CIS1100.java — Fall 2024 — Exam 2

Full Name: _____

PennID (e.g. 12345678): _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

_____   _____

Signature   Date

Instructions are below. Not complying will lead to a 0% score on the exam.

- Do not open this exam until told by the proctor.
- You will have exactly 60 minutes to take this exam.
- Make sure your phone is turned OFF (not on vibrate!) before the exam starts.
- Food and gum are not permitted—don't be noisy or messy.
- You may not use your phone or open your bag for any reason, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is closed-book, closed-notes, and closed computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written in proper Java format, including all curly braces and semicolons.
- Do not separate the exam pages. Do not take any exam pages with you. The entire exam packet must be turned in as is.
- Only answers on the FRONT of pages will be graded. There is one blank page at the end of the exam if you need extra space for any graded answers.
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to you.
- When you turn in your exam, you may be required to show your PennCard. If you forgot to bring your ID, talk to an exam proctor immediately.
- We wish you the best of luck!

Q1	Q2	Q3	Q4	Q5	Q6 (bonus)

Q1. Apartment Unit Tests

Penn Apartments keeps track of the number of apartment units that are currently empty in their building using the following `Apartment` class:

```java
public class Apartment {
    private int empty;

    public Apartment(int numUnits) {
        empty = numUnits;
    }

    public boolean canRent() {
        return empty != 0;
    }

    public void rentUnit() {
        empty -= 1;
    }
}
```

Question 1.1

Consider the following unit tests and **mark the names of the ones that fail** when run with the implementation given above.

☐ testOne	☐ testTwo	☐ testThree	☐ testFour

```java
@Test
public void testOne() {
    Apartment a = new Apartment(0);
    assertEquals(false, a.canRent());
}
```

```java
@Test
public void testTwo() {
    Apartment a = new Apartment(2);
    assertEquals(true, a.canRent());
}
```

```java
@Test
public void testThree() {
    Apartment a = new Apartment(2);
    a.rentUnit();
    a.rentUnit();
    assertEquals(false, a.canRent());
}
```

```java
@Test
public void testFour() {
    Apartment a = new Apartment(2);
    a.rentUnit();
    a.rentUnit();
    a.rentUnit();
    assertEquals(false, a.canRent());
}
```

Question 1.2

Harry and Jessica are interns at Penn Apartments, and they each propose a fix to the implementation of `Apartment` that they claim will lead all of the tests to pass.

Harry wants to replace `canRent` with the below, and leave the other functions as is:

```java
public boolean canRent() {
    return empty < 0;
}
```

If we make just Harry's fix, will all four test cases on the previous page now pass?

☐ Yes   ☐ No

If you answered No, mark the names of the tests that fail.

☐ testOne   ☐ testTwo   ☐ testThree   ☐ testFour

Question 1.3

Separately, Jessica wants to replace `rentUnit` with the below, and leave the other functions as is:

```java
public void rentUnit() {
    if (empty > 0) {
        empty -= 1;
    }
}
```

If we make just Jessica's fix, will all four test cases on the previous page now pass?

☐ Yes   ☐ No

If you answered No, mark the names of the tests that fail.

☐ testOne   ☐ testTwo   ☐ testThree   ☐ testFour

Q2. Trading Up (Debugging)

Angelina has some Courses she registered for. She also has a bunch of other Courses she's interested in taking. The goal of the `bestReplacement` function is to compare one Course to an array of potential replacements and return the name of the Course with the highest rating that *doesn't meet on a Friday* and *has the same TE (technical elective) status as the course it would replace*. If there's no more highly-rated class that meets both criteria, the function should just return the one she was trying to replace.

In order to do this, she has defined a Record Type like so:

```
public record Course(String name, String days, boolean isTE, double rating)
{}
```

name is the name of the course, e.g. "CIS 1100". **days** is a String containing the initials of the days on which the course meets, where the only options are M, T, W, H, and F. The day Strings are always provided in this order; that is, Friday is always the last day initial in the String if it's there. **isTE** denotes whether a course is a "technical elective"—Angelina only wants to replace a course with another if they both are TEs or neither are TEs. **rating** is a number between 0 and 4 that denotes the course quality.

Help fix Angelina's implementation of `bestReplacement`. In the table that follows, identify the line on which each of the bugs appears, the part of the code that is incorrect, and the code you can replace that part with to make it correct. We filled in an example bug on the first line. Note that there are no further bugs in the function signature—i.e., the input and return types are correct. **There are six other bugs.**

```
1. public static Strong bestReplacement(Course replace,
2.                                       Course[] options) {
3.     Course bestSoFar = replace;
4.     for (int i = 1; i < options.length; i++) {
5.         Course currOption = options[i];
6.         String currDays = currOption.rating();
7.         char lastMeetingInWeek = currDays.charAt(currDays.length - 1);
8.
9.         boolean matchesCategory = currOption.isTE() != replace.isTE();
10.        boolean notOnFriday = lastMeetingInWeek != "F";
11.        boolean isBetter = currOption.rating() > bestSoFar.rating();
12.        if (matchesCategory || notOnFriday && isBetter) {
13.            bestSoFar = currOption;
14.        }
15.    }
16.    return bestSoFar.name();

    }
```

| Line Number	Incorrect Code	Replacement Code
1	Strong	String

Q3. Flight Itineraries

Consider this definition of Node:

```
public class Node {
    public String city;
    public Node next;

    // ...constructors omitted (not used in this problem)
}
```

