CS2102, B15

Exam 1

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 a class hierarchy, we are looking for the interfaces, classes, and abstract classes that you would create for the problem. In particular:

- Include **implements** and **extends** statements
- Include field names and types
- Include method headers (names, return type, and input parameter types)
- Full credit requires that all types and implements/extends relationships are clear. Be sure your work is clear if you use class diagrams instead of Java syntax.
- You may omit constructors
- You may omit method bodies
- You may omit the `Examples` class (examples of data and test cases) unless a question asks otherwise

Grading Summary

Exam starts on the next page

Topic	Max Points	Score
Q1: Data Structures	30	
Q2: Program Design	50	
Q3: Java Programming	34	

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

(exam starts on the next page)

1. **Topic: Data Structures**

   (a) (20 points) Imagine that you were testing an approach to creating balanced binary search trees (such as AVL trees). You want a test case that would confirm that adding an element to an AVL tree does indeed return a balanced tree, as opposed to just a (potentially unbalanced) binary search tree.

   In the boxes below, give/draw ONE example of (a) a balanced binary search tree and (b) an integer such that inserting the integer into the tree using normal binary-search tree insertion (i.e, without rotation) would result in an unbalanced tree. You do **NOT** need to draw the resulting tree.

Balanced Initial Tree	New Element

   (b) (10 points) Draw an example of a binary search tree whose left and right subtrees at the root (top) have the same height, but the overall tree is not balanced.

(exam continues next page)

2. **Topic: Program Design (Classes, Abstract Classes, and Interfaces)**

   A company rents three kinds of vehicles: *bicycles*, *cars*, and *motorbikes*. For all vehicles, they record the *number of miles* the vehicle has been driven or ridden. Other information or methods, however, apply only to certain kinds of vehicles:

   - For bicycles, they also record the *number of gears*.
   - For motorized vehicles (here, cars and motorbikes), they need a method *tuneEngine* that returns a boolean indicating whether the vehicle is due for a tune-up.
   - Their sales department likes to run special promotions on vehicles that carry only one person (here, bicycles and motorbikes), so they need to maintain a list of just those vehicles (part (b) will ask you to show how you capture this information).

   (a) (40 points) Provide whatever classes, abstract classes, and interfaces you need to capture this information on vehicles. Include `implements` and `extends` statements as appropriate. Do **NOT** implement the *tuneEngine* method, but include its header in your classes and interfaces as appropriate.

(exam continues next page)

(b) (10 points) The following class captures the `RentalShop` data. Fill in the blanks with the appropriate types using your answer to part (a). Any class or interface used here should be included in your answer to part (a).

```
class RentalShop {

  LinkedList<______________________> allVehicles; // all vehicles

  LinkedList<______________________> holdsOnePerson; // bikes and motorbikes
}
```

(exam continues next page)

3. **Topic: Java Programming**

(34 points) The next page (page 7) provides classes and interfaces that capture information about WPI projects and co-ops (off-campus work for companies). Refer to those definitions to answer the questions on this page.

The `Examples` class on this page creates objects from the classes on the next page. This page lists several (partial) expressions that use those objects. Fill in each table indicating whether the field and method uses in each expression would compile (check one of "Compiles" or "No/Error"); if no, briefly explain the problem (at the level of "interfaces don't extend classes"). Ignore syntax issues/errors (such as missing semicolons).

Assume that all the object definitions compile (meaning all ... areas are filled in with appropriate code).

```
class Examples {
  Project MQP = new Project(...);
  IOffCampus anIQP = new GlobalProject(...);
  GlobalProject otherIQP = new GlobalProject(...);
  CoOp workAway = new CoOp(...);
}
```

In the expressions below, `contains` is the built-in method for checking whether something is in a list.

(a) `anIQP.sponsor.name.equals("IBM")`

Compiles	No/Error	Explanation if No/Error

(b) `otherIQP.sponsor.name.equals("IBM")`

Compiles	No/Error	Explanation if No/Error

(c) `anIQP.terms.contains(workAway.inTerm)`

Compiles	No/Error	Explanation if No/Error

(d) `otherIQP.terms.contains(workAway.inTerm)`

Compiles	No/Error	Explanation if No/Error

(e) `MQP.sponsor.isCompany`

Compiles	No/Error	Explanation if No/Error

(f) `MQP.hasCorporateSponsor()`

Compiles	No/Error	Explanation if No/Error

(exam continues next page)

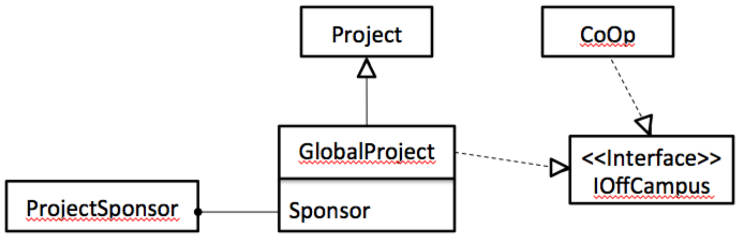
These are the class and interface definitions for the previous page. The diagram at the bottom of the page summarizes the relationships between the classes (dashed lines for “implements”, solid lines with arrowheads for “extends”, and solid lines with circles for “has a field of”). You can ignore the diagram if you wish.

Assume that the classes have appropriate constructors and that these definitions all compile (so any ... areas are filled in with appropriate code).

This page is only for reference. Do not write anything on this page.

```
interface IOffCampus {
  // is this work sponsored by a company
  boolean hasCorporateSponsor();
}

class Project {
  String title;
  LinkedList<String> terms;  // which terms project runs in
}

class GlobalProject extends Project implements IOffCampus {
  ProjectSponsor sponsor;    // who the project is for
  String location;           // "London", "Hong Kong", etc

  public boolean hasCorporateSponsor() { ... }
}

class CoOp implements IOffCampus {
  String inTerm;  // which term co-op runs in

  public boolean hasCorporateSponsor() { ... }
}

class ProjectSponsor {
  String name;
  boolean isCompany;
}
```

A graphical view of the classes and interface:



(end of exam)