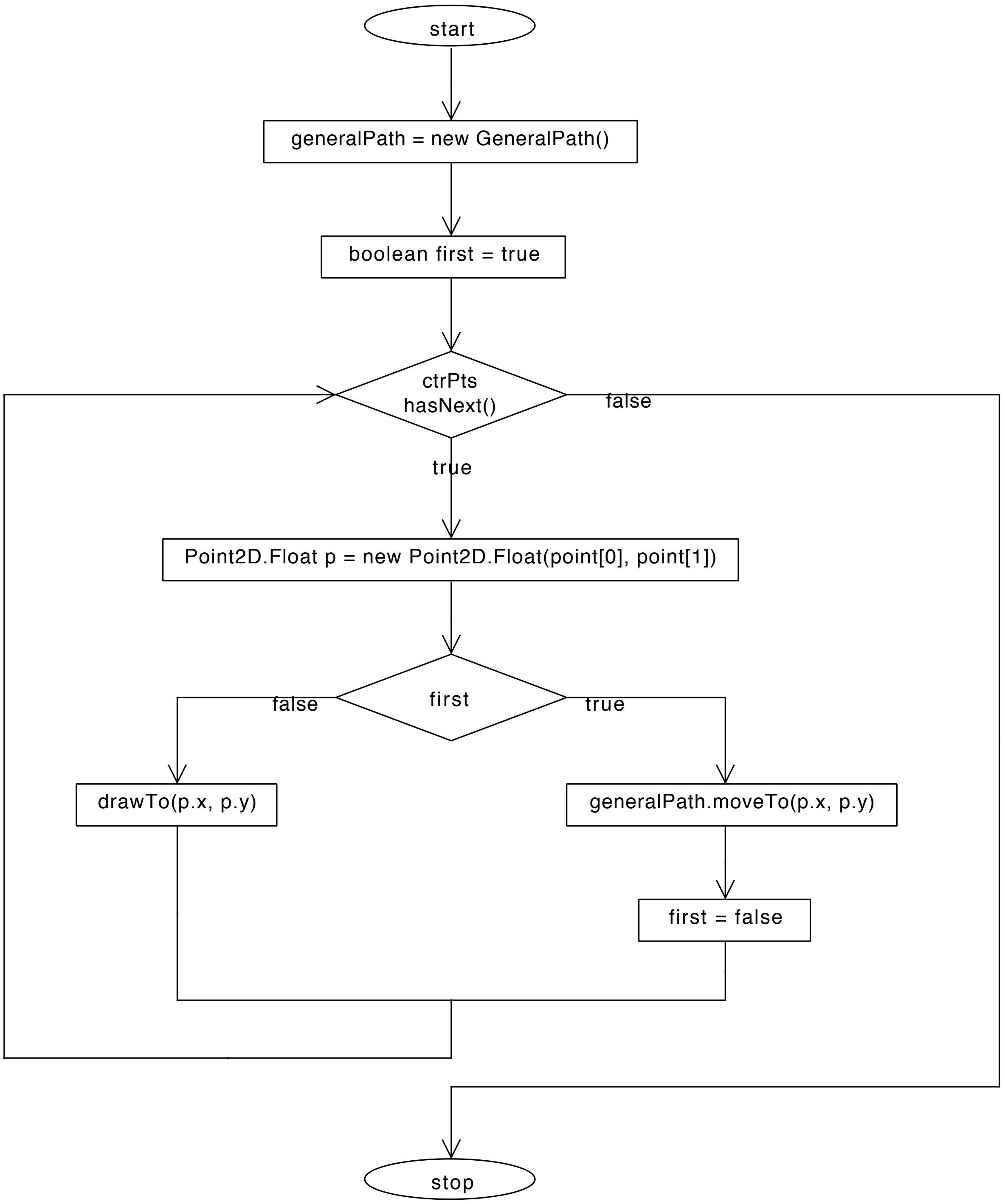
```
Calls from 'processMouseEvent(MouseEvent)' – in workspace
▼ ● processMouseEvent(MouseEvent) : void – com.marinilli.draw.AbstractLine
    ● getModifiers() : int – java.awt.event.InputEvent
    ● isEditMode() : boolean – com.marinilli.draw.AbstractSymbol
    ● getPoint() : Point – java.awt.event.MouseEvent (2 matches)
  ▼ ● addCtrPoint(float[]) : void – com.marinilli.draw.AbstractLine
      ● getLocation() : Point – com.marinilli.draw.AbstractSymbol (2 matches)
      ● add(E) : boolean – java.util.ArrayList
    ▼ ● redraw() : void – com.marinilli.draw.AbstractLine
        ● GeneralPath() – java.awt.geom.GeneralPath
        ● Float(float, float) – java.awt.geom.Point2D.Float
        ● moveTo(float, float) : void – java.awt.geom.Path2D.Float
        ● drawTo(float, float) : void – com.marinilli.draw.AbstractLine
  ▼ ● processMouseEvent(MouseEvent) : void – com.marinilli.draw.AbstractCurve
      ● getID() : int – java.awt.AWTEvent
      ● Float(float, float) – java.awt.geom.Point2D.Float (2 matches)
      ● getPoint() : Point – java.awt.event.MouseEvent (3 matches)
      ● getLocation() : Point – com.marinilli.draw.AbstractSymbol (2 matches)
      ● distanceSq(Point2D) : double – java.awt.geom.Point2D
      ● redraw() : void – com.marinilli.draw.AbstractCurve
      ● setLocation(Point) : void – com.marinilli.draw.AbstractSymbol
```