Penn Airlines represents flight itineraries using a sequence of Nodes. For example, if a pilot flies from Denver to Austin to Boston to Denver, her itinerary would correspond to this sequence:
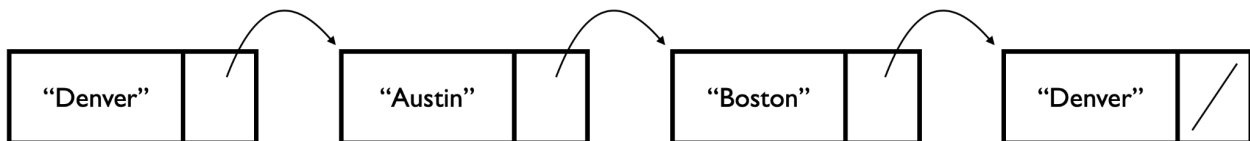
Question 3.1

An itinerary is called **agreeable** if it starts and ends in a pilot's home city. Your job will be to fill in the blanks in a function `isAgreeable` that, given a sequence of Nodes starting at `head` and a `homeCity`, checks if the sequence starts and ends at `homeCity`. Some examples:

- The example itinerary on the previous page is agreeable if `homeCity` is Denver, and not agreeable otherwise.
- An itinerary from Denver to Austin to Boston would never be agreeable.
- Itineraries with one node are agreeable if and only if the node's city is the home city.

```
// Assumes that head is not null.
public static boolean isAgreeable(Node head, String homeCity) {
    String startCity = ___BLANK_0___;
    if (!startCity.equals(homeCity)) {
        return ___BLANK_1___;
    }

    Node curr = head;
    while (curr.next != null) {
        ____BLANK_2_____;
    }

    String endCity = curr.city;
    return ___BLANK_3___;
}
```

Blank #	Code
0	
1	
2	
3	

Question 3.2

Next, implement `hasCity` below, which checks if an itinerary contains a particular city.
(Note that this question is unrelated to the previous, so you should not use `isAgreeable` here.)

```
/**
* Returns true if the sequence starting at head contains city,
* and false otherwise. You can assume that head is not null.
*/
public static boolean hasCity(Node head, String city) {




}
```

Q4. Mysteries

For each function below, suggest a name that would be descriptive for that function. Recall that good function names are typically verbs or short verb phrases, e.g. `repeatKTimes`, `flip`, or `isSymmetric`. There are several acceptable answers for each function.

Question 4.1

```
// Assumes n >= 0.
public static int mystery(int n) {
    if (n == 0) {
        return 0;
    }
    return n + mystery(n - 1);
}
```

Write **one** suggested name for the above function here:

```
```

Question 4.2

```
// Assumes n >= 0.
public static boolean mystery(int n) {
    if (n == 0) {
        return true;
    } else if (n == 1) {
        return false;
    }
    return mystery(n - 2);
}
```

Write **one** suggested name for the above function here:

```
```

Question 4.3

```
public static String mystery(String str) {
    if (str.equals("")) {
        return str;
    }
    return mystery(str.substring(1)) + str.charAt(0);
}
```

Write **one** suggested name for the above function here:

```
```

Question 4.4

```java
public static boolean mystery(int[][] arr) {
    int x = arr.length;
    for (int i = 0; i < x; i++) {
        if (arr[i].length != x) {
            return false;
        }
    }
    return true;
}
```

Write **one** suggested name for the above function here:

Q5. Object-Oriented Zoo

We're opening a zoo! Zoos contain animals, which are of some type and require some food:

```java
public class Animal {
    private String type;      // type of animal (e.g. "Tiger")
    private int foodRequired; // daily food required, in cups (e.g. 5)

    public Animal(String type, int foodRequired) {
        this.type = type;
        this.foodRequired = foodRequired;
    }

    public String getType() {
        return this.type;
    }

    public int getFoodRequired() {
        return this.foodRequired;
    }
}
```

Question 5.1

There are also zookeepers, who each specialize in a certain type of animal:

```
public class Keeper {
    private String specialty; // type of animal they specialize in

    public Keeper(String specialty) {
        this.specialty = specialty;
    }
}
```

Inside the `Keeper` class, we will add a `specializes` method that checks whether the zookeeper specializes in the given animal's type. Your implementation must satisfy these tests:

```
@Test                                    @Test
public void testDoesSpecialize() {       public void testDoesNotSpecialize() {
    Keeper k = new Keeper("Lion");           Keeper k = new Keeper("Tiger");
    Animal a = new Animal("Lion", 10);       Animal a = new Animal("Lion", 10);
    assertEquals(                            assertEquals(
      true, k.specializes(a));                 false, k.specializes(a));
}                                        }
```

