UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Engineering, Built Environment and IT
Department of Computer Science

# Software Modelling
# COS 121

Examination Opportunity 2 (EO2)

26 October 2016

**Name (Initials and Surname):** ______________________________

**Student number:** ______________________________

## Examiners

**Internal:** Dr Linda Marshall, Mr Marius Riekert and Ms Renette Ros

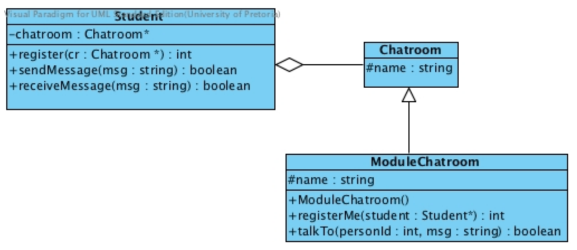
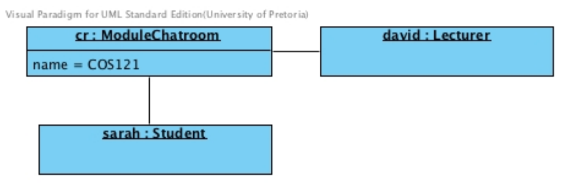
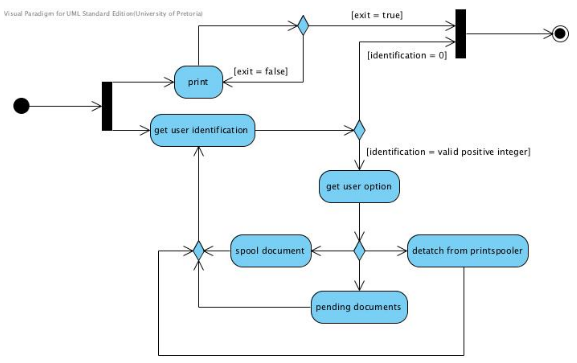
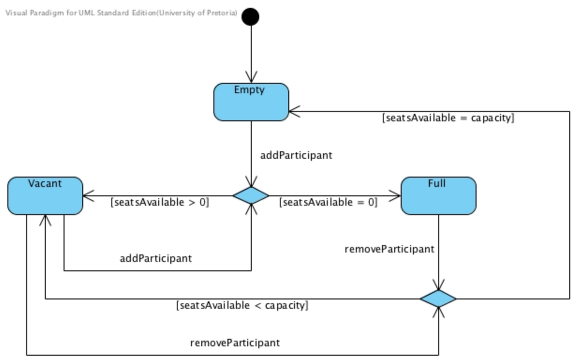
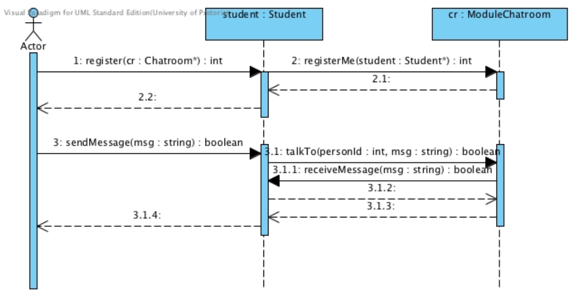
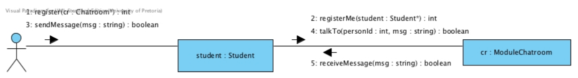
## Instructions

1. Read the question paper carefully and answer all the questions on the multiple choice answer sheet provided.
2. The assessment opportunity comprises of **46** questions on **22** pages.
3. You have **50** minutes to complete the paper.
4. Make sure you write you name on the question paper and hand it in with your answer sheet.
5. The answer sheet instructions are on the sheet. Make sure you use a pencil to fill it in and use **Side B** of the answer sheet.
6. Please switch off your cell phone, and keep it off for the duration of the paper.

