00203

THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 210: MIDTERM EXAM – MARCH 4, 2015

Last Name: TAYLOR First Name: BRIAN

Signature: Brian Taylor UBC Student #: 5,2,3,1,1,8,5,9

Important notes about this examination

1. You have 90 minutes to complete this exam. Put away books, papers, cell phones ... everything but your laptop (or lab computer), pens, pencils, erasers, id card, and this exam. On your laptop or lab computer you may **only** access the Java projects: HotelReservationSystem, MarioOlympicGames, PizzaMidterm (we will indicate where to find these projects)
2. You may not open any program other than Eclipse on your laptop, and cannot access code from any existing projects in your workspace. Selected Java documentation is provided to you on hard copy.
3. No other materials may be consulted. Good luck!

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. Examination candidates are not permitted to ask questions of the examiners or invigilators, except in cases of supposed errors or ambiguities in examination questions, illegible or missing material, or the like.
3. No examination candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
   i. speaking or communicating with other examination candidates, unless otherwise authorized;
   ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
   iii. purposely viewing the written papers of other examination candidates;
   iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
   v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)—(electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

Please do not write in this space:

10	Question 1:	5
10	Question 2:	4
12	Question 3:	5
14	Question 4:	4
26	Question 5:	22
6	Question 6:	3
70		

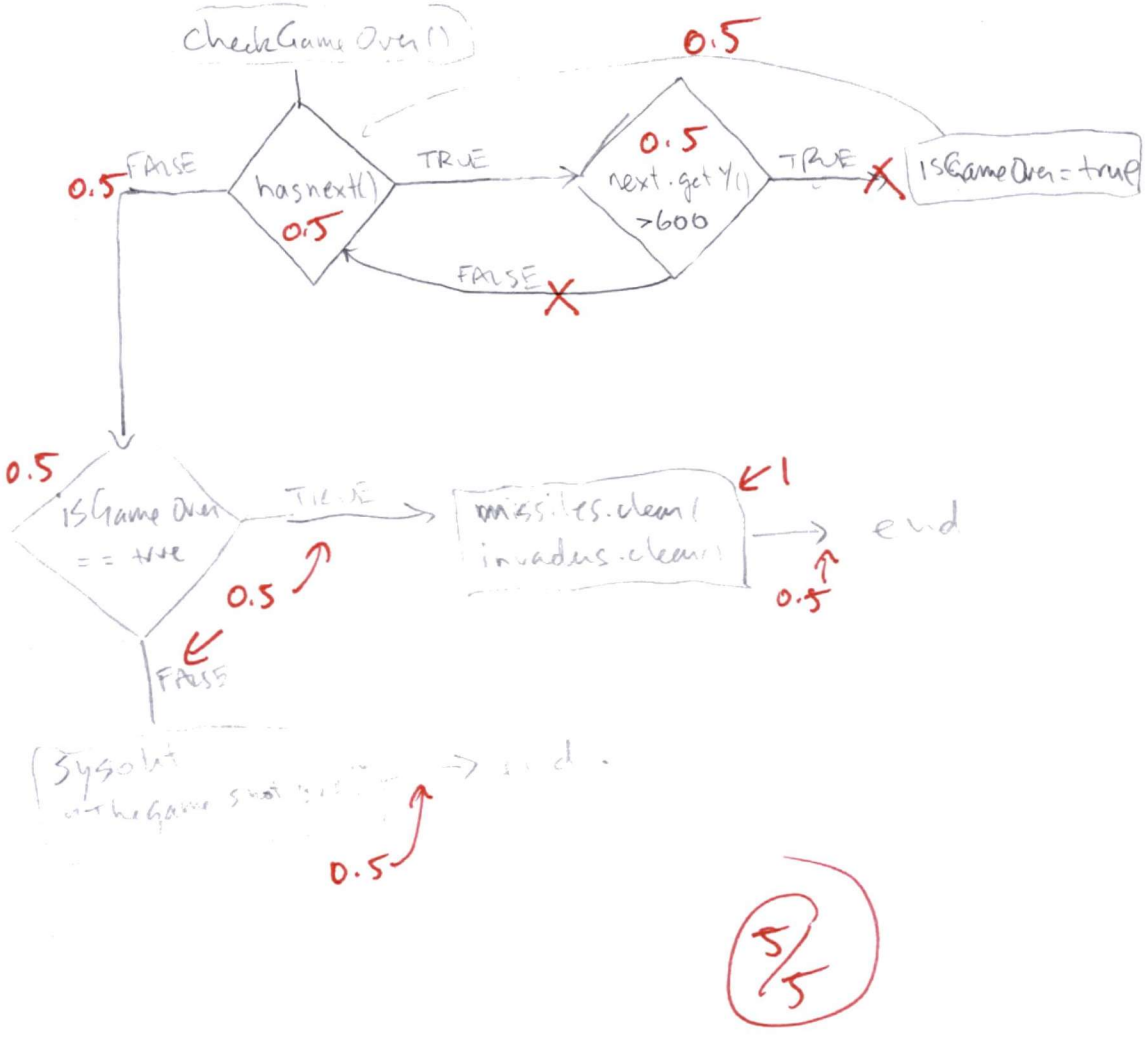
0 236366 453321

Question 1. [10 points total; 5 points for each part]

This question refers to the **"Question 1 code"** listing in the hard copy code pack you received.

Part 1

Draw the intra-method control flow graph (i.e., flow chart) for the `SIGame checkGameOver` method. You may use line numbers to refer to code instead of re-writing the code in the diagram you draw.



Part 2

Below, we have provided you with a number of empty test methods to test `checkGameOver` method in the `SIGame` class. Write the **minimal** combination of test methods to ensure every line of code in `SIGame`'s `checkGameOver` method is executed at least once by at least one of the methods. You do not need to use any `assert` methods in your tests. These methods should be written in Java code.

```
@Test
public void testA(){