AbstractLine processMouseEvent(MouseEvent)
InputEvent getModifiers()
AbstractSymbol isEditMode()
AbstractLine addCtrPoint(float[])
AbstractCurve processMouseEvent(MouseEvent)
ArrayList<float[]> add(float[])
AbstractLine redraw()
AbstractSymbol getLocation()
MouseEvent getPoint()
AbstractCurve redraw()
AWTEvent getId()
Point2D distanceS1(Point2D)
AbstractSymbol setLocation(Point)
AbstractLine drawTo(float,float)
GeneralPath GeneralPath()
Float moveTo(float,float)
Float Float(float,float)

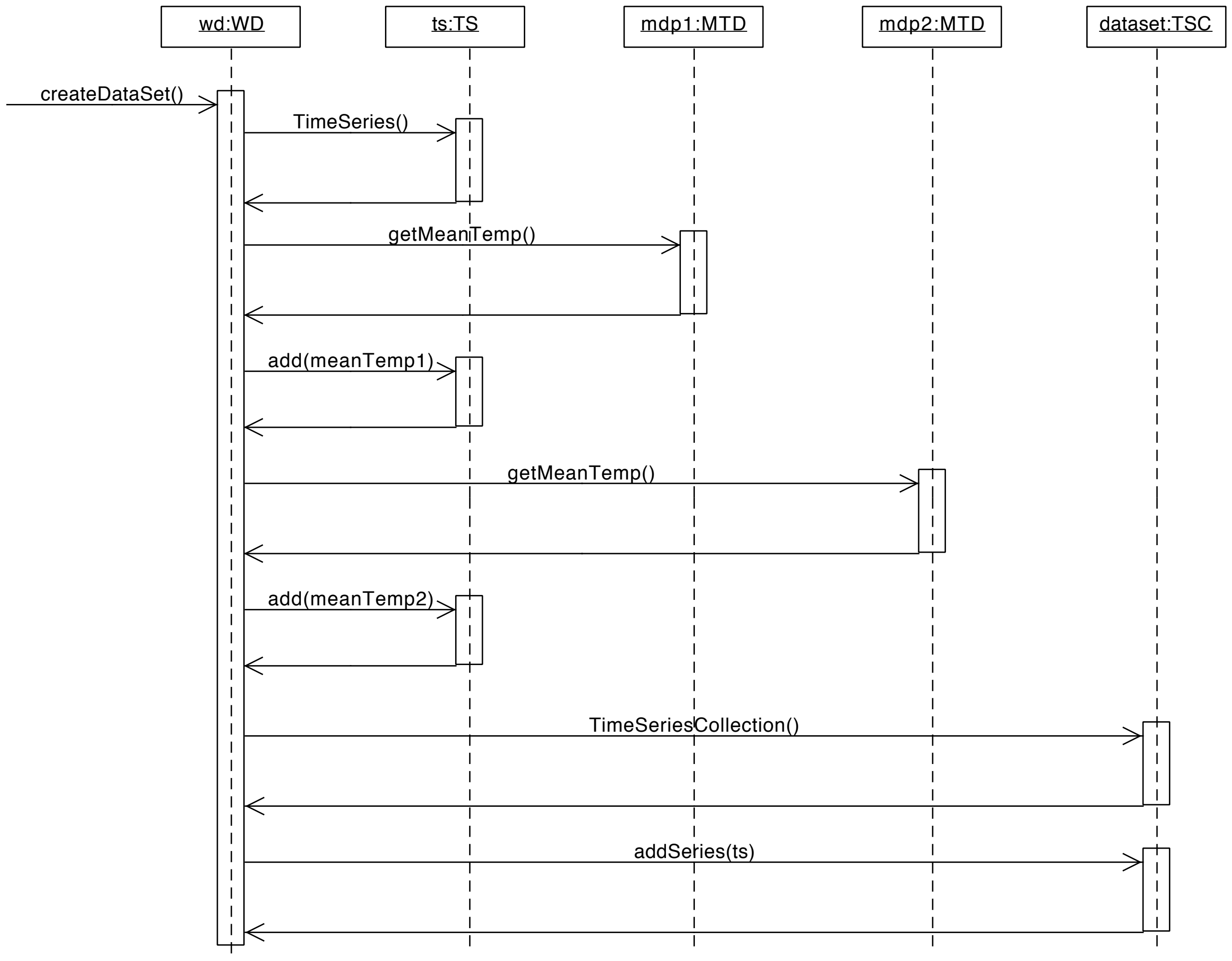
Question 3. Intra-method Control Flow (Flowchart)

