CS2102 – B16

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

---

Your numeric grades on each question will be in a table on the last page.

1. **Topic: Program Design (Classes, Abstract Classes, and Interfaces) [40 points]**

   A friend is creating a program to manage recipes. Their code is on the next page. Your friend has created:

   - a *Recipe* class, which stores a list of ingredients, the baking temperature, and the baking time
   - a *Pizza* recipe which includes a topping and gets baked in the oven
   - an *IceCreamSundae* recipe which includes an ice cream flavor and an extra item but does not get baked

   Your friend also wants recipes to include a method to remove nuts from the ingredients when possible.

   The following questions ask you to modify your friend's code to achieve two goals. You may add new classes, abstract classes, or interfaces. Edit your friend's code to use your additions (just cross out details or write on the existing code – do not copy existing classes).

   (a) (20 points) As the constructor for *IceCreamSundae* shows, your friend sets the `bakeForMinutes` and `bakeAtTemp` fields to 0 for recipes that are not baked. Modify the existing code so that only recipes that get baked have these two fields. Write any new classes, etc in the space below on this page.

   (b) (20 points) Both *IceCreamSundae* and *Pizza* provide a *nutFreeVersion* method. Not all recipes can be made witout nuts, though. Assume we want to define a list of recipes that can be made without nuts. Edit the code so that *IceCreamSundae* and *Pizza* can appear in this list, then fill in the blank below with a type that can capture such recipes. Write any new classes, etc in the space below on this page.

```
LinkedList<____________________> nutFreeRecipes;
```

(question continues next page)

```java
class Recipe {
  LinkedList<String> ingredients;  // ingredients in this recipe
  int bakeAtTemp;                  // temperature for baking
  int bakeForMinutes;              // baking time in minutes

  // constructor
  Recipe(LinkedList<String> ingred, int bakeTemp, int bakeMins) {
    this.ingredients = ingred;
    this.bakeAtTemp = bakeTemp;
    this.bakeForMinutes = bakeMins;
  }

  // returns ingredients with old one replaced with new one
  LinkedList<String> substitute(String oldIngredient, String newIngredient) {
    // details omitted -- oldIngredient need not be in list
  }
}

// a recipe that does not require baking
class IceCreamSundae extends Recipe {
   String addOn;

   IceCreamSundae (String flavor, String addOn) {
    super (new LinkedList<String>(), 0, 0);
    this.addOn = addOn;
    this.ingredients.add(flavor + " ice cream");
    this.ingredients.add(addOn);
  }

  // replace nuts with cherries (if nuts are there)
  LinkedList<String> nutFreeVersion() {
    return this.substitute("nuts", "cherry");
  }
}

// a recipe that does require baking
class Pizza extends Recipe {
  Pizza (String topping) {
    super (new LinkedList<String>(), 450, 15);
    this.ingredients.add("cheese");
    this.ingredients.add(topping);
  }

  // already nut-free, nothing to change
  LinkedList<String> nutFreeVersion() {
    return this.ingredients;
  }
}
```

(exam continues next page)

2. **Topic: Java Programming – Classes and Objects Under the Hood [50 points]**

   The code on this page creates classes for managing two kinds of appointments: *Interviews* and *Retreats*. For sake of space, we have omitted methods and standard constructors (that set values of all fields).

   The next page provides a sequence of four expressions. Your job is to identify the objects that get created by running these four expressions.

   To get you started, the next page shows several possible objects that these expressions might create (drawn as boxes). For sake of space, we show only fields (no methods).

   (a) (24 points) For each possible object on the next page, either circle it if it would exist in memory (a.k.a. under the hood) *after all four expressions have been run*, or cross it out if it would not exist.

   (b) (10 points) Draw (in similar style) any additional/missing objects that Java would create.

   (c) (16 points) For any objects you circled or drew that have arrows coming out, connect the heads of the arrows to the objects that they refer to.

