CSE115 **Introduction to Computer Science for Majors I** **Spring 2008**
Final Exam

First/Given Name **(PRINT)** _____ May 6, 2008

Last/Family Name **(PRINT)** _____

Person #: _____ Recitation Section: B1: T 2:00pm B2: W 10:00am
(Circle one) B3: R 12:00pm B4: F 8:00am

This book has 21 pages, make sure you have all 21.

The exam consists of 10 questions, read and answer all 10.

Do not write below this line.

Question	Part a	Part b	Part c	Part d	Part e	Part f	Part g	Part h	Part i	Part j	Subtotal
1	2	2	2	2							8
2	2	2	2	2	2	2	2	2	2	2	20
3	11										11
4	12										12
5	2.5	4.5									7
6	3	4	15								22
7	1	1	1	1							4
8	2	2	2	2	2	2					12
9	2	2									4
10	3	3	3	3	3	3	3	3			24
										Total:	124

Question 1 **[8 points]**

Fill in the code for the class named `ExamGenerator` as specified by parts a-c. Use the code you created to answer part d.

Part a) Declare an instance variable of type `Exam` in the class. You can assume these two classes are in the same package.

Part b) Inside the constructor, assign the instance variable a new instance of an `Exam`. (Note that the `Exam` constructor takes no parameters)

Part c) Create a method named `gradeExam` that has a return type of `void` and takes no parameters. Inside the method, call your instance variable's `grade` method, which also takes no parameters.

```
public class ExamGenerator {




    public ExamGenerator() {




    }









}
```

Part d) In parts a & b, you created one of the relationships we studied so far this semester, what is the formal name of that relationship?

Question 2 **[20 points]**

Use the code segment below to answer parts a - d.

```
public class Test {

      public Test() {
            new Balloon();                  //Part b
            Balloon one = new Balloon();    //Part c
            Balloon two = new Balloon();    //Part d
            two = one;                      //Part e
            Balloon three = new Balloon(); //Part f
            two = three;                    //Part g
            one = two;                      //Part h
            two = new Balloon();            //Part i
            new Balloon();                  //Part j
      }

}
```

Part a) What is the total number of balloons created when the constructor for Test is run?

For parts b – j, you need to draw the object ovals/circles and the references that will be present after each line of code is executed. Remember that objects that are no longer referred to still stay inside memory, so even if there is no reference to an object, if it was there in the previous step, it should remain in the next step.

Part b)

Part c)

Part d)

Part e)

Part f)

Part g)

Part h)

Part i)

Part j)

Question 3 **[11 points]**

For each of the statements given below, circle whether the statement is true or false.