Draw a flowchart for the `redraw()` method defined in the `AbstractLine` class of the `com.marinilli.draw` package.


start
generalPath = new GeneralPath()
boolean first = true
ctrPts hasNext()
false
true
Point2D.Float p = new Point2D.Float(point[0], point[1])
first
false
true
drawTo(p.x, p.y)
generalPath.moveTo(p.x, p.y)
first = false
stop

Question 4. UML Sequence Diagram

Consider the UML Sequence Diagram given below. **Sketch** the code implementing the method `createDataSet()` on a `WeatherData` object that is described by this diagram. You may need to assume where there are likely return values from calls to methods. Handle these return values in the code you sketch. You can add comments to explain your code. We will **not** be grading for correct Java syntax.



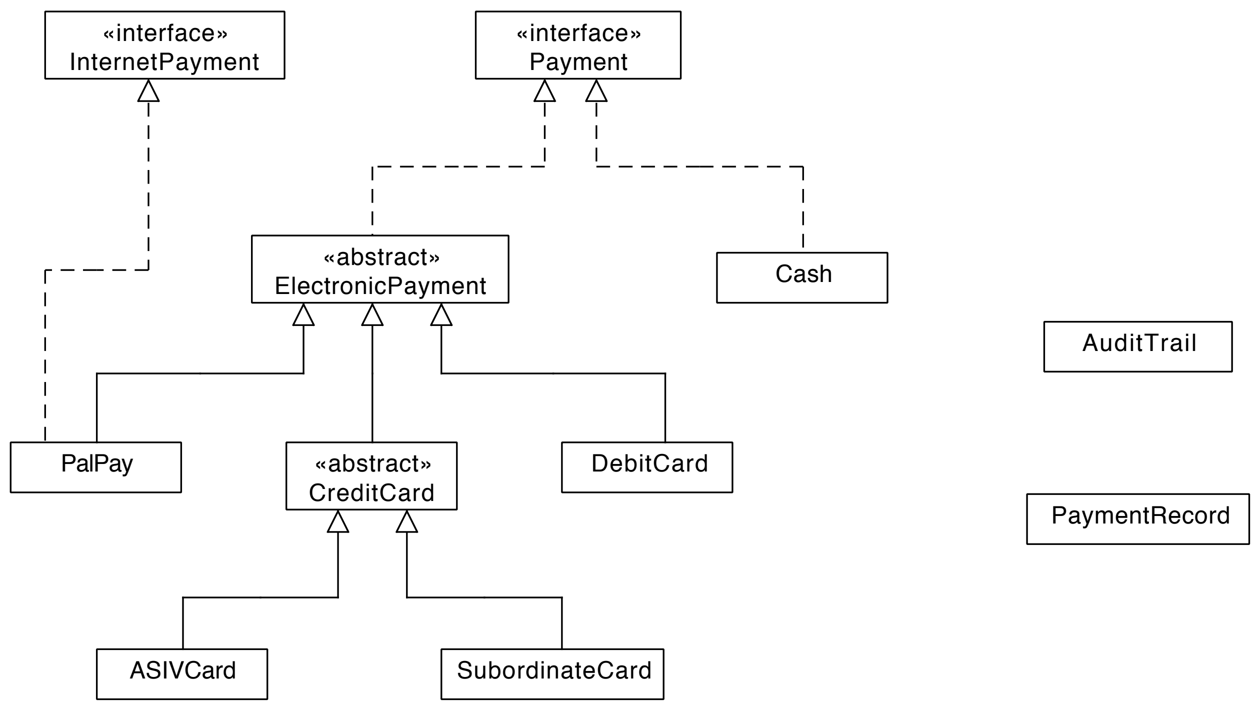
LEGEND
WD: WeatherData
TS: TimeSeries
MTD: MonthlyTemperatureData
TSC: TimeSeriesCollection

```
createDataSet() {
     ts = new TimeSeries();
     meanTemp1 = mdp1.getMeanTemp();
     ts.add(meanTemp1);
     meanTemp2 = mdp2.getMeanTemp();
     ts.add(meanTemp2);
     dataset = new TimeSeriesCollection();
     dataset.addSeries(ts);
}
```

Questions 5 through 8apply to the `PaymentSystem` provided in the specified repository.

