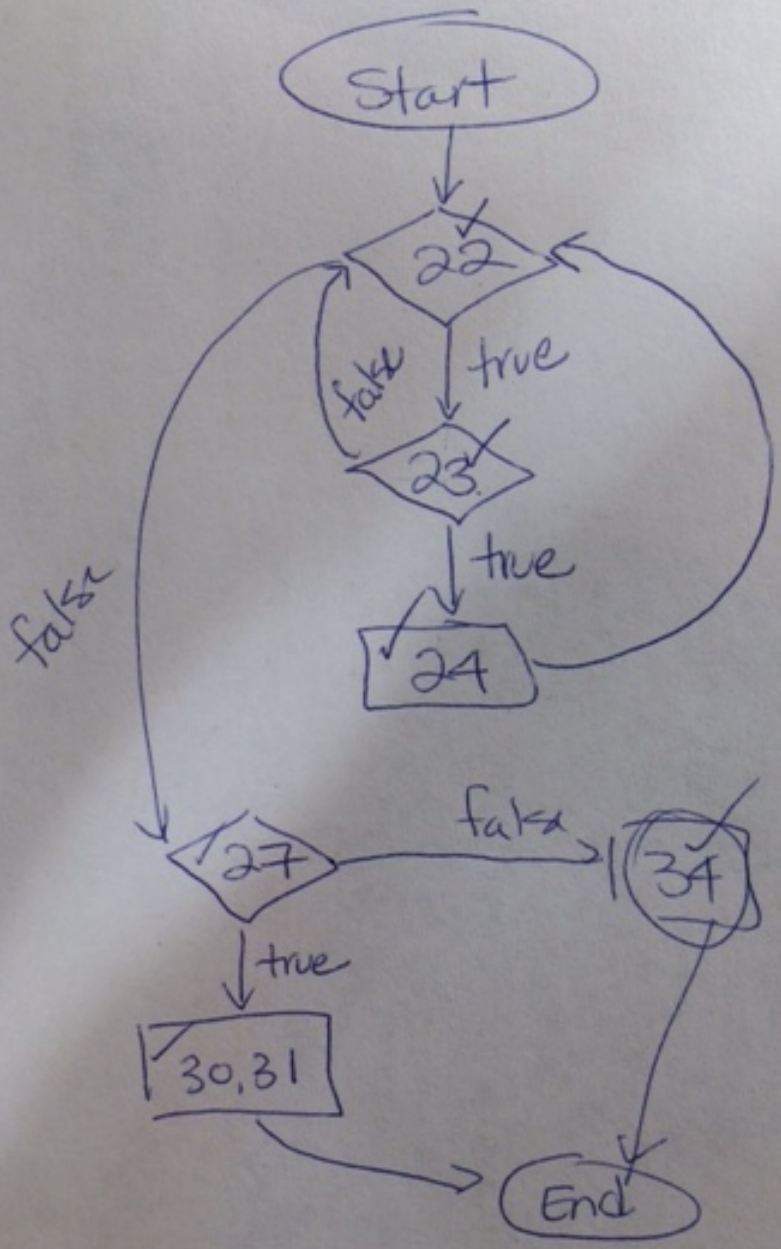
Question 1. [10 points total; 5 points for each part]

This question refers to code in the hard-copy code package that you were given.

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.

ethod control flow graph (i.e., flow chart) for the `SIGame checkGameOver` me
numbers to refer to code instead of re-writing the code in the diagram you



Part 2

Below, we have provided you with a number of empty test methods to test the `SIGame checkGameOver` method. Write the **minimal** number of test methods using Java code to ensure every line of code in the `SIGame checkGameOver` method is executed at least once. You do not need to use any `assert` methods in your tests.

```
@Test
public void textA() {

    SIGame siGame = new SIGame();
    siGame.addInvader(100,601);
    siGame.checkGameOver();

}

@Test
public void testB() {

    SIGame siGame = new SIGame();
    siGame.checkGameOver();

}
```

testA

Question 2. [10 marks total]

This question refers to code in the hard-copy package that you were given.