}

@Test
public void testB() {

}
```

```
@Test
public void testC() {

}

@Test
public void testD() {

}
```

Question 2. [10 marks total]

This question refers to the **"Question 2 code"** listing in the hard copy code pack you received.

Part 1

A. [2 marks] Circle the appropriate answer for the following statement.

A `SumpPump` object is substitutable for a `Pump` object according to type substitutability rules.

(TRUE) ... FALSE ✓

B. [3 marks] Write a few lines of Java code (i.e., instantiate a `SumpPump` object and demonstrate with calls to its methods with parameter values) to demonstrate whether a `SumpPump` object is or is not substitutable for a `Pump` object.

```
SumpPump sp = new SumpPump(); (1)
(1) sp.openValve(400); ✗
```

Since the openValve method requires pressure < 500.00 a SumpPump object doesn't break the requirements of a PumpObject's openValve method.

4/5

Part 2

C. [2 marks] Circle the appropriate answer for the following statement.

A `WellWaterPump` object is substitutable for a `Pump` object according to type substitutability rules.

TRUE ... (FALSE) ✗

D. [3 marks]
Write a few lines of Java code (i.e., instantiate a `WellWaterPump` object and demonstrate with calls to its methods with parameter values) to demonstrate whether a `WellWaterPump` object is or is not substitutable for a `Pump` object.

~~WellWaterPump~~ Pump wwp = new WellWaterPump(); (1)

(1) wwp.openValve(1500); ✗

This line indicates the unsuitability of WellWaterPump since its a valid call to openValve in the WellWaterPump class definition by openValve, but it does not satisfy the REQUIRES clause of Pump class's openValve method.

7/5

Question 3. [12 points]

This question refers to the **"UML Diagram for Question 3"** provided in the hard copy code pack you received. You may also refer to the UML reference sheet entitled **"UML Class Diagram Notation"** in that same code pack.

The UML class diagram describes a University department system. Your task is to provide (in the space below) the implementations of the associations and aggregations shown in the diagram for the `Professor`, `Course`, `Lecture`, and `LearningOutcome` classes. You only need to focus on the implementation of the associations and aggregations in the diagram that involve these classes. For instance, your implementation must express the relationship between `Course` and `Lecture`, but not between `Course` and `Student`. Similarly you do not need to include fields in your implementation that do not pertain to the associations and aggregations you are implementing.

