CPSC 210

Quiz Practice Questions

Note: the questions in this document do not constitute an actual quiz! Also, please keep in mind that this set of questions does not exhaust all the possibilities and therefore should not be used as your primary source of study material!

There are three systems that you will need to check out to answer these sample questions:

`JDrawing`

`PaymentSystem`

`SimGame`

`KafeCompany`

`EmailManager`

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(s) declared in the Java library.

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.

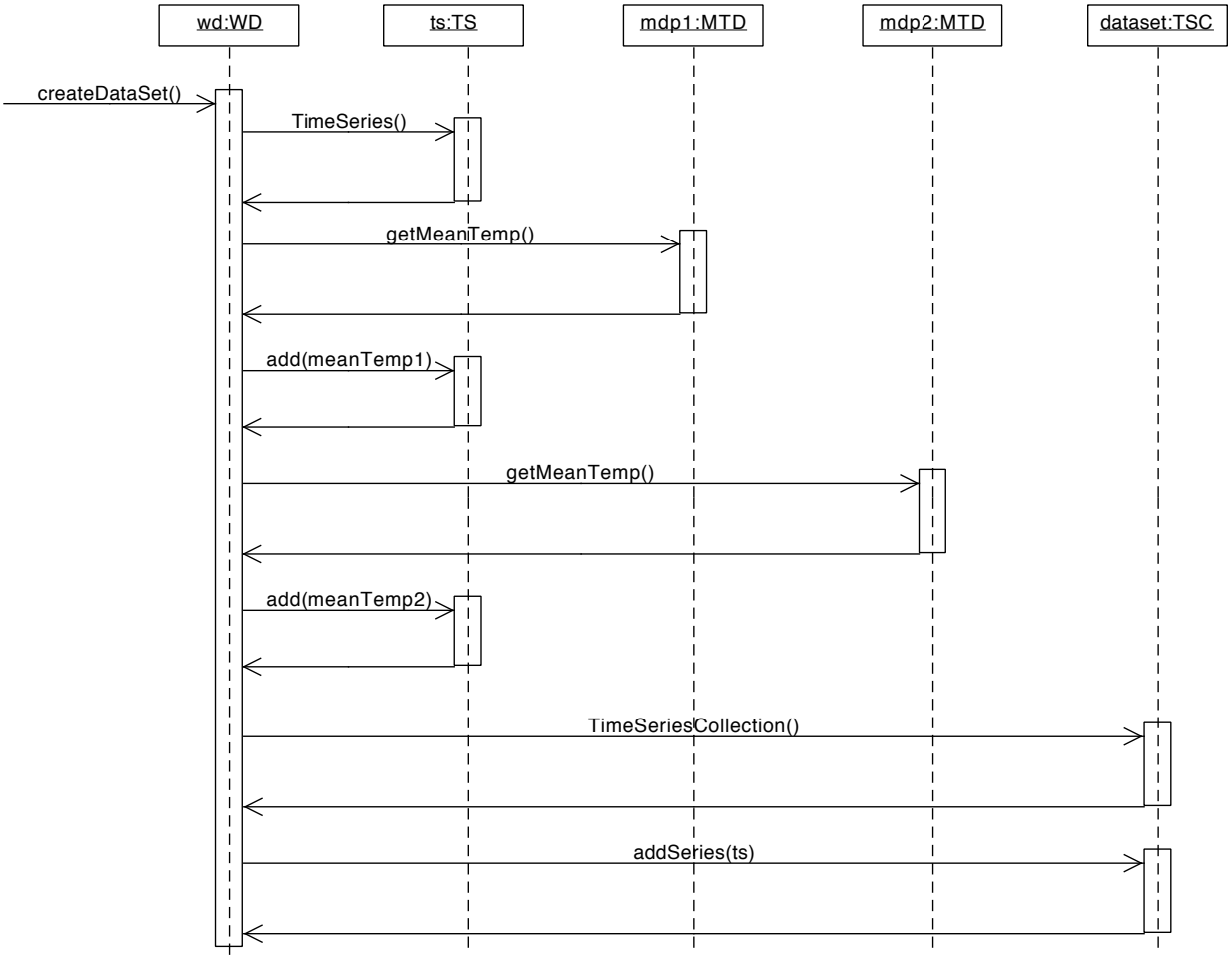
Question 3. Intra-method Control Flow (Flowchart)

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

Name Student No

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

IMPORTANT: Questions 5 through 8 apply to the `PaymentSystem`

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`. 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 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?

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 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

    public void payByTicket(Ticket t) {
        …
    }

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

Question 9. Data Abstraction:

The `SimGame` project contains the partial specification and implementation for a `SimPet` class, along with associated unit tests. A `SimPet` object 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.

You can use Eclipse to help you answer this question.

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.

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

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.

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.

Note: Questions 10 & 11 refer to the `KafeCompany` project checked out of the repository.

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.

Question 11. 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 help you with this question.

Note: Questions 12 refers to the `EMailManager` project checked out of the repository.

Question 12: 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.