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
The answer is FALSE

B. [3 marks] Write a few lines of Java code to show why a `SumpPump` object is or is not substitutable for a `Pump` object.
The answer follows.

```
Pump p = new SumpPump();
// The next call is fine from the viewpoint of the Pump specification
// but unknown behaviour will result from the SumpPump that is
// expecting a value less than 500
p.openValve(750.0);
```

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
The answer is TRUE

D. [3 marks]
Write a few lines of Java code to show why a `WellWaterPump` object is or is not substitutable for a `Pump` object.
The answer follows.

```
Pump p = new WellWaterPump();
p.openValve(999.0); // Fine for both Pump and WellWaterPump
double d = p.outputPressure();
// This code expects d > 100 according to Pump and WellWater is
// guaranteed to give back d > 200
```

Question 3. [12 points]

This question refers to the UML diagram in the hard-copy code package you were given. You may also refer to the UML reference sheet in the same package.

The UML class diagram describes a University department system. Your task is to fill in below, the implementations of the associations and aggregations shown in the diagram for the `Professor`, `Course`, `Lecture`, and `LearningOutcome` classes. You need only focus on the implementation of the associations and aggregations in the diagram that involve these classes. One extra constraint not shown in the diagram is that there can not be any duplicate `Lecture`s 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.

The following is just one solution.

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

   private Course c1, c2;

   public Professor() {
       c1 = new Course();
       c1.assignProf(this);
       c2 = new Course();
       c2.assignProf(this);
   }

}

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

   private Professor prof;
   private Set<Lecture> lectures;

   public Couse() {
       prof = null;
       lectures = new HashSet<Lecture>();
   }

   public void assignProf(Professor p) {
      prof = p;
   }
```

```
    public boolean addLecture(Lecture l) {
       return lectures.add(l);
    }


}

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

   private List<LearningOutcomes> outcomes;

   public Lecture(String topic, String details) {
      outcomes = new ArrayList<LearningOutcome>();
      addLearningOutcome(topic, details);
   }

   public boolean  addLearningOutcome(String topic, String details) {
      outcomes.add(new LearningOutcome(topic, details);
   }

}

class LearningOutcome {

  // nothing needed from viewpoint of associations and aggregations

}
```

Question 4. [14 points]

This question relates to the HotelReservationSystem Java project that is available in your lab 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 in above in part 1. If you introduce any new types, you must also provide definitions and implementations of those types.

```
/ REQUIRES: amount > 0
// MODIFIES: this
// EFFECTS: if getCharges() + amount before method call < chargeLimit
//     then getChargers() == getChargeLimit() + amount
//     otherwise throw a DeclineApprovalException
public void approvePayment(double amount) throws DeclineApprovalException {
   if (charges+amount > chargeLimit)
     throw new DeclineApprovalException();
   charges = charges + amount;
}

public class DeclineApprvalException extends Exception {}
```

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.

The solution is to add an "(expected=DeclineApprovalException.class)" to the @Test line and to add a call to fail with a suitable message in the last line of the method

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

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 in part 2.

```
1  // Book a given guest into a given roo
2  // REQUIRES: guest is not null and room is not null
3  // EFFECTS: if room.isAssigned() then return false
4  //     else guest is given room and method returns true
5  public BOOLEAN bookRoom(Guest guest, Room room) {
6     if (!room.isAssigned()){
7        room.assign(guest);
8        guest.assign(room);
9        return true;
10    }
11    else
```

```
12          return false
13  }
```

Alternatively you could have used an exception such as BookedRoomException

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 in part 2.

Change line below to "assertFalse(hotel.bookRoom(guest, room));"

```
1  @Test
2  public void testAssignAGuestAnUnassignedRoom() {
3    // Hold a room, one should be available
4    Room room = hotel.holdRoom();
5    assertTrue(room != null);
6
7    // Create a guest for the room
8    Guest guest = new Guest(``Gail'', new CreditCard(500));
9
10   // Book the room
11   hotel.bookRoom(guest, room);
12   }
```

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

These questions refer to the MarioOlympicGame Java project that is available in your lab 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
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 }
```

should be addBirdo not just add

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.