One extra constraint not shown in the diagram is that there can not be any duplicate `Lectures` for a `Course`.

You must add appropriate constructors and fields to these classes to complete the implementations of the associations and aggregations. You must also complete any method shown for the class that pertains to the association or aggregation. You may assume that appropriate `hashCode()` and `equals()` methods have been generated for each of these classes. You may also assume all necessary import statements have been provided.

```
/** Adds itself to the Course via the assignProf(...) method in Course. */
class Professor { extends DepartmentMember {
```

```
Course course1 = new Course( String website, Integer number);
Course course2 = new Course( String website, Integer number);
```

4

4/6

```
}
```

```
/** A lecture may be added using addLecture(...). */
class Course {
```

Collection students; ∅

0/4

```
}
```

```
/** Uses addLearningOutcome(...) to add a LearningOutcome. */
class Lecture {
```

Collection learningOutcomes; (1)

1/2

```
}
```

```
class LearningOutcome {

}
```

0/1

Question 4. [14 points]

This question relates to the HOTELRESERVATIONSYSTEM Java project. You were directed at the start of the exam where to download this code from the repository

Part 1

Consider the `testMakeReservationOverLimit` method in `HotelTest` in the `test` package. In the version of the code we have given you, this test passes. Your task is to improve the robustness of the `CreditCard` class such that a `DeclineChargeException` occurs if a guest does not have enough credit limit to charge the room.

A. [5 marks] Rewrite the specifications and implementations for the method or methods you would alter in the `CreditCard` class to improve the robustness of `CreditCard` as described above. If you introduce any new types, you must also provide definitions and implementations of those types.

```
REQUIRES: amount > 0 (1)
EFFECTS: ...?
public void approvePayment(double amount) throws DeclineChargeException {
    if (getCharges() + amount (1) > chargeLimit)
        throw new DeclineChargeException(); (2)
    charges = charges + amount;
}
```

B. [2 marks] Here is a reproduction of the `testMakeReservationOverLimit` method. Mark up the test method as to show how you would change it to appropriately test the functionality described in part 1.

