NAME:

CS 2102
Exam 2
D-Term 2017

Question 1:	_____	(25)
Question 2:	_____	(10)
Question 3:	_____	(25)
Question 4:	_____	(10)
Question 5:	_____	(30)
TOTAL:	_____	(100)

1. (25 points) You're writing software to monitor a healthy lifestyle. The software maintains a database of foods and the number of calories per serving of each food.

   One of the methods you've developed recommends meals based on calorie counts. The user provides a list of foods they would like to eat, and the maximum number of calories they want to consume at one meal. The meal recommender produces a list of foods that the user could consume at one meal, such that the total calories comes in under the limit. For example, assume the database contained information on these foods (among many others):

   ```
   haddock, 250 calories          rice, 200 calories
   peas, 100 calories             cheetos, 300 calories
   soda, 150 calories             cake, 500 calories
   ```

   If the user provided all the foods given above in their list of desired foods, and set the maximum calories to 600, the recommender could return a list containing (haddock, peas, rice), or a list containing (cake), or a list containing (cheetos, soda, peas), etc. Also correct would be an answer where the list contained (soda, soda, soda), or an answer where the list contained (cheetos). There are other correct answers.

   The following code fragments show Java classes with the essential fields and methods

   ```
   // Users call the recommendMeal method of this class to get a meal
   // recommendation.  The list of desired foods provided to recommendMeal contains
   // the types of foods that the user likes to eat, like "haddock" and "cheetos"
   class Recommender{
   ...
     LinkedList<String> recommendMeal(LinkedList<String> desiredFoods,
                                      int calorieLimit){
   ...
   }

   class Food{
     String type;
     int caloriesPerServing;
     ...
   }
   ```

   Outline how you would test `recommendMeal`. Your test strategy should work for any valid list of desired foods (not just for the example given above). A good answer to this question will

   - describe any additional methods you need to write to support testing. For each method, indicate which class it goes in and provide only a signature and a clear purpose statement (don't write the method body).
   - give one concrete example of a JUnit test that illustrates your testing approach.

Assume that the database of foods exists elsewhere (do not try to define it), and that you are only testing foods that exist in the database. Ignore cases in which there is no valid meal that can be created from the desired foods.

Put the class/purpose/signature for each new method here (don't write the method bodies)

(continue your answer on the next page)

Write your JUnit test here (fill in the Examples class below)

```
public class Examples{
  // provide any initialization needed









  // test the recommendMeal method
  @Test
  public void testRecommendMeal(){









  }
}
```

2. (10 points) Draw pictures of each of the following data structures (where the elements are integers):

   (a) a balanced binary tree containing the numbers 1 through 7, such that the tree is neither a binary search tree nor a heap.

   (b) an AVL tree containing the numbers 1 through 8

3. (25 points) Choose the best answer for each of the following questions.

   (a) What is the purpose of the `throw` statement?

      i. It is used to pass arguments to another method.
      ii. It is used to detect an exceptional situation.
      iii. It is used to pass control to a handler when an exceptional situation is detected.
      iv. It defines a block of code to be executed upon detection of an exceptional situation.

   (b) When implementing a queue as an ArrayList, which of these statements is correct?

      i. For better efficiency, elements should be added at position `0` and removed from position `size()-1`
      ii. For better efficiency, elements should be added at position `size()-1` and removed from position `0`
      iii. There is effectively no difference in efficiency between the choices (i) and (ii)
      iv. You cannot implement a queue using an ArrayList

   (c) When implementing a stack as an ArrayList, which of these statements is correct?

      i. For better efficiency, elements should be pushed and popped at position `0`
      ii. For better efficiency, elements should be pushed and popped at position `size()-1`
      iii. There is effectively no difference in efficiency between the choices (i) and (ii)
      iv. You cannot implement a stack using an ArrayList

(d) Which of the following statements about hash tables is correct?

i. The hash code of the key is used to determine where to store each element
ii. A hash table allows duplicate keys
iii. No two keys of a hash table can have the same hash code
iv. The elements in a hash table are stored in order by key

(e) Which of the hash table methods (`put`, `get`, `containsKey`) make use of the `equals` method?

i. `put` only
ii. `put` and `get`, but not `containsKey`
iii. `containsKey` only
iv. all three methods make use of `equals`

4. (10 points) Briefly explain (in 1 or 2 sentences) what is meant by the "catch or declare" rule, as it pertains to checked exceptions in Java.

5. (30 points) A library keeps a list of its patrons. Each patron has a list of books that he or she currently has on loan from the library. The library wants to generate a list of the names of patrons who currently have overdue books. Here is the code (**you will be marking up this code when you answer parts (a) and (c)**):

```
import java.util.LinkedList;

// A list of Patrons at a library
class Patrons{
  private LinkedList<Patron> patrons;

  public Patrons(){
    this.patrons = new LinkedList<Patron>();
  }

  // add a Patron
  public void addPatron(Patron p){
    this.patrons.add(p);
  }

  // return list of names of Patrons with overdue books
  public LinkedList<String> delinquents(){
    LinkedList<String> delinquentPatrons = new LinkedList<String>();

    for (Patron p:  patrons){
      LinkedList<Book> pBooks = p.getBooks();
      for (Book b:  pBooks){
        if (b.isOverdue())

          // put the name of the delinquent patron in delinquentPatrons
          // only if the name isn't already there
          if(!delinquentPatrons.contains(p.getName()))
             delinquentPatrons.add(p.getName());
      }
    }
    return delinquentPatrons;
  }
}
```

code continues on next page

```
class Patron{
  private String name;
  private LinkedList<Book> books;  // books currently on loan by this Patron

  public Patron(String name){
    this.name = name;
    this.books = new LinkedList<Book>();
  }

  // add given Book to the list of Books on loan by this Patron
  public void addBook(Book b){
    this.books.add(b);
  }

  // return the name of this Patron
  public String getName(){
    return this.name;
  }

  // return the list of Books on loan by this Patron
  public LinkedList<Book> getBooks(){
    return this.books;
  }
}

class Book{
  private String title;
  private Date dateDue; // details of how a Date is represented omitted

  public Book (String title, Date dateDue){
    this.title = title;
    this.dateDue = dateDue;
  }

  // returns true if this Book is overdue
  public boolean isOverdue(){
    // details of implementation omitted
  }
}
```

Answer these questions about the given code:

(a) (5 points) The method `delinquents()` in the `Patrons` class violates the encapsulation principle that data and the methods that operate on that data should be be in the same class. Draw a box around the part of the code in `delinquents()` that violates this principle.

(b) (10 points) What statement(s) would you use to replace the code that you boxed off? If any of your statements are calls to new methods, state which class the new method would be defined in, and provide its signature and purpose. You do not have to write the body of the method.

(c) (15 points) (This problem refers to the Patron class; start with the code afresh for this problem, ignoring any changes you may have made to the Patron class for problem 5b). In the Patron class, you decide to encapsulate the representation of the patron's books. Edit the Patron class to allow for this modification. Mark edits on the original code. You may also use the space below for any new methods/interfaces. If you define a new interface, give the complete interface definition. If you define a new method, just give the purpose and signature.