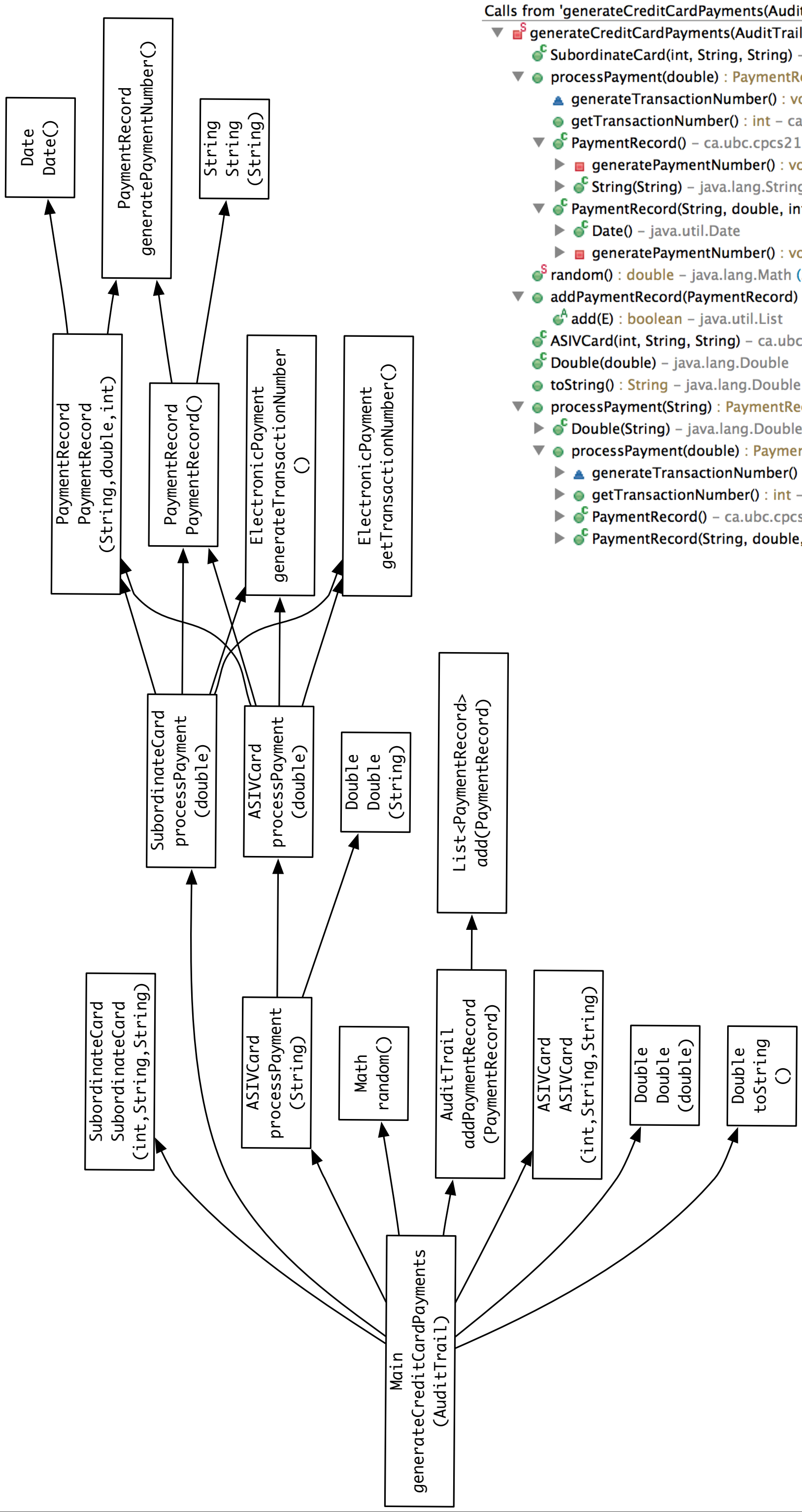
Question 5. Type Hierarchy

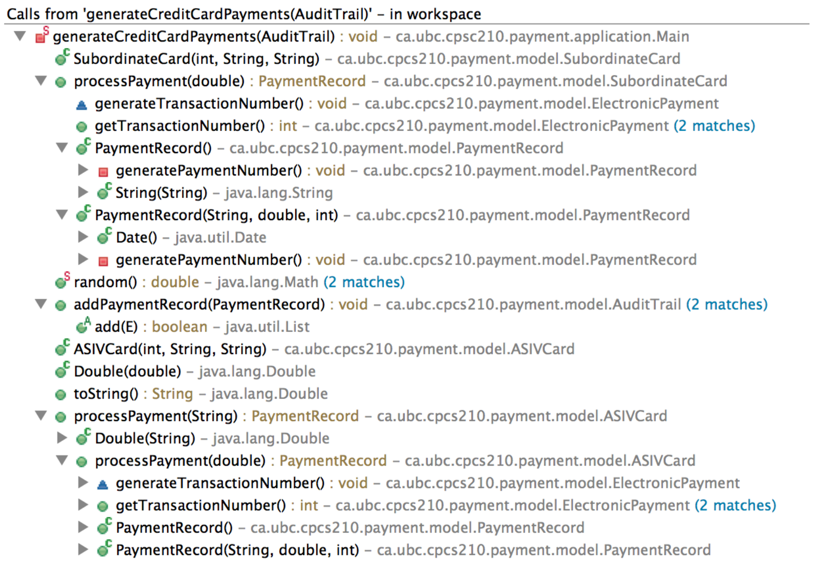
Draw the type hierarchy for all types declared in the `ca.ubc.cpsc210.payment.model` package. Use directional arrows to relate subtypes to supertypes in the drawing (i.e., lines between types should have an arrowhead only at one end; lines should go from the subtype to the supertype with the arrowhead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



Question 6. 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`.