Should not be any …

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, than 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
TRUE

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 compile but will execute with an exception ..... TRUE ... FALSE
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
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
FALSE

Question 6. [8 marks]

The solution to Assignment 4, the Pizza project, is available in your lab repository. This solution 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 a test for you to help implement this class. Write the code for the class in the space below.

```
1  public class GlutenFreeCrust extends Crust implements Surchargeable {
2
3          private int surcharge;
4
5          @Override
6          public void setSurcharge(int extraCharge) {
7                  surcharge = extraCharge;
8          }
9
10         public int computeCost() {
11                 return surcharge + cost;
12         }
13
14 }
```

Question 1 code.

```
1  public class SIGame {
2
3     private List<Missile> missiles;
4     private List<Invader> invaders;
5     private boolean isGameOver;
6
7     public SIGame() {
8       missiles = new ArrayList<Missile>();
9       invaders = new ArrayList<Invader>();
10      isGameOver = false;
11    }
12
13    public void addMissile(int x, int y) {
14      missiles.add(new Missile(x, y));
15    }
16
17    public void addInvader(int x, int y) {
18      invaders.add(new Invader(x,y));
19    }
20
21    private void checkGameOver() {
22       for (Invader next: invaders) {
23          if (next.getY() > 600) {
24             isGameOver = true;
25          }
26       }
27       if (isGameOver) {
28          // The clear method removes all objects from a list
29          // so the list is empty
30          missiles.clear();
31          invaders.clear();
32        }
33        else {
34          System.out.println(  The game is not over!   );
35        }
36    }
37 }
38
39 public class Missile {
40
41    private int x, y;
42
43    public Missile(int x, int y) {
44      this.x = x;
45      this.y = y;
46    }
47 }
48
49
50
51
```

```
52
53
54 public class Invader {
55
56     private int x, y;
57
58     public Invader(int x, int y) {
59       this.x = x;
60       this.y = y;
61     }
62
63     public int getY() {
64       return y;
65     }
66 }
```

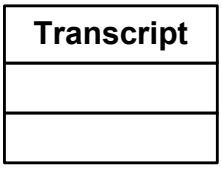
Question 2 code.

```
1  public class Pump {
2
3     protected boolean isOpen;
4
5     public Pump() {
6       isOpen = false;
7     }
8
9     // Open the valve on the pump
10    // REQUIRES: pressure < 1000.00
11    // MODIFIES: this
12    // EFFECTS: isOpen() is true
13    public void openValve(double pressure) {
14      // Assume a bunch of code for opening the valve is here
15    }
16
17    // Return the output pressure on the pump
18    // EFFECTS: returns a value > 50.5 or throws LowPressureException
19    public double outputPressure() {
20      // Assume a bunch of code reading current output pressure and
21      //returning it or throwing a LowPressureException
22    }
23
24    // Is the pump valve open?
25    // EFFECTS: Returns true if the pump valve is open and false
26    //   otherwise
27    public boolean isOpen() {
28      return isOpen;
29    }
30 }
31
32 public class SumpPump extends Pump {
33
34     // Open the valve on the pump
35     // REQUIRES: pressure < 500.00
36     // MODIFIES: this
37     // EFFECTS: isOpen() is true
38     public void openValve(double pressure) {
39        // Assume a bunch of code for opening the valve is here.
40     }
41
42    // Return the output pressure on the pump
43    // EFFECTS: returns a value > 100.0 or throws LowPressureException
44    public double outputPressure() {
45        // Assume a bunch of code for reading the current output
46        // pressure and returning it or throwing the low pressure
47        // exception
48    }
49
50 }
51
```

```
52 public class WellWaterPump extends Pump {
53
54     // Open the valve on the pump
55     // REQUIRES: pressure < 2000.00
56     // MODIFIES: this
57     // EFFECTS: isOpen() is true
58     public void openValve(double pressure) {
59         // Assume a bunch of code for opening the valve is here.
60     }
61
62    // Return the output pressure on the pump
63    // EFFECTS: returns a value > 200.0 or throws LowPressureException
64    public double outputPressure() {
65         // Assume a bunch of code for reading the current output pressure
66         // and returning it  or throwing the low pressure exception
67    }
68
69 }
```

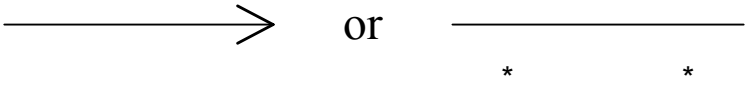
UML Class Diagram Notation

Class

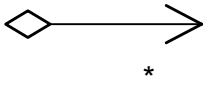


Interface

Association

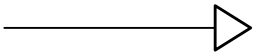


Aggregation

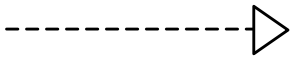


Composition

Generalization



Realizes



UML Diagram for Question 3