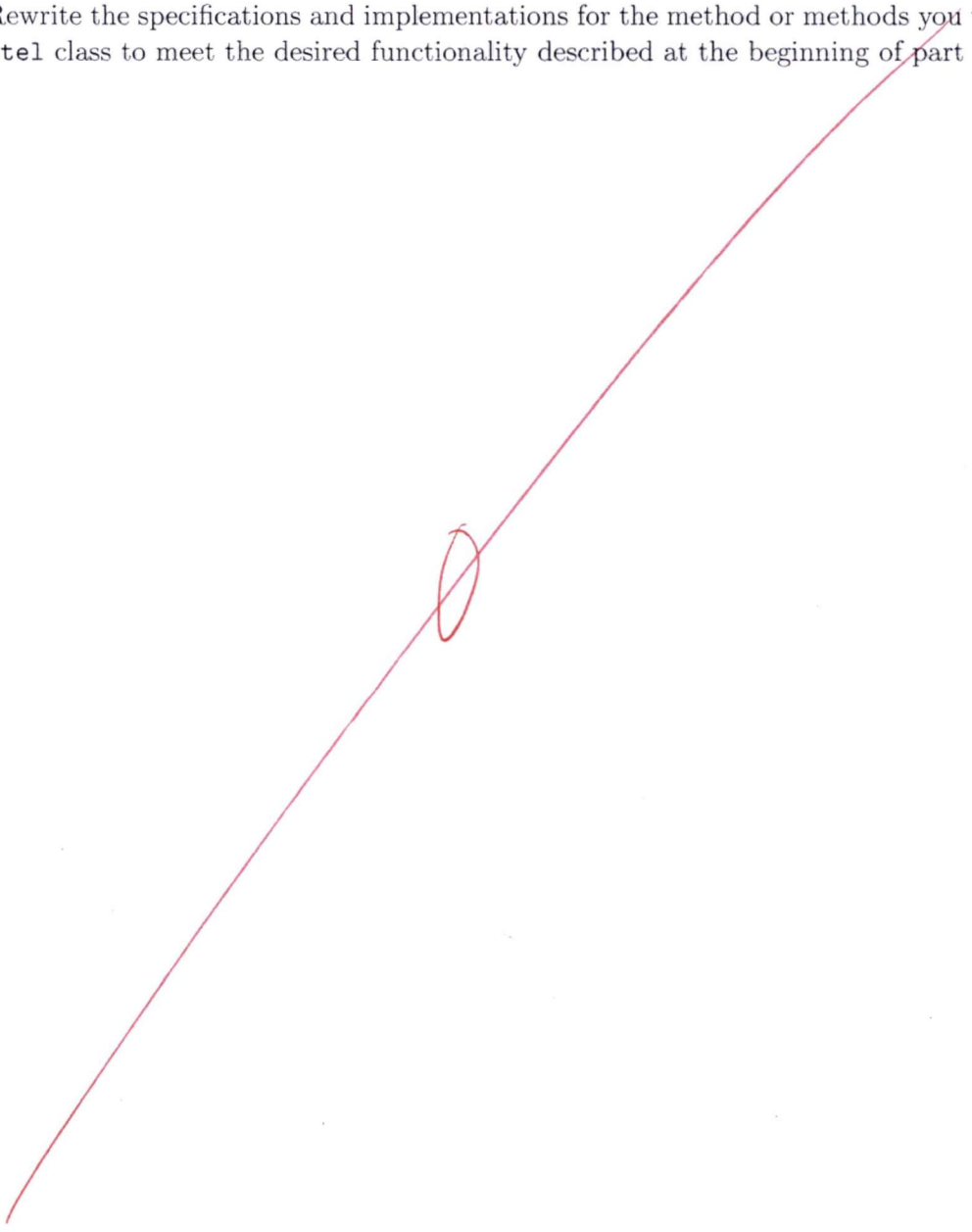
```
1  @Test
2
3  public void testMakeReservationOverLimit() {
4
5    // Hold a room, one should be available
6
7    Room room = hotel.holdRoom();
8
9    assertTrue(room != null);
10
11
12   // Create a guest for the room
13
14   Guest guest = new Guest("Elisa", new CreditCard(100));
15
16
17   // Approve the payment for the room
18
19   CreditCard c = guest.getCreditCard();
20
21   c.approvePayment(room.getRate());
22
23
24   // Book the room
25
26   hotel.bookRoom(guest, room);
27
28   c.applyPayment(room.getRate());
29
30 }
```

4/7

Part II

Consider the `testAssignAGuestAnAssignedRoom` method in `HotelTest` in the `test` package. In the version of the code we have given you, this test passes. We want to improve the robustness of the `Hotel` class such that the caller of `bookRoom` does not need to ensure that the room specified to book has not already been assigned. If you introduce any new types, you must also provide definitions and implementations of those types.

A. [5 marks] Rewrite the specifications and implementations for the method or methods you would alter in the `Hotel` class to meet the desired functionality described at the beginning of part 2.



B. [2 marks] Here is a reproduction of the `testAssignAGuestAnAssignedRoom` method. Mark up the test method to show how you would change it to appropriately test the functionality described at the beginning of part 2.

```
1  @Test
2
3  public void testAssignAGuestAnAssignedRoom() {
4
5          // THIS TEST SHOULD NOT PASS AS THE ROOM IS ALREADY ASSIGNED.
6          // HOW CAN YOU MAKE HOTEL MORE ROBUST TO SIGNAL THE ERROR AND
7          // HOW SHOULD THIS TEST CHANGE?
8
9
10         // Hold a room, one should be available
11
12         Room room = hotel.holdRoom();
13
14         assertTrue(room != null);
15
16         // Create a guest for the room
17
18         Guest guest1 = new Guest("Gail", new CreditCard(500));
19
20         // Book the room
21
22         hotel.bookRoom(guest1, room);
23
24         // Create a second guest
25
26         Guest guest2 = new Guest("Elisa", new CreditCard(1000));
27
28         / Book the same room
29
30         hotel.bookRoom(guest2, room);
31
32 }
```

Question 5. [26 points, 2 per correct, -1/4 per wrong]

These questions refer to the MARIOOLYMPICGAME Java project. You were directed at the start of the exam where to download this code from the repository.