Date Date()
PaymentRecord generatePaymentNumber()
String String (String)
PaymentRecord PaymentRecord (String,double,int)
PaymentRecord PaymentRecord()
ElectronicPayment generateTransactionNumber ()
ElectronicPayment getTransactionNumber()
SubordinateCard processPayment (double)
ASIVCard processPayment (double)
Double Double (String)
List<PaymentRecord> add(PaymentRecord)
SubordinateCard SubordinateCard (int,String,String)
ASIVCard processPayment (String)
Math random()
AuditTrail addPaymentRecord (PaymentRecord)
ASIVCard ASIVCard (int,String,String)
Double Double (double)
Double toString ()
Main generateCreditCardPayments (AuditTrail)


Calls from 'generateCreditCardPayments(AuditTrail)' – in workspace

```
▼ generateCreditCardPayments(AuditTrail) : void – ca.ubc.cpsc210.payment.application.Main
    SubordinateCard(int, String, String) – ca.ubc.cpcs210.payment.model.SubordinateCard
  ▼ processPayment(double) : PaymentRecord – ca.ubc.cpcs210.payment.model.SubordinateCard
      generateTransactionNumber() : void – ca.ubc.cpcs210.payment.model.ElectronicPayment
      getTransactionNumber() : int – ca.ubc.cpcs210.payment.model.ElectronicPayment (2 matches)
    ▼ PaymentRecord() – ca.ubc.cpcs210.payment.model.PaymentRecord
      ▶ generatePaymentNumber() : void – ca.ubc.cpcs210.payment.model.PaymentRecord
      ▶ String(String) – java.lang.String
    ▼ PaymentRecord(String, double, int) – ca.ubc.cpcs210.payment.model.PaymentRecord
      ▶ Date() – java.util.Date
      ▶ generatePaymentNumber() : void – ca.ubc.cpcs210.payment.model.PaymentRecord
    random() : double – java.lang.Math (2 matches)
  ▼ addPaymentRecord(PaymentRecord) : void – ca.ubc.cpcs210.payment.model.AuditTrail (2 matches)
      add(E) : boolean – java.util.List
    ASIVCard(int, String, String) – ca.ubc.cpcs210.payment.model.ASIVCard
    Double(double) – java.lang.Double
    toString() : String – java.lang.Double
  ▼ processPayment(String) : PaymentRecord – ca.ubc.cpcs210.payment.model.ASIVCard
    ▶ Double(String) – java.lang.Double
    ▼ processPayment(double) : PaymentRecord – ca.ubc.cpcs210.payment.model.ASIVCard
      ▶ generateTransactionNumber() : void – ca.ubc.cpcs210.payment.model.ElectronicPayment
      ▶ getTransactionNumber() : int – ca.ubc.cpcs210.payment.model.ElectronicPayment (2 matches)
      ▶ PaymentRecord() – ca.ubc.cpcs210.payment.model.PaymentRecord
      ▶ PaymentRecord(String, double, int) – ca.ubc.cpcs210.payment.model.PaymentRecord
```

Question 7. 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?

`DebitCard`

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

`Payment`

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

`InternetPayment`

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

`PalPay`

Question 8. 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

    // Pay fare by ticket
    // Modifies: this, t
    // Effects: value of ticket is added to total fare collected
    //          and value is deducted from t
    public void payByTicket(Ticket t) {
        …
    }

    // Pay fare by cash
    // Modifies: this
    // Effects: value of coins is added to total fare collected
    public void payByCash(int coinValue) {
        …
    }
}
```

Note: it would not be unreasonable to put a requires clause on `payByTicket` to indicate that the ticket is valid - in other words, we haven't inserted a ticket that's already had the fare value deducted from it.

Question 9. Data Abstraction [21 marks]: The `SimGame` project contains the partial specification and implementation for a `SimPet` class, along with associated unit tests. The `SimPet` represents a pet in a simulated world. Each pet has a location in the two-dimensional world and an energy level. A pet can be pointing in one of only four directions: North, South, East or West. We assume that the pet's location is specified using integer coordinates. In this question, we do not concern ourselves with the size of the world – so we don't worry about pets walking off the edge.

We want to be able to feed the pet and specify the number of units of energy it eats, assumed to be an integer value. We also want to be able to move the pet one unit in whatever direction it is currently pointing. *Each time the pet moves, it consumes one unit of energy*. We also want to be able to rotate the pet left or right by 90 degrees so that it can move in different directions. When a pet rotates, it does not consume any energy. If the pet's energy level drops to zero, it dies.

In this question, you can write your code in Eclipse but you **must** copy it on to this exam paper **before** the end of the exam – there is no electronic submission! Note that it is not necessary to copy the comment statements.

a) Write the implementation of the `SimPet` constructor. Run the JUnit tests provided in `ca.ubc.cpsc210.simgame.test.TestSimPet` and ensure that `testConstructor` passes.