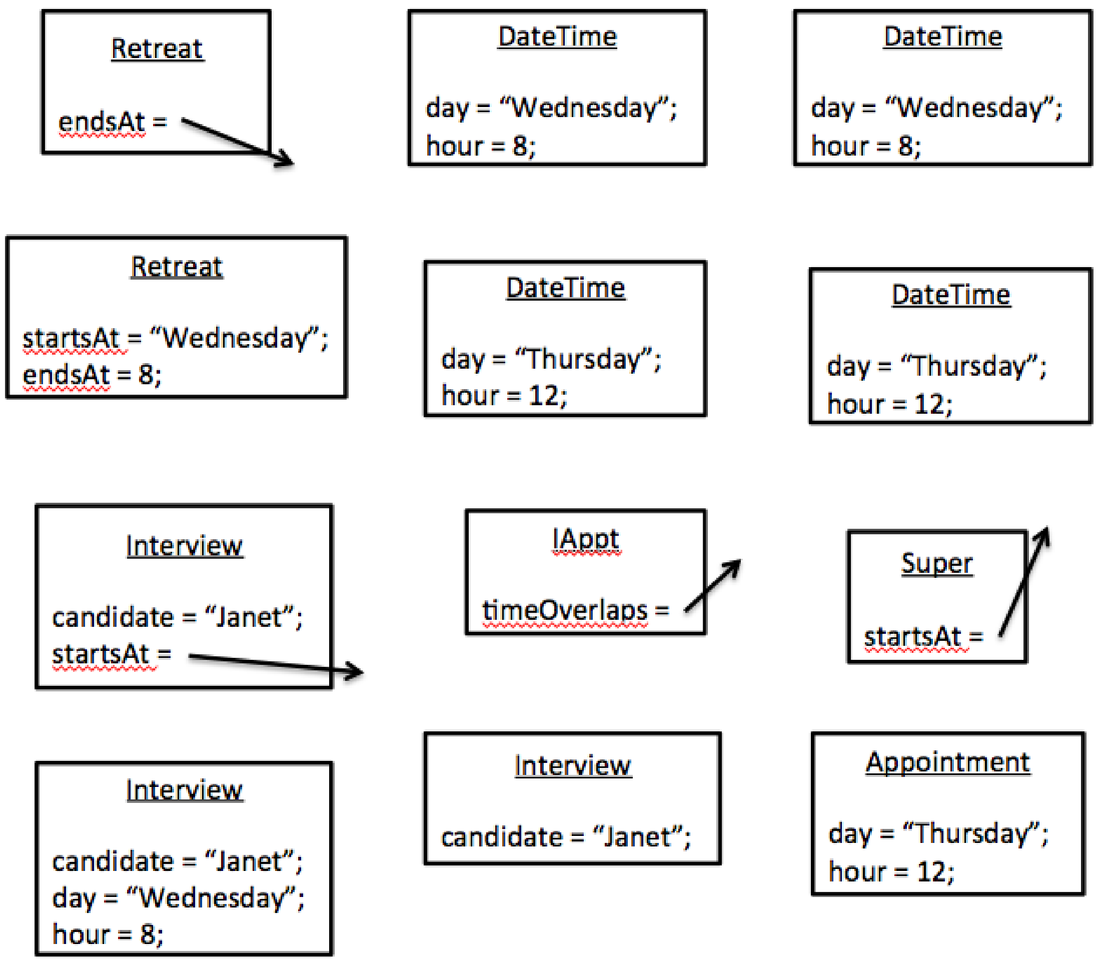
```
class DateTime {
  String day;
  int hour;  // in 24-hour format, so 13 means 1pm
  // standard constructor omitted
}

interface IAppt {
  boolean timeOverlaps (int anHour);
}

abstract class Appointment implements IAppt {
  DateTime startsAt;
  // standard constructor omitted
}

class Interview extends Appointment {
  String candidate;

  Interview(String candidate, DateTime start) {
    super(start);
    this.candidate = candidate;
  }
}

class Retreat extends Appointment {
  DateTime endsAt;

  Retreat(DateTime startsAt, DateTime endsAt) {
    super(startsAt);
    this.endsAt = endsAt;
  }
}
```

(question continues next page)

The Four Expressions To Evaluate:

```
Interview programmer = new Interview("Janet", new DateTime("Wednesday", 8));
DateTime thursLunch = new DateTime("Thursday", 12);
IAppt leadership = new Retreat(new DateTime("Wednesday", 8), thursLunch);
Interview tester = new Interview("Pat", thursLunch);
```

Possible Objects (circle or cross out each one, add missing ones on this page, and connect arrows):



(exam continues next page)

3. **Topic: Java Programming – Compiling Expressions [15 points]**

   Here are the same classes from question 2, now with a method in the *Retreat* class that returns the length of the retreat, as well as part of an `Examples` class. *The details of the methods are not important.* You must determine whether certain expressions would compile against this code (assume constructors were included).

```
class DateTime {
  String day;
  int hour;  // in 24-hour format, so 13 means 1pm
}

interface IAppt {
  boolean timeOverlaps (int anHour);
}

abstract class Appointment implements IAppt {
  DateTime startsAt;
}

class Retreat extends Appointment {
  DateTime endsAt;

  // computes how long the retreat lasts
  int numDays() { // details omitted ... }
  // determines whether retreat overlaps given time
  boolean timeOverlaps (int anHour) { // details omitted }
}

class Examples {
  Retreat staffRetreat = new Retreat(...);
  IAppt hiringRetreat = new Retreat(...);
}
```

   Assume that each of the following expressions was written in the `Examples` class. Indicate whether the field and method uses in each one 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 (such as missing semicolons).

   (a) `staffRetreat.startsAt.day.equals("Tuesday")`

Compiles	No/Error	Explanation if No/Error

   (b) `boolean timeClash = timeOverlaps(10)`

Compiles	No/Error	Explanation if No/Error

   (c) `hiringRetreat.numDays()`

Compiles	No/Error	Explanation if No/Error

(end of exam)

Grading Summary

Topic	Max Points	Score
Q1: Program Design	40	
Q2: Java Programming	50	
Q3: Java Programming	15	

(As on homeworks, the theme scores are separate. The exam is not intended to add up to 100 points.)