Be careful with your time. If you try to code all of these questions, you may not finish as we have given only skeleton code. For a true/false question, you may only select either true or false. For a multiple-choice question, you may select only one answer. Circle your choice of answer for each question.

Note that some of these questions make a call to `System.out.println(...)`. This call writes the given parameter to the console. So `System.out.println(''Foo'')` writes **Foo** to the console. Each call to `System.out.println(...)` writes the output to a separate line on the console.

Part 1

Consider the following code:

```
1 List<Sport> bestSports = new ArrayList<Sport>();
2 bestSports.add(new Javelin());
3 Luigi l = new Luigi(bestSports);
4 l.addFriend(new Mario(bestSports));
5 Character c = l;
6 String name = c.getName();
```

At Line (6), the apparent type of the object referred to by the variable `c` is:

(a) nothing, because this code will neither compile nor execute

(b) `Character`

(c) `Luigi`

(d) `Mario`

Part 2

Considering the same code as for Part 1.
At Line 6, the actual type of the object referred to by the variable `c` is:

(a) nothing, because this code will neither compile nor execute

(b) `Character`

(c) `Luigi`

(d) `Mario`

Part 3

Consider the following code:

```
1 SharedCharacterBehaviour scb = new Birdo();
2 scb.fly();
```

Is the following statement true or false:

This code will compile and execute. ..... TRUE .. FALSE

Part 4

Consider the following code:

```
1 try {
2   FiftyMetreDash fmd = new FiftyMetreDash();
3   fmd.add(new Birdo());
4   fmd.add(new Birdo());
5   fmd.performRun();
6   } catch(FlyingException fe) {
7       System.out.println("A flying exception!");
8 }
```

When this code executes, the program prints out:

(a) **A Birdo crashed**
A Flying exception!

(b) **The game has problems.**

(c) **A Birdo crashed**
The game has problems.
A flying exception!

(d) None of the above

Part 5

Consider the following code:

```
1  try {
2    MedlaySwimRelay msr = new MedlaySwimRelay();
3    msr.addPeach(new Peach());
4    msr.getWet();
5    } catch (GameException ge) {
6      System.out.println("The race is over.");
7    }
8  System.out.println("Try another race");
```

When this code executes, the program prints out:

(a) **The race is over.**
Try another race.

(b) **Get the lifeguard, peach is drowning!**
Don't let peach swim
Try another race

(c) **Get the lifeguard, peach is drowning!**
Get the lifeguard, peach is drowning!
Don't let peach swim
Try another race

(d) **Get the lifeguard, peach is drowning!**
Don't let peach swim
The race is over.
Try another race.

(e) None of the above

Part 6

Consider adding the following exception type:

```
1 public class SinkingException extends RuntimeException {
2 }
```

Consider then changing the `Peach swim()` method to throwing a `SinkingException` instead of a `DrowningException`.
Consider the following code given these changes:

```
1  try {
2    MedlaySwimRelay msr = new MedlaySwimRelay();
3    msr.addPeach(new Peach());
4    msr.getWet();
5    } catch (SinkingException se) {
6        System.out.println("Peach is sinking");
7    } catch (GameException ge) {
8        System.out.println("The race is over." );
9  }
10 System.out.println("Try another race");
```

When the program runs it prints out:

(a) **Get the lifeguard, peach is drowning!**
Get the lifeguard, peach is drowning!
and so on until the program runs out of stack

(b) **Don't let peach swim**
Peach is sinking

(c) **Don't let peach swim**
Peach is sinking
The race is over
Try another race

(d) **Don't let peach swim**
Peach is sinking...
Try another race.