**Full marks is: 45**

## Programming Tools

1. What does the acronym IDE stand for? (1)
   - A. Interactive Design Executable
   - B. Integrated Development Environment
   - C. Iterative Design Exercise
   - D. Interactive Development Environment
   - E. Integrated Design Environment
2. What is Valgrind? (1)
   - A. A Design Pattern
   - B. A programming language
   - C. A collection of memory debugging tools
   - D. A Documentation Tool
   - E. An IDE
   - F. A compiler
   - G. A debugger
3. What is GDB? (1)
   - A. A Design Pattern
   - B. A programming language
   - C. A collection of memory debugging tools
   - D. A Documentation Tool
   - E. An IDE
   - F. A compiler
   - G. A debugger
4. Which tool is best suited to finding the origin of a segmentation fault? (1)
   - A. Valgrind
   - B. GDB
   - C. Doxygen
   - D. IDE
   - E. G++
   - F. Symbol Table
5. Which of the following is **not** an IDE that can be used for C++ development? (1)
   - A. Visual Studio
   - B. Clion
   - C. Eclipse
   - D. Code::Blocks
   - E. GDB
6. Which of the following is not a GDB command? (1)
   - A. help
   - B. run
   - C. check
   - D. break
   - E. watch
   - F. continue

| | |
|---|---|
|  (i) |  (ii) |
|  (iii) |  (iv) |
|  (v) |  (vi) |

Student
-chatroom : Chatroom*
+register(cr : Chatroom *) : int
+sendMessage(msg : string) : boolean
+receiveMessage(msg : string) : boolean
Chatroom
#name : string
ModuleChatroom
#name : string
+ModuleChatroom()
+registerMe(student : Student*) : int
+talkTo(personId : int, msg : string) : boolean

cr : ModuleChatroom
name = COS121
david : Lecturer
sarah : Student

[exit = true]
[identification = 0]
[exit = false]
print
get user identification
[identification = valid positive integer]
get user option
spool document
detatch from printspooler
pending documents

Empty
[seatsAvailable = capacity]
addParticipant
Vacant
[seatsAvailable > 0]
[seatsAvailable = 0]
Full
addParticipant
removeParticipant
[seatsAvailable < capacity]
removeParticipant

Actor
student : Student
cr : ModuleChatroom
1: register(cr : Chatroom*) : int
2: registerMe(student : Student*) : int
2.1:
2.2:
3: sendMessage(msg : string) : boolean
3.1: talkTo(personId : int, msg : string) : boolean
3.1.1: receiveMessage(msg : string) : boolean
3.1.2:
3.1.3:
3.1.4:

1: register(cr : Chatroom*) : int
3: sendMessage(msg : string) : boolean
2: registerMe(student : Student*) : int
4: talkTo(personId : int, msg : string) : boolean
student : Student
cr : ModuleChatroom
5: receiveMessage(msg : string) : boolean

Table 1: UML diagrams

## UML Notation

### Questions 7 to 13 relate to the UML diagrams given in Table 1

7. Which UML diagram had not been presented prior to the classes moving to an online format using the discussion board? [Note: it has subsequently been presented and therefore forms part of the scope of this test by a process of elimination.] (1)

   A. (i)
   B. (ii)
   C. (iii)
   D. (iv)
   E. (v)
   F. (vi)

8. Which diagrams are classified in the UML specification as structural? (1)

   A. (i) only
   B. (ii) only
   C. (i) and (ii)
   D. (iii) and (iv)
   E. (v) and (vi)

9. Which diagrams are classified in the UML specification as behavioural? (1)

   A. (i) and (ii)
   B. (iii) and (iv)
   C. (v) and (vi)
   D. (i), (ii), (iii) and (iv)
   E. (iii), (iv), (v) and (vi)
   F. All of the diagrams
   G. None of the diagrams

10. Which diagram does not represent objects or their interaction? (1)

   A. (i)
   B. (ii)
   C. (iii)
   D. (iv)
   E. (v)
   F. (vi)

11. A sequence diagram is given in: (1)

   A. (i)
   B. (ii)
   C. (iii)
   D. (iv)
   E. (v)
   F. (vi)

12. There is a subtle difference between how an activity diagram and a state diagram is drawn. Identify the state diagram. (1)