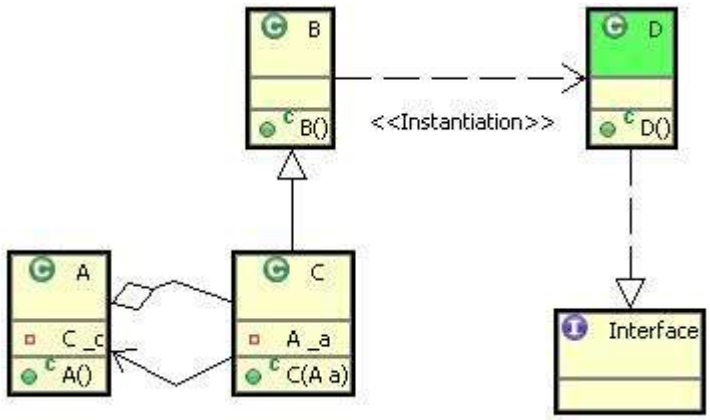
TRUE	FALSE	`void` is a valid return type for a method.
TRUE	FALSE	`void` is a valid return type for a constructor.
TRUE	FALSE	`void` is what you put in the parameter list of a method that takes no parameters.
TRUE	FALSE	`void` is what you put in the parameter list of a constructor that takes no parameters.
TRUE	FALSE	You can never have more than one parameter to a method.
TRUE	FALSE	If you define a constructor that takes more than one parameter, then you need to separate those parameters with commas in the parameter list.
TRUE	FALSE	Constructors are not methods.
TRUE	FALSE	A method definition is made up of a method header and method body.
TRUE	FALSE	When you call a method, you need to use the keyword new.
TRUE	FALSE	If the method body of a method is empty, it will never compile.
TRUE	FALSE	The lifetime of a local variable is the same as the lifetime of the object.
TRUE	FALSE	The scope of a local variable is the method it is declared in.
TRUE	FALSE	The scope of an instance variable is only within the constructor of a class.
TRUE	FALSE	The lifetime of an instance variable is the same as the lifetime of the object.
TRUE	FALSE	Accessor methods are used to change the value of an instance variable.
TRUE	FALSE	When you use a for-each loop, you are iterating over a collection using an iterator.
TRUE	FALSE	Given the two styles of entry-test loop in Java (for and while), we can say that they are interchangeable, meaning that if you can write the code using one loop, it can be re-written using the other and will still function the same.
TRUE	FALSE	Two of the mechanisms for implementing selection that are available in Java are polymorphism and if-statements.
TRUE	FALSE	If you use generics when creating your collection, then the only types of things you can insert into your collection are the type of thing specified by the generic type.
TRUE	FALSE	All instance variables should have their visibility set to public.
TRUE	FALSE	Instance variables are the way we represent properties of an object when we right the class definition.
TRUE	FALSE	When we create the composition and association relationships in code, both use an instance variable.
TRUE	FALSE	You can create an instance of an abstract class.
TRUE	FALSE	You can not create an instance of an interface.
TRUE	FALSE	A concrete class will have some methods that are abstract.
TRUE	FALSE	An interface can contain the definitions for its methods.
TRUE	FALSE	An abstract class can have instance variables.
TRUE	FALSE	An interface can inherit from one or more other interfaces.
TRUE	FALSE	An abstract class can inherit from one interface.
TRUE	FALSE	An abstract class can extend another abstract class.
TRUE	FALSE	A concrete class can extend a concrete class and an abstract class.
TRUE	FALSE	A concrete class can extend an abstract class and implement an interface.
TRUE	FALSE	An interface can implement multiple other interfaces.

Question 4 **[12 points]**

For each one of the parts of code listed in the column at the right, circle and clearly identify by number one and only one example of each term in the code on the left. If no examples of the term exist, write the words "No example" next to it.

1. package declaration
2. class header
3. interface name
4. Java keyword
5. access control modifier
6. comment
7. constructor definition
8. formal parameter list
9. argument list
10. method call
11. code that creates an instance of an object
12. non-constructor method definition
13. instance variable declaration
14. assignment statement
15. local variable name
16. explicit Call to the superclass method or constructor
17. method return type specification
18. variable whose type is that of a collection
19. method overriding (identify one of the methods)
20. method overloading (identify one of the methods)
21. loop
22. boolean expression

```
package bounce;

public class BouncingBallPair implements IBouncer {

      private BouncingBall _firstBall;
      private BouncingBall _secondBall;

      public BouncingBallPair() {
            _firstBall = new BouncingBall();
            _secondBall = new BouncingBall();
      }

      public void setColor(Color newColor) {
            _firstBall.setColor(newColor);
            _secondBall.setColor(newColor);

      }

      public void slowDown() {
            Integer speed = _firstBall.getSpeed();
            speed = speed - 4;
            _firstBall.setSpeed(speed);
            _secondBall.setSpeed(speed);
      }

      public void speedUp() {
            Integer speed = _firstBall.getSpeed();
            speed = speed + 15;
            _firstBall.setSpeed(speed);
            _secondBall.setSpeed(speed);
      }

      public void speedUp(Integer limit) {
            if(_firstBall.getSpeed() < limit) {
               _firstBall.setSpeed(limit);
            }
            _secondBall.setSpeed(limit);
      }

      public void crazy() {
            for(Integer count=1; count<=10; count++){
                  new BouncingBall();
            }

}
```



Question 5 **[7 points]**

Part a: There are five relationship arcs in the above diagram. Identify the formal name of each relationship that each arc represents. You should then indicate the direction of the relationship using the informal name. For example, if the formal name of relationship A was "Structure" and the informal name was "props up", then you would write as an answer "Structure: ClassName props up ClassName" filling in the appropriate class names.