```
public SimPet(int x, int y, int initialEnergy) {
     this.x = x;
     this.y = y;
     this.energy = initialEnergy;
     this.direction = 0;
     this.hasHadShots = false;
     }
```

b) Write the implementation of the `SimPet.move` method. Run the JUnit tests provided and ensure that they all pass.

```
public void move() {
     if (energy > 0) {
          if (direction == 0)
               x = x + 1;
          else if (direction == 1)
               y = y + 1;
          else if (direction == 2)
               x = x - 1;
          else if (direction == 3)
               y = y - 1;
          energy = energy - ENERGY_TO_MOVE;
     }
}
```

c) Now suppose we want to add a method that will give a pet its shots. Write the specification for a method `SimPet.giveShots` and include a stub for this method. Assume that a pet can be given its shots only if it has an energy level of at least 5 and hasn't already had its shots. Note that a pet does not consume any energy when it is given its shots. Write your specification in such a way that there is no *requires* clause.

```
// REQUIRES:
// MODIFIES:  this
// EFFECTS:  if this pet is alive and it has not had its shots,
//             then record that the pet has now had its shots
```

d) In this part of the question, we ask you to demonstrate how to *use* a data abstraction. Write code that will create a new `SimPet` object located at the origin with 10 units of energy. Your code must then rotate the `SimPet` so that it is pointing west and move it forward 5 steps. Finally, declare a variable of an appropriate type and assign to it the amount of energy that your pet has remaining after rotating and moving.

```
public void doStuff() {
     SimPet p = new SimPet(0, 0, 10);
     p.rotateLeft();          // Or you could
     p.rotateLeft();
     for (int i = 0; i < 5; i++) {
          p.move();
     }
     int remainingEnergy = p.getEnergy();
}
```

Question 10. Data Abstraction:

The `ca.ubc.cs.cpsc210.kafe.CoffeeCard` class in the `KafeCompany` project contains a partial specification for a data type that represents a loyalty card for the Kafe company. A coffee card can be loaded with credits that can be used to purchase drinks at Kafe stores. Every time a drink is purchased, a bean is added to the card. For every 9 beans earned, a free drink is added to the card. To be purchased, some drinks require more credits than others. However, only one bean is earned per drink purchase, regardless of the number of credits required to purchase the drink.

Study the provided code for the `CoffeeCard` class carefully before continuing.

a) Suppose the topUp method has the following implementation rather than the one provided in the CoffeeCard class checked out of the repository.

```
public void topUp(int numCredits) {
      if (numCredits > 0)
      credits += numCredits;
}
```

Write the specification for the version shown above.

