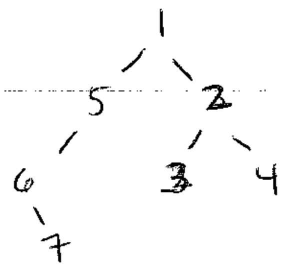
1. **Topic: Data Structures**

Draw an example (pictures, not code) of each of the following:

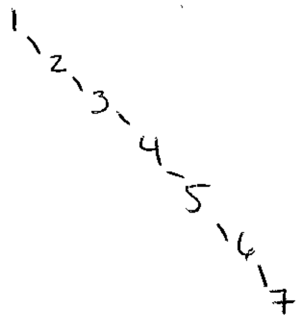
(a) A heap containing the numbers 1 through 7 that is NOT a binary search tree (BST)

pretty much any heap would do

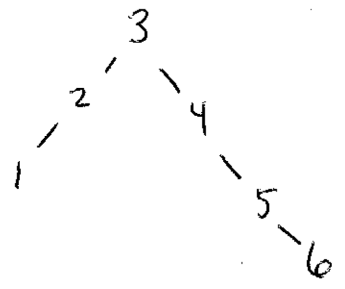


(b) A binary search tree (BST) containing the numbers 1 through 7 that is NOT an AVL tree

any unbalanced BST would do



(c) A binary search tree (BST) in which the left subtree has height 2, the right subtree has height 3, but the BST is not an AVL tree (Hint: numbers 1 through 6 are enough to answer this)



easiest example

(exam continues next page)

2. **Topic: Class Hierarchies (Classes, Abstract Classes, Interfaces) and Access Modifiers**

   You are developing an application to help people manage their meetings and to-do lists (question 3 starts from the same code). The application contains the following classes (constructors omitted to save space):

```
class Date {
  public    String month;
  public    int day;
}

class Meeting { extends Appt
  private   ~~String description;~~
  public    D~~ate when~~;
  private   String location;
}

class Calendar {
  private LinkedList<~~Meeting~~ IAppt> meetings;
  private LinkedList<String> toDoList;

  // produce list of appointments on the given date
  LinkedList<~~Meeting~~ IAppt> getReminders(Date fordate) {
    LinkedList<~~Meeting~~ IAppt> result = new LinkedList<~~Meeting~~ IAppt>();
    for (~~Meeting~~ IAppt m:meetings) {
      if (m.when.day==fordate.day && m.when.month.equals(fordate.month)) {
        result.addFirst(m);
      }
    }
    return result;
  }

  // check off a to-do list item
  void finishToDo(String toDoItem) {
    this.toDoList.remove(toDoItem);
  }
}
```

must be public so that if stmt in getReminders will work

After the initial product release, you realize that your application needs to manage scheduled phone calls as well as in-person meetings. For each phone call, you need to store a description of what the call is about, the date for the call (ignore the time to simplify the problem), the number to call, and the name of the person to call.

   (a) How could the original code have been written differently to let you make this change without editing any of the existing classes? Answer in text (do not write code or edit the above code).

   If used an interface for meetings in place of <Meeting> in LinkedLists in Calendar class

(exam continues next page)

(b) Edit the code to replace the current `LinkedList` of meetings with a `LinkedList` containing both meetings and phone calls. If you introduce any new classes, provide class definitions to the same level of detail as the `Date` and `Meeting` classes. Provide any new interfaces in full; you only need to reproduce the original functionality (not add new features or methods). Do **NOT** encapsulate data for this question—just make the edit requested in this problem. Mark edits on the original code, using the space around the code and below for any new classes/interfaces.

```
interface IAppt {}

abstract class Appt implements IAppt {
        String description;
        Date when;
    }
    class PhoneCall extends Appt {
        int number;
        String nameToCall;
    }
```

(c) Fill in the blanks in the code with appropriate access modifiers that would also allow the code to compile *as written.* You do not need to write anything else for this part.

(exam continues next page)

3. **Topic: Encapsulation**

You are developing an application to help people manage their meetings and to-do lists (question 2 starts from the same code). For this question, use the code in its original form, NOT with any edits made in question 2.

```
class Date {
  String month;
  int day;
}

class Meeting {
  String description;
  Date when;
  String location;
}

class Calendar {
  LinkedList<Meeting> meetings;
  LinkedList<String> toDoList;

  // produce list of appointments on the given date
  LinkedList<Meeting> getReminders(Date fordate) {
    LinkedList<Meeting> result = new LinkedList<Meeting>();
    for (Meeting m:meetings) {
      if (m.when.day==fordate.day && m.when.month.equals(fordate.month)) {
        result.addFirst(m);
      }
    }
    return result;
  }

  // check off a to-do list item
  void finishToDo(String toDoItem) {
    this.toDoList.remove(toDoItem);
  }
}
```

(a) On the code itself, box off the code fragments that need to change in order to encapsulate the meetings data in the Calendar class. You only need to draw boxes to answer this part.

(exam continues next page)

(b) Provide classes and/or interfaces you would add in order to encapsulate the `meetings` data. Provide complete interfaces. For classes, provide variables and headers for methods that are not required by their interface. Include only variables and method headers needed to encapsulate the code on the previous page (don't add new features).

```
interface IMeetings {
    ~~FindMtgs~~
    LinkedList<Meeting> findMtgsByDate (Date fordate);
}

class MeetingList ~~{~~ implements IMeetings {
    LinkedList<Meeting> meetings;

}
```

(c) Would well-encapsulated code require any changes to the comparison of dates in the `getReminders` method? Either explain needed changes briefly in text or justify why no changes are necessary (you do not need to write or edit code).

Would need to move comparison on day & month into the Date class

(end of exam)