Relationship a)

Relationship b)

Relationship c)

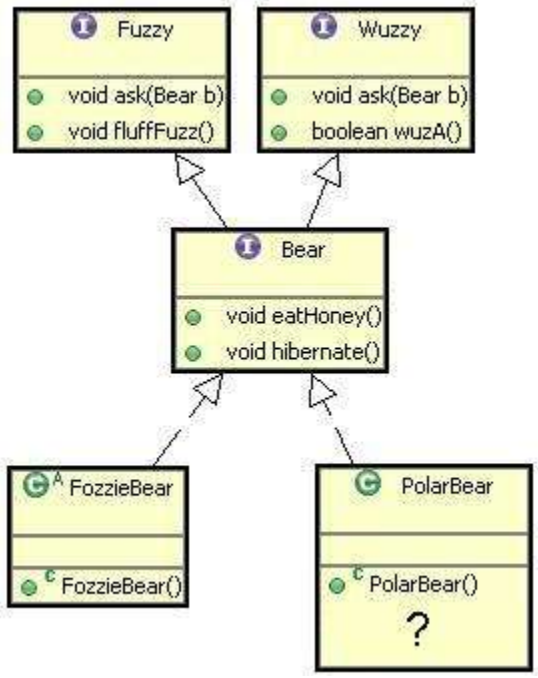
Relationship d)

Relationship e)

Part b: Write the code for the class A

Question 6 **[22 points]**

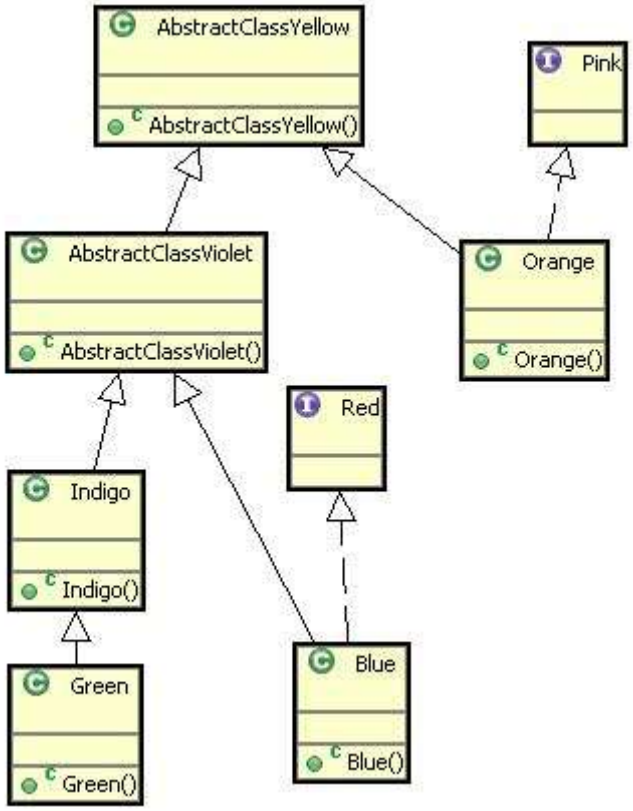
Base your answer to parts a-b on the following UML diagram.



Part a) Note the question mark in the diagram. With the question mark, PolarBear does not compile. List the names of the methods that PolarBear must have in addition to the constructor to fulfill its obligations to its relationships.

Part b) The class FozzieBear does not need any additional methods and is perfectly valid Java code – why would this be the case?

Use the following UML diagram to answer parts c-f. For parts c - f, indicate the names of all the constructors that are executed when the object specified is created.