```
// top up credits
// MODIFIES: this
// EFFECTS: adds numCredits to number of credits on card
// only if numCredits > 0
```

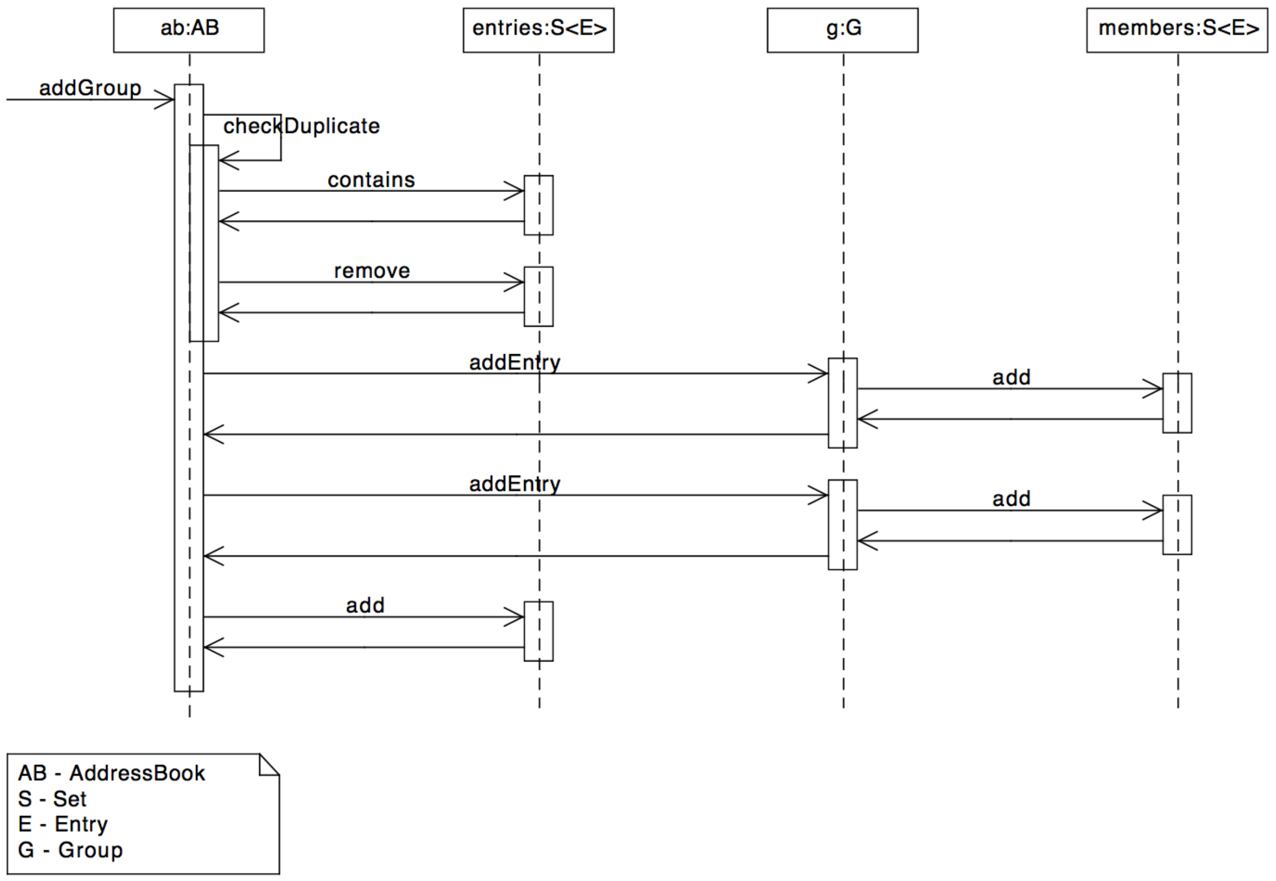
Question 13. Data Abstraction

Write an implementation for the `CoffeeCard.purchaseDrink` method. Note that some tests are provided for you in the `CoffeeCardTests` class but do not assume that these tests will catch every possible bug in your code. You may use Eclipse to develop your solution but you must make a copy of your work onto this exam paper *before the end of the exam*. It is not necessary to copy the provided documentation/comments.

```
public boolean purchaseDrink(int numCredits) {
      if (credits < numCredits)
            return false;

      credits -= numCredits;
      beans++;

      if (beans >= BEANS_PER_FREE_DRINK) {
            freeDrinks++;
            beans = 0;
      }

      return true;
}
```

Question 14: Control & Data Models

Draw a sequence diagram for the method `AddressBook.addGroup` within the package `ca.ubc.cs.cpsc210.addressbook`. If you have to loop over a collection, you must assume that there are exactly two objects in that collection. Include calls to *all* methods in the `EMailManager` project *except* constructors. You must also include the first level of calls to the Java library, if any. Be sure to include a legend if you abbreviate class names.



Note: In 2014W2 we are NOT stepping into calls to methods from the starting method. For instance, entries.contains(), entries.remove() and members.add() would not be needed in our diagrams. They are included here in case people decided to follow further on their own.