COMP 1406 In-Class Test #3 W18

Tutorial you attend:
☐ Tues 5:30 (B1) ☐ Wed 1:00 (B3)
☐ Friday 4:00 (B2)

Name: _____ **Student#:** _____

1. Consider the **Buffet** (all you can eat restaurant) and **City** classes partially defined as follows:

```
public class Buffet {
  private String name;
  private float fee;
  private ArrayList<String> food
}
```

```
public class City {
  private ArrayList<Buffet> buffets;

  //... remaining code omitted ...
}
```

(a) **[4 marks]** Assuming that all **getter** methods are available in each class, complete the following method in the **City** class so that it returns the **name** of the buffet with the best deal: That is, highest value of the ratio of fee/(# food items). If there are multiple buffets with the best deal return the first one you encounter. Return null if there are no buffets in the city.

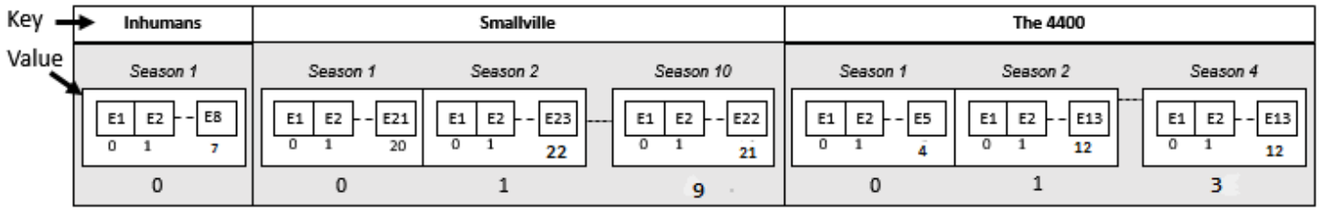
```
public String bestDeal() {
   if( buffets == null || buffets.size() == 0)   // 1/2 mark for null
      return null;

   Buffest best = buffets.get(0)

   for( Buffet b : buffets ){                    // 1 mark for looping

      if ( b.getFee()/b.getFood().size()     // 1 mark for using getters
            > best.getFee()/best.getFood().size() ){
         best =  b;
      }

   return best.getName();          // 1/2 return name of best deal buffet

                                    // 1 mark for solving max problem
                                    //  correctly
}}
```

(b) **[6 marks]** Write a proper **public** instance method in the **City** class called **foodAvailable(float money)** that returns an ArrayList of all the different food (Strings) that can possibly be eaten by spending at most money (dollars). Do not include duplicate items in the returned ArrayList. Return an empty ArrayList if there are none.

```
public ArrayList<String> foodAvailable(float money){

   ArrayList<String>  result;
   result = new ArrayList<String>();
   HashSet<String> food = new HashSet<String>();

   for (Buffet b: buffets) {

      if (b.getFee() <= money){
         if( !food.contains(b.getName() ){
            food.add(b.getName());
            result.add(b.getName);
         }
      }
   }
   return result;
}

1 mark - for correct method signature/return type
2 marks - generate and return ArrayList of all
          food available under money dollars
3 marks - use a set to prevent duplicates of food items
(2 marks if they repeatedly search the arraylist)
1/2 mark - for looping over the buffets
1/2 mark - for using getters
```

2. Consider the **FletNix** and **Episode** classes partially defined as follows: (do not worry about getters/setters)

```
public class FletNix {
   public HashMap<String, ArrayList<ArrayList<Episode>>>  tvShows;
   // i.e., value of hashmap is a list (of seasons) of lists of episodes
}
public class Episode {
   public String  name;
   public int     length;           // in minutes (e.g., a 42 minute episode)
   public boolean hardToUnderstand; // the episode was hard to follow
}
```



(a) **[5 marks]** Complete the **totalSeriesTime(**String name**)** method for the **FletNix** class that returns the total amount of minutes it would take to watch all episodes from all seasons for the tv series with the given input name (return **0** if show does not exist). Note that episodes that are *hard to understand* must be watched TWO times.

```
public int totalSeriesTime(String name){

    if (!tvShows.containsKey(name))                                  // 1 mark
          return 0;

    int count = 0;
    ArrayList<ArrayList<Episode>> seasons = tvShows.get(name);       // 1 mark

    for (ArrayList<Episode> episodes: seasons) {                     // 1 mark

       for (Episode episodes: e) {                                   // 1/2 mark
           count += e.length;                                        // 1/2 mark

           if( e.hardToUnderstand ) count += e.length;               // 1 mark
       }
    }
    return count;

}
```

(b) **[5 marks]** Describe in pseudocode how you would print out the names of all the episodes in a tv series sorted by the length of each episode. Which Abstract Data Types (Java classes) are needed for this? (If you prefer to write actual Java code that is fine).

There will be many solutions for this.

1 mark iterate over FletNix object to process each episode
1 mark to store each episode in something (arraylist or treeset of episodes). The mark is for a valid collection class from java.
1 marks for either implementing Comparable interface (override compareTo) or making Comparator object (with compare method) to sort things by length [1 mark for the class/interface, 1 mark for the method to override]
1 mark for the "sorting part". This might calling Collections.sort, Arrays.sort, or by using a TreeSet to store the data.
1 mark iterate over created structure and print out names in sorted order (by time)