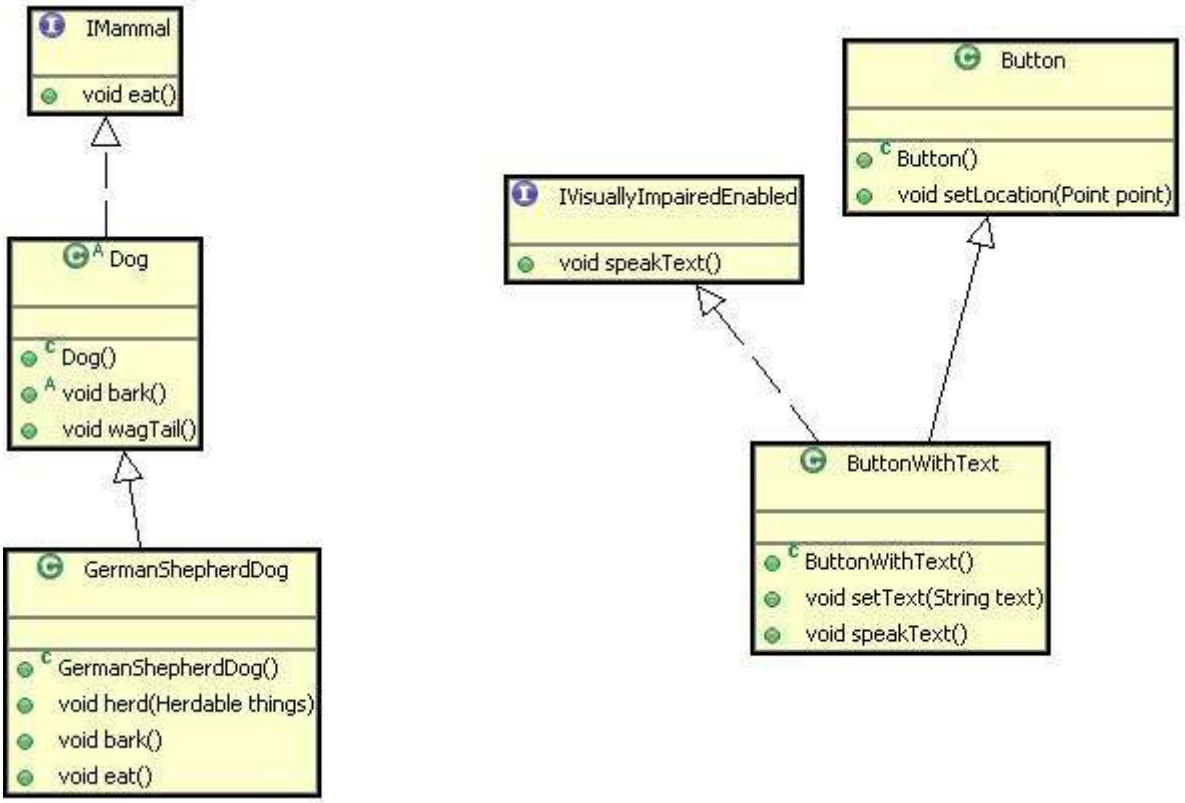
Part c) new Indigo();

Part d) new Green();

Part e) new Orange();

Part f) new Blue();

Use the following UML diagram to answer parts g – p. For parts g – p, circle if the following variable declarations/assignments would be allowed in Java. ***If*** the declaration/assignment would be allowed, circle the name of the class whose method would be executed when the subsequent method call was made. Please note that even those these classes appear in the same diagram, they are not from the same programs, they are simply combined together for space purposes.




```
g) Dog d = new IMammal();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
d.eat();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.bark();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.wagTail();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.herd(…);
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
h) IMammal d = new Dog();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
d.eat();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.bark();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.wagTail();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.herd(…);
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
i) IMammal d = new GermanShepherdDog();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
d.eat();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.bark();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.wagTail();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.herd(…);
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
j) Dog d = new GermanShepherdDog();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
d.eat();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.bark();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.wagTail();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.herd(…);
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
k) GermanShepherdDog d = new GermanShepherdDog();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
d.eat();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.bark();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.wagTail();
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
d.herd(…);
```
Class whose method would be executed: IMammal Dog GermanShepherdDog Can't call

```
l) IVisuallyImpairedEnabled b = new Button();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
b.setLocation(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.setText(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.speakText();
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
m) IVisuallyImpairedEnabled b = new ButtonWithText();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
b.setLocation(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.setText(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.speakText();
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
n) Button b = new ButtonWithText();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
b.setLocation(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.setText(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.speakText();
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
o) ButtonWithText b = new Button();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
b.setLocation(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.setText(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.speakText();
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
p) Button b = new Button();
```