   A. (i)
   B. (ii)
   C. (iii)
   D. (iv)
   E. (v)

F. (vi)

13. Diagram ... is a representation of an object diagram. (1)

A. (i)

B. (ii)

C. (iii)

D. (iv)

E. (v)

F. (vi)

**Questions 14 to 18 relate to the partial code given below and the UML diagram given in Figure 1.**

```
class Chatter { // Colleague
  public:
    Chatter();
    virtual void receiveMessage(string) = 0;
    virtual void sendMessage() = 0;
    void reg(Chatroom*);
    void leave();
    virtual ~Chatter();
  protected:
    Chatroom* chatroom;
    int       myId;
};

class Student : public Chatter {
  public:
    Student();
    virtual void receiveMessage(string);
    virtual void sendMessage();
    virtual ~Student();
};

void Chatter::reg(Chatroom* cr){
  chatroom = cr;
  myId = chatroom->registerMe(this);
}

void Student::sendMessage(){
  string toId;
  string msg;
  cout<<"Student "<<myId<<" send message to? ";
  getline(cin,toId,'\n');
  //cin>>toIdstring;
  cout<<"Student "<<myId<<" message? ";
  getline(cin,msg,'\n');
  //cin>>msg;
  ostringstream convert;
  convert << myId;
  chatroom->talkTo(atoi(toId.c_str()),convert.str()+": "+msg);
}

bool Chatroom::talkTo(int id, string msg){  // From which ModuleChatroom inherits
  // Must check is the participant is still registered before sending the message.
  vector<Participant*>::iterator it;
  bool found = false;

  it = participant.begin();
  while ((it != participant.end()) && (!found)) {
    if ((*it)->id == id) {
```