Provide a **one line implementation** of the method below. Keep in mind that the answer depends on comparing Strings, one of which is a *private* instance variable of an object of a different class.

```
public boolean specializes(Animal a) {



}
```

Question 5.2

The Zoo class contains a list of Animals, like so:

```
public class Zoo {
    private ArrayList<Animal> animals;
    public Zoo() {
        animals = new ArrayList<Animal>();
    }

    public void addAnimal(Animal animal) {
        animals.add(animal);
    }
}
```

Inside the Zoo class, we will add a `calculateFoodRequired` method which, for a given `Keeper`, figures out how much food they need in order to feed all of the animals **whose type they specialize in**. Here is an example test that your implementation must satisfy:

```
@Test
public void testCalculate() {
    Keeper k = new Keeper("Lion");
    Zoo z = new Zoo();

    z.addAnimal(new Animal("Tiger", 5));
    assertEquals(0, z.calculateFoodRequired(k));

    z.addAnimal(new Animal("Lion", 10));
    assertEquals(10, z.calculateFoodRequired(k));

    z.addAnimal(new Animal("Bobcat", 15));
    assertEquals(10, z.calculateFoodRequired(k));

    z.addAnimal(new Animal("Lion", 20));
    assertEquals(30, z.calculateFoodRequired(k));
}
```

Fill in the implementation of the method below. You can use the `specializes` method from before, as well as any other methods defined previously.
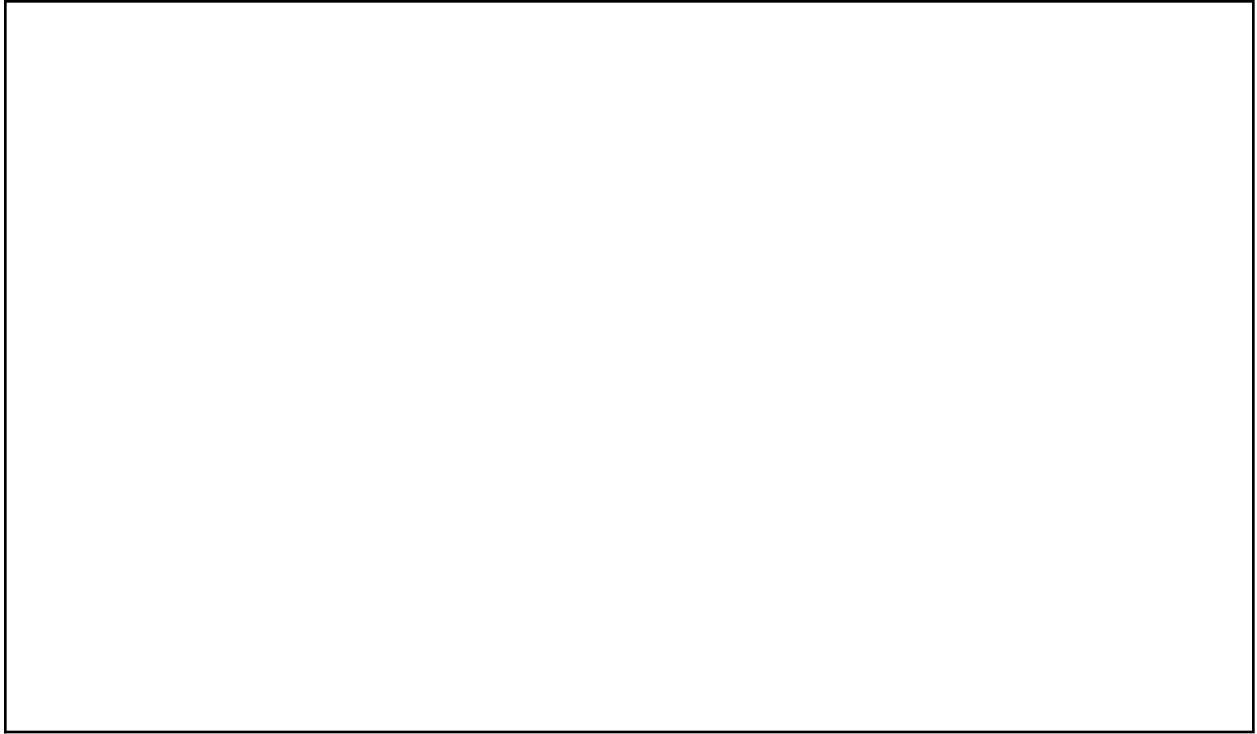
```
public int calculateFoodRequired(Keeper k) {
    int totalFood = 0;
    for (int i = 0; i < animals.size(); i++) {
        Animal a = animals.get(i);

        // TODO: Write your code here.




    }
    return totalFood;
}
```

Q6. (Bonus)

Recommend us something! Anything at all—music, movie, TV, game, book, restaurant, park, anything. All exams will get full credit for this question, no pressure to write anything in particular!

Extra Answers Page (This page is intentionally blank)

You may use this page for additional space for answers; keep it attached to this exam. Clearly note on the original question page that your answer is on this extra page, and clearly note on this page what question you are answering.