Part 7

Consider the following code:

```
1  int timeToFly;
2  timeToFly = // initialized by reading a value provided by the user
3  try {
4     Birdo b = new Birdo();
5     b.fly(timeToFly);
6  } catch(FlyingException fe) {
7     System.out.println("Flying exception.");
8  } finally {
9     System.out.println("Finally");
10 }
```

Which of the following statements is most accurate about this block of code when it executes:

(a) The word **Finally** is only written to the console if `timeToFly <= 5`

(b) The word **Finally** is always written to the console and **Flying exception** may or may not be written to the console

(c) If **Flying Exception** is written to the console, then **Finally** is not

(d) Neither **Flying Exception** nor **Finally** are ever written to the console

Part 8

Consider the following code:

```
1  try {
2     Birdo b = new Birdo();
3     b.fly(timeToFly, -2);
4  } catch(FlyingException fe) {
5     System.out.println("Flying exception.");
6  } catch(GameException ge) {
7     System.out.println("Game exception");
8  }
```

When this code executes:

(a) Only the `fly` method declared at line 15 of the `Birdo` class (i.e., `public void fly(int time)`) executes

(b) First the `fly` method declared at line 15 of the `Birdo class` (i.e., `public void fly(int time)`) executes and then the `fly` method declared at line 22 of the `Birdo` class (i.e., `public void fly(int time, int height)`) executes

(c) First the `fly` method declared at line 22 of the `Birdo` class (i.e., `public void fly(int time, int height)`) executes and then the `fly` method declared at line 15 of the `Birdo` class (i.e., `public void fly(int time)`) executes

(d) First the `fly` method declared at line 22 of the `Birdo` class (i.e., `public void fly(int time, int height)`) executes and then possibly the `fly` method declared at line 15 of the `Birdo` class (i.e., `public void fly(int time)`) executes

Part 9

Consider the following code:

```
1  Character c = new Mario(new ArrayList<Sport>());
2  c.setName("Mario The Stupendous");
```

When this code executes:

(a) Only the `setName` method declared in `Mario` executes

(b) Only the `setName` method declared in `SharedCharacterBehaviour` executes

(c) The `setName` method declared in `Mario` executes followed by the `setName` method declared in `SharedCharacterBehaviour`

(d) The `setName` method declared in `Mario` executes followed by the `setName` method declared in `SharedCharacterBehaviour` followed by the `setName` method declared in `Character`

Part 10

Consider the following code:

```
1 Character c = new Character();
```

Is the following statement True or False?

The line of code above will neither compiler nor execute ..... TRUE ... FALSE

Part 11

Consider the following code:

```
1 public class LittleBirdo extends Birdo {
2     public LittleBirdo() {
3         height = 1000;
4     }
5 }
```

and

```
1 Birdo b = new LittleBirdo();
```

Is the following statement True or False?

The line of code above will compile, but will execute with an exception ..... TRUE ... FALSE

Part 12

Consider that we make the following change to the `Peach` class:

```
1 public class Peach extends SharedCharacterBehaviour implements PoolSport {
2     public void getWet() {
3     }
4 }
```

and

```
1 PoolSport ps = new Peach();
```

Is the following statement True or False?

The line of code above will not compile..... TRUE ... FALSE

Part 13

Consider that we make the following change to the `Peach` class:

```
1      public void getWet() {
2      }
3
4     public void performRun() {
5     }
6  }
```

and

```
1  PoolSport ps = new Peach();
2  RunningSport rs = ps;
```

Is the following statement True or False?

The code above will execute without an error. ..... TRUE ... FALSE

11·2=22
X 2·25=·5
21.5

Question 6. [6 marks]

This question refers to the PIZZAMIDTERM Java Project. You were directed at the start of the exam where to download this code from the repository. PIZZAMIDTERM is a modified solution for Assignment 4. It is altered in one way from Assignment 4, the constructor for the `Pizza` class now takes as a parameter the `Crust` object to use.

Implement a new class, called `GlutenFreeCrust`, to support a gluten-free crust. The base cost of a gluten-free crust is 20. The surcharge of a gluten-free crust in your area is 20; you must implement the surcharge using the `Surchargeable` interface. Your new class must reuse the existing code in the PIZZA project to the maximum extent possible. We have provided for you a test method in the Java Project to help you implement this class. Write the code for the class in the space below.

```
public class GlutenFreeCrust extends Crust implements Surchargeable {

    // Constructor
    public GlutenFreeCrust() {
        super();
        name = "gluten free crust";
        cost = 20;
    }

    @Override
    public void setSurcharge(int extraCharge) {
        cost = cost + extraCharge;
    }
}
```

(1) (1) (1)

3/6