Valid Declaration/Assignment Invalid Declaration/Assignment

```
b.setLocation(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.setText(…);
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

```
b.speakText();
```
Class whose method would be executed:
IVisuallyImpairedEnabled Button ButtonWithText Can't call

Question 7 **[4 points]**

For each part a – d, indicate which code blocks would be executed based on the given truth values of the Boolean expressions.

Part a)

```
public void partA() {
   if(booleanExp1) {
     //A
   }
   //B
}
```

If `booleanExp1` was true, indicate all the code blocks that would be executed.

□ A □ B □ None

Part b)

```
public void partB() {
   if(booleanExp1) {
      //A
   }
   else {
      //B
   }
}
```

If `booleanExp1` was false, indicate all the code blocks that would be executed.

□ A □ B □ None

Part c)

```
public void partC() {
   if(booleanExp1) {
      //A
   }
   else if (BooleanExp2) {
      //B
   }
   else {
      //C
   }
}
```

If `booleanExp1` was true and `booleanExp2` was true, indicate all the code blocks that would be executed.

□ A □ B □ C □ None

Part d)

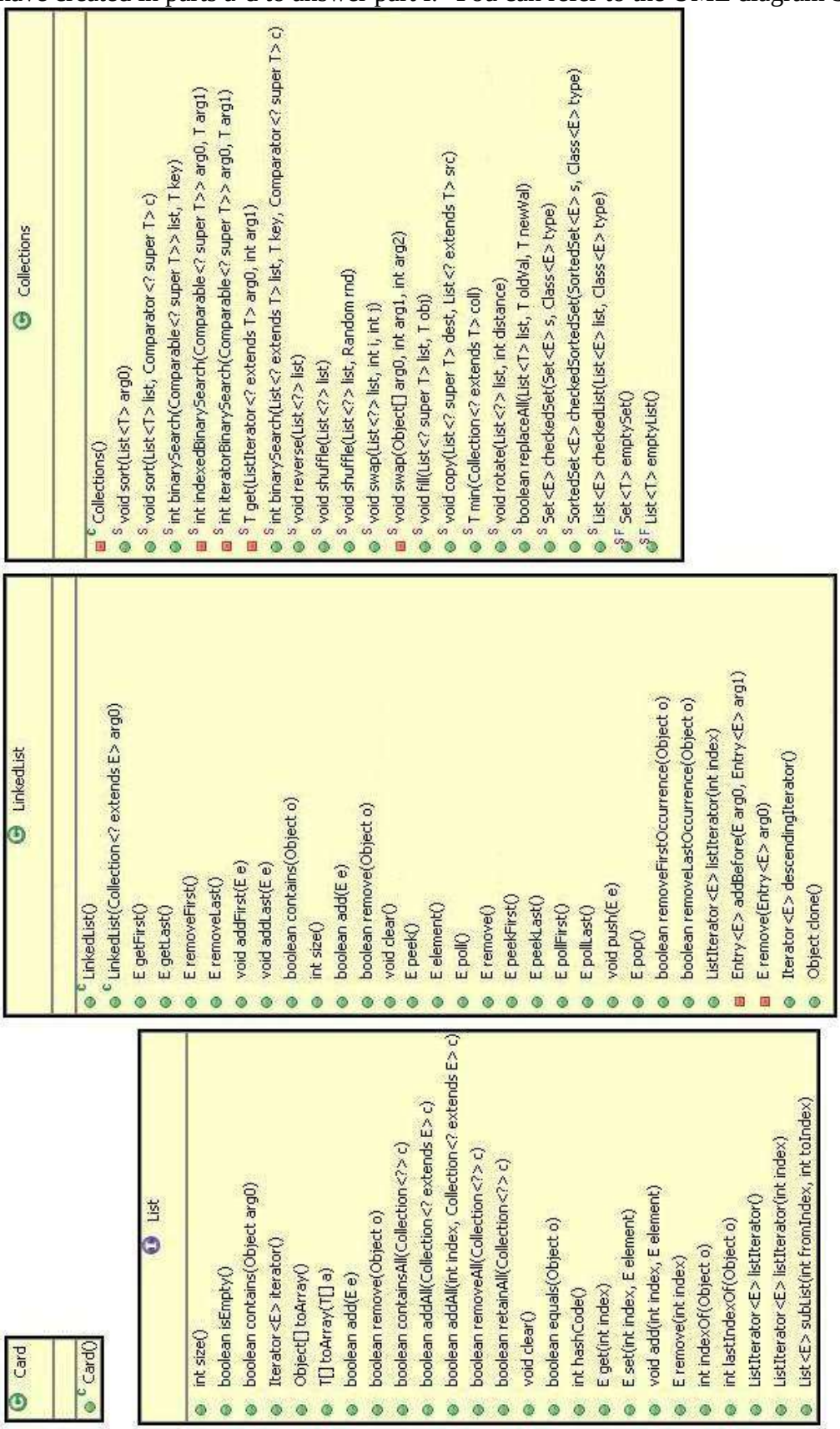
```
public void partD() {
   if(booleanExp1) {
      if(booleanExp2) {
         //A
      }
      else {
         //B
      }
      //C
   }
   else {
      //D
      if(booleanExp3){
         //E
      }
      //F
   }
}
```

If `booleanExp1` was false and `booleanExp2` was false and `booleanExp3` was true, indicate all the code blocks that would be executed.

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ None

Question 8 **[12 points]**

For this question, you will fill in the body of the Deck class as described in parts a – e. You will use what you have created in parts a-d to answer part f. You can refer to the UML diagram below as you work.



Part a) Declare an instance variable for the `Deck` class whose type is `List` where the list holds onto `Card` objects.

Part b) The first line of the constructor of the `Deck` should create an instance of a `LinkedList` and assign it to the variable you declared in part a.

