NAME: SECTION:

CS 2102
Exam 1
D-Term 2014

Question 1:	_____	(10)
Question 2:	_____	(5)
Question 3:	_____	(15)
Question 4:	_____	(20)
Question 5:	_____	(30)
Question 6:	_____	(20)
TOTAL:	_____	(100)

1. (10 points) Given below are two of the axioms defined for the Abstract Data Type `Bag`. (You may remember that the ADT `Bag` is the same as the ADT `Set`, except that a `Bag` allows duplicate elements, while a `Set` does not.)

   (a) Restate this axiom in English in one sentence:

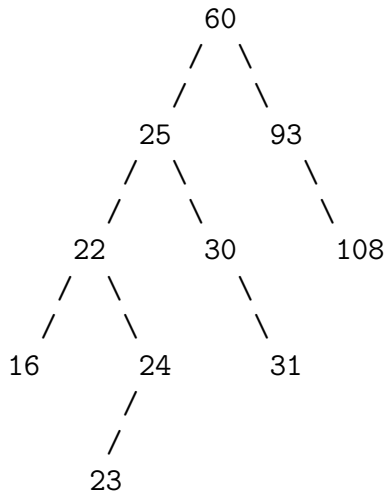
   ```
   // on addElt and remElt and hasElt:
       hasElt(remElt(addElt(addElt(B,e)),e),e) = true
   ```

   (b) Should this axiom for the ADT `Bag` also be an axiom for the ADT `Set`? Why or why not?

   ```
   // on addElt and size:
       size(addElt(B,e)) = size(B) + 1
   ```

2. (5 points) State the invariant for a heap.

3. (15 points) Here is a picture of a binary search tree of integers.



(a) (5 points) Is the pictured tree an AVL tree? If it is, explain why. If it is not, explain why not.

(b) (10 points) Draw a picture of the given binary search tree after the value 25 has been removed from the tree. (You should draw your picture based on the implementation for remElt() that we developed in class. Your result does not necessarily need to be an AVL tree.)

4. (20 points) An abstract class and two concrete classes are defined as follows:

```
abstract class AbsPet{
  public AbsPet(){}

  public String noiseMadeBy(){
    if (this instanceof Cat)
      return "meow";
    else
      return "woof";
  }









}

class Cat extends AbsPet{
  public Cat(){}









}

class Dog extends AbsPet{
  public Dog(){}









}
```

(a) (15 points) The method `noiseMadeBy()` checks the type of the pet to determine which noise it should return. We know that checking types in code is not a desirable thing to do in an object-oriented program. Edit the code so that `noiseMadeBy()` does not do any type-checking (i.e. does not use instanceof). (You may edit the code by writing on the text above; cross out removed code, add additional code, etc.) The signature of `noiseMadeBy()` in `AbsPet` should stay the same (you should restrict changes to the body of the method). You may implement additional methods, if necessary, in any of the classes `AbsPet`, `Cat`, and `Dog`.

(b) (5 points) What is the benefit of writing the code with no type-checks? (i.e., in three or fewer sentences explain the advantage of relying on Java's dynamic dispatch rather than explicitly checking the types of objects).

5. (30 points) In HW5 you are developing a weather monitoring tool that keeps a collection of Daily Weather Reports (DWR's). You may have defined classes that looked like this:

```
class WeatherMonitor{
  private LinkedList<DWR> report;

  public WeatherMonitor(){
     this.report = new LinkedList<DWR>();
  }
}

class DWR{
  private Date date;
  private double highTemp;
  private double lowTemp;

  public DWR(Date date, double highTemp, double lowTemp){
    this.date = date;
    this.highTemp = highTemp;
    this.lowTemp = lowTemp;
  }
}

class Date{
  private int month;  // 1 for January, 2 for February, ...
  private int day;
  private int year;

  public Date (int month, int day, int year){
    this.month = month;
    this.day = day;
    this.year = year;
  }
}
```

Assume there exists a method in the `WeatherMonitor` class that adds a DWR to the report list, and that this method has been used to populate the list of reports.

(a) (20 points) Write the following method for the `WeatherMonitor` class. You should write clean object-oriented code that follows the principles of encapsulation. If you need to write a helper method (or methods), indicate which class the method belongs to, and document the method with a purpose.

```
// counts the number of DWR's in this WeatherMonitor for the month of April
// (the day and year don't matter)
public int countAprilDWRs()
```

(b) (10 points) As given, the `WeatherMonitor` uses a linked list to hold the daily weather reports. Suppose you wanted to be able to choose, at the time you create your `WeatherMonitor`, from a number of different data structures for this purpose. Indicate the changes that would have to be made to the `WeatherMonitor` class to support this change. You may either rewrite the code for the `WeatherMonitor` class (crossing out and/or adding code on the previous page), or just explain what needs to be done.

6. (20 points) The class diagram on the last page of this exam represents a hierarchy of constituents of the WPI community. A constituent may be a faculty member, a non-faculty staff member, or a student. A student is either an undergraduate or a graduate. The `permitValid()` method required by the `IWPI` interface returns true if the constituent's `parkingPermit` number matches some (unspecified) criteria for valid parking permit numbers. What constitutes a valid permit number is different for each kind of constituent. Answer the following questions about the hierarchy.

   (a) (10 points) Write the constructor for the `Graduate` class, assuming the constructors for the abstract classes `AbsIWPI` and `AbsStudent` have been defined in the usual way.

(b) (10 points) Visitors come to WPI and are issued temporary parking permits. When the campus police make their daily rounds, they check the permits of all cars for validity, whether the car belongs to a visitor, an employee, or a student. Visitor permits are considered valid if the number of the permit is between 1 and 99 inclusive.

Extend the given hierarchy to include a class `Visitor`. You will do this by writing Java code. In Java code, write as much of the Visitor class as you can given the information in this problem.

(end of exam)