CS2102, B15

Exam 2

Name:

You have 50 minutes to complete the problems on the following pages. There should be sufficient space provided for your answers.

If a problem asks you to create an interface, you should provide a complete interface, including method headers and argument types.

If a problem asks you to create a class:

- Include **implements**, **extends**, and **throws** statements as appropriate
- Include field names and types
- Omit constructors
- Omit methods unless a problem asks otherwise

Omit the `Examples` class (examples of data and test cases) unless a question asks otherwise.

If a problem needs you to give an example of a data structure (like a list, graph, or hashmap), you do **NOT** need to write these in full code (even in a test case). You may use reasonable abbreviations such as

- a picture for a graph
- `List[2,3]` for a list of numbers
- `HM[``Kathi'' -> 5118, ``Chris'' -> 5776]` for a hashmap from names to extensions

If a problem asks you to *specify* a data structure, provide the type of content as well as the name of the data structure (for example, a `LinkedList<String>`, a `HashMap<String,Dillo>`, a graph with People as nodes). You may indicate the content types either in code or in prose.

If a problem asks you to *evaluate* whether a data structure is a good choice for a problem, your answer should give us evidence that you understand the data structure (its traits, benefits, limitations, etc). Don't just say yes or no.

Grading Summary

Topic	Max Points	Score
Q1: Data Structures	30	
Q2: Java Programming	35	
Q3: Program Design	40	

(Each of these is a separate score in the corresponding course theme)

The questions on this exam all center around a common theme of patient medical records. All of the problems are about questions that a medical center might need to address about patient data.

(exam starts on the next page)

1. **(30 points) Data Structures**

   A medical center has given each patient a pedometer that reports how many steps the patient takes on every Date (month/day/year). The center plans to use the data to check (1) that patients are taking a similar number of steps each week, and (2) that the average steps per week is above 50,000.

   Below, we propose three data structures for the steps data for a *single* patient. Comment on whether each is a good choice for the steps-per-week checks described above (ignore other possible uses of the data). Briefly explain why (a sentence or two is enough), indicating strengths and/or weaknesses as appropriate. There may be multiple good choices or no good choices; evaluate each independently.

   Your answer should demonstrate that you understand the traits of the proposed data structure.

   The first two proposals use the following class to store the number of steps for a given date:

```
class DailySteps {
  Date whichDay;
  int numSteps;
}
```

   - `LinkedList<DailySteps>`, assuming the list is sorted by `Date` (from most recent to least recent).

   - `LinkedList<LinkedList<DailySteps>>`, where each inner list has `DailySteps` objects from the same week, but there is no specific order in either the inner or outer lists.

   - `HashMap<Date,Integer>` that maps each date to the number of steps taken that day.

(exam continues next page)

Classes Common to Questions 2 and 3 (No Problems on this Page)

EHR class: An Individual Electronic Health Record

This stores whether someone smokes, their current weight, and a history of their LDL (a kind of cholesterol) readings (most recent first). You don't need to know anything about LDL readings (just that they are just numbers).

```
class EHR {
  boolean isSmoker;
  double weight;
  LinkedList<Double> LDLHistory;
}
```

TestResult class: Results from a Blood Test

A lab reports blood-test results with objects from the following class: it contains the name of a patient and their numeric test result. This class does not specify which test the result is for. Depending on context, the number could be blood sugar, LDL levels, blood cell counts, etc.

```
class TestResult {
  String patientName;
  double result;
}
```

MedicalCenter Class: Patient Data and a Method

Both questions feature a `MedicalCenter` class that has a `HashMap` from patient names to their `EHR`s.

Both questions work with a method `updateLDLHistory` that takes a `LinkedList<TestResult>` and adds the result of each test to the `LDLHistory` in the corresponding patient's `EHR`. Test results for patients who are not in the `HashMap` are ignored.

Each question works with a slightly different version of this method, but the core functionality (storing test results) is identical in both questions. You may do the questions in either order.

This page does not contain questions.
It only contains classes common to multiple problems.

2. **(35 points) Exceptions/Java Programming**

One day, someone at the medical center accidentally updates patients' LDL history with results from a different test whose numeric results are much smaller than valid LDL values. To guard against this happening again, the center wants to check that test results are in an expected range before updating LDL data in `EHR` objects.

The current (unsafe) `updateLDLHistory` method (for updating `EHR` objects with LDL test results, as described on the previous page) is as follows:

```
class MedicalCenter {

  HashMap<String, EHR> patientData; // map patient names to EHRs

  void updateLDLHistory(LinkedList<TestResult> newData) {

     for (TestResult tr : newData) {

       EHR patientEHR = patientData.get(tr.patientName);

       if (!(patientEHR == null)) {

         patientEHR.LDLHistory.add(tr.result);

       }

     }

  }

}
```

Here is a method that checks whether every `TestResult` in a list has a `result` within a given range.

```
// check whether all numeric results are between the low and high values
boolean checkRange(LinkedList<TestResult> data, double low, double high) {
  for (TestResult tr : data) {
    if ((low > tr.result) || (tr.result > high)) // value out of range
      return false;
  }
  return true;
}
```

(a) (30 points) Edit either or both of these methods as needed to store the test results only if **all of them** lie within the range low=100 to high=250. If even one `TestResult` is out of range, none should be stored and a `NotLDLException` should be thrown. Include **`throws`** declarations as needed. Assume both methods are in the same class.

(b) (5 points) Somewhere on this page, write the class definition for `NotLDLException`. Assume it has no fields. You do **NOT** need to write the constructor.

(exam continues next page)

3. **(40 points) Encapsulation/Program Design**

In this question, as the `updateLDLHistory` method processes `TestResult`s, it also creates a "watchlist" of patients whose results have changed significantly since their last test. (The inner **`if`** statement and the `watchlist` variable are the differences from the version in question 2.)

```
class MedicalCenter {
  HashMap<String, EHR> patientData; // map patient names to EHRs

  LinkedList<String> updateLDLHistory(LinkedList<TestResult> newData) {
    // names of patients we should check on based on results
    LinkedList<String> watchlist = new LinkedList<String>();

    for (TestResult tr : newData) {
      EHR patientEHR = patientData.get(tr.patientName);
      if (!(patientEHR == null)) {
        // if difference to last LDL reading above 50, put patient in watchlist
        if ((patientEHR.LDLHistory.size() > 0) &&
            ((tr.result - patientEHR.LDLHistory.get(0)) > 50)) {
          watchlist.add(tr.patientName);
        }
        patientEHR.LDLHistory.add(tr.result);
      }
    }
    return watchlist;
  }
}
```

(a) (10 points) Draw boxes on the above code around computations that should be in a class other than `MedicalCenter` (which contains the `updateLDLHistory` method). Label each box with a unique number (1, 2, ...). You only have to draw and number boxes for this part.

(b) (30 points) Using the space on this page and the next (if needed), for each box:

- Write the method call (method name and arguments) that should replace the boxed-off code. This call can be to a new method (give it a descriptive name). **Write only the call, do not write the actual method.** Use the box numbers to label calls that are not immediately next to their boxes.
- Indicate which class each new method should be in. If the class exists, simply name it (i.e., "in class Dillo"). If the class does not exist, define it (giving the class name, fields, and field types).

(exam continues next page)

(Additional space to answer Question 3)

(end of exam)