Part c) The rest of the constructor should create 10 cards and place each of them in the list. Note that the `Card` class constructor does not take any parameters.

Part d) Create an accessor that returns the list of cards that you created in parts a-c.

Part e) Create a method called `shuffle` that does not return anything and shuffles the cards in the `List` that you created in parts a-c.

```
public class Deck {




      public Deck() {









      }









}
```

Part f)

Fill in the return type, formal parameter list and the body of the method called `deal` given below. Deal should take in a deck of cards, shuffle them, and then for each card in that deck, call the card's `deal()` method.

```
public                deal(                                ) {




}
```

Question 9 **[4 points]**

Part a) Assume for this question that you are modeling a standard deck of playing cards. If you are unfamiliar, a standard deck has 52 cards, each of which has one of four suits (heart, diamond, spade, club) and one of thirteen values (2 through 10, Jack, Queen, King, Ace). Suppose that you need to write an equals method for the Card class. What could it mean for two cards to be equal? Feel free to talk about as many reasonable possibilities as you would like.

Part b) What if you were allowed to consider two decks of 52 playing cards? How could that change your answer to part a?

Question 10 **[24 points]**

Answer each of the following questions in the space provided.

Part a) What is a variable?

Part b) What is a variable used for?

Part c) What is the difference between a local variable and an instance variable?

Part d) How do you know when a method you are writing needs to take in a parameter?

Part e) What is the difference between a class and an object?

Part f) Explain the difference between the association relationship and the composition relationship.

Part g) Why would you use selection in a program?

Part h) Why would you use a loop in a program?