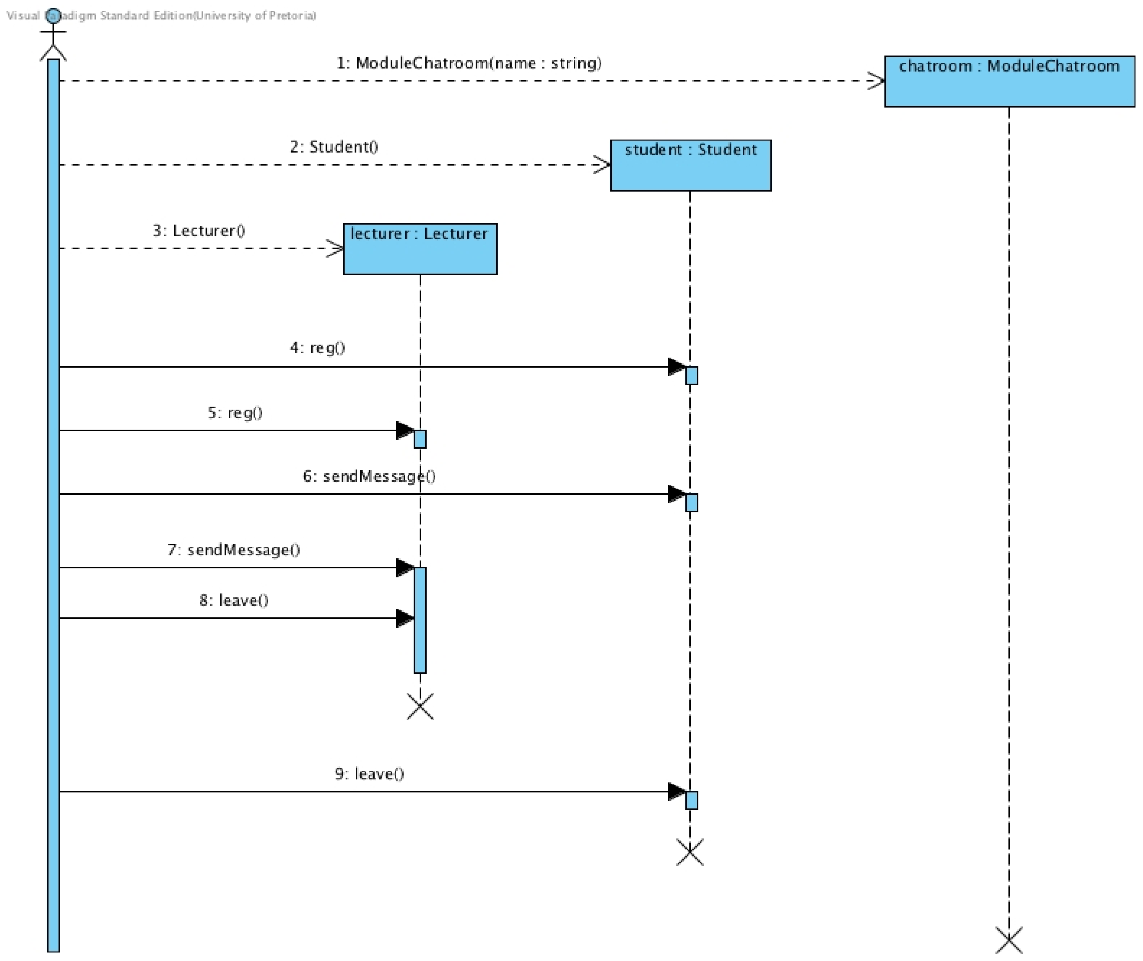


Figure 1: UML diagram showing interaction between instances of classes making up the mediator pattern

```
            found = true;
            (*it)->chatter->receiveMessage(msg);
        } else {
            it++;
        }
    }
    return found;
}
```

14. Which of the following statements is an accurate depiction of sequence 1. on the figure? (1)
    - A. `Chatroom chatroom("COS121");`
    - B. `Chatroom* chatroom = new Chatroom("COS121");`
    - C. `Chatroom* chatroom = new ModuleChatroom("COS121");`
    - D. All of the above.
    - E. None of the above.

15. Before sequence 5. is called, sequence 4. interacts with which object and which feature is called? (1)
    - A. Object `lecturer` and feature `talkTo`.
    - B. Object `student` and feature `talkTo`.
    - C. Object `chatroom` and feature `talkTo`.
    - D. Object `lecturer` and feature `reg`.
    - E. Object `student` and feature `reg`.
    - F. Object `chatroom` and feature `reg`.

16. The call to `sendMessage` is followed by a call to which of the follow? (1)
    - A. Object `lecturer` and feature `talkTo`.

B. Object `student` and feature `talkTo`.

C. Object `chatroom` and feature `talkTo`.

D. Object `lecturer` and feature `reg`.

E. Object `student` and feature `reg`.

F. Object `chatroom` and feature `reg`.

17. A call to `lecturer`'s `receiveMessage` takes place after sequence 6. following a call to ... . (1)

    A. `chatroom`'s `receiveMessage`

    B. `chatroom`'s `sendMessage`

    C. `chatroom`'s `reg`

    D. `chatroom`'s `leave`

    E. `chatroom`'s `talkTo`

18. Which statements are most likely to have been called after sequence 9? (1)

    A. `student->leave();` and `lecturer->leave()`

    B. `student->leave();` and `delete student;`

    C. `lecturer->leave();` and `delete lecturer;`

    D. `chatroom->leave();` and `delete chatroom;`

    E. `delete student;` and `delete chatroom;`

## Design Patterns

### Questions 19 to 28 relate to the following scenario.

Figure 2 shows the UML class diagram for a logic circuit simulation system. A circuit consists of `Widgets` connected by `Connectors`. The system supports four different types of widgets: input widgets, output widgets, logic gates and complex widgets built from other widgets. A circuit has different views that are automatically updated when the circuit changes. Additionally the system supports undoable operations.

19. Which creational pattern is present in this system? (1)

    A. Factory Method

    B. Adapter

    C. Prototype

    D. Decorator

    E. Composite

    F. Singleton

    G. Iterator

    H. Abstract Factory

    I. Command

    J. Builder

20. Which classes are part of the answer in 19? (1)

    A. `Widget` hierarchy

    B. `View` hierarchy

    C. `WidgetAction` hierarchy

    D. `EventAction` hierarchy

    E. `Snapshot`

    F. `Connector` hierarchy

    G. `EventSystem`

    H. `CircuitDiagram`

21. The composite pattern is present in this system. What classes are part of it? (1)

    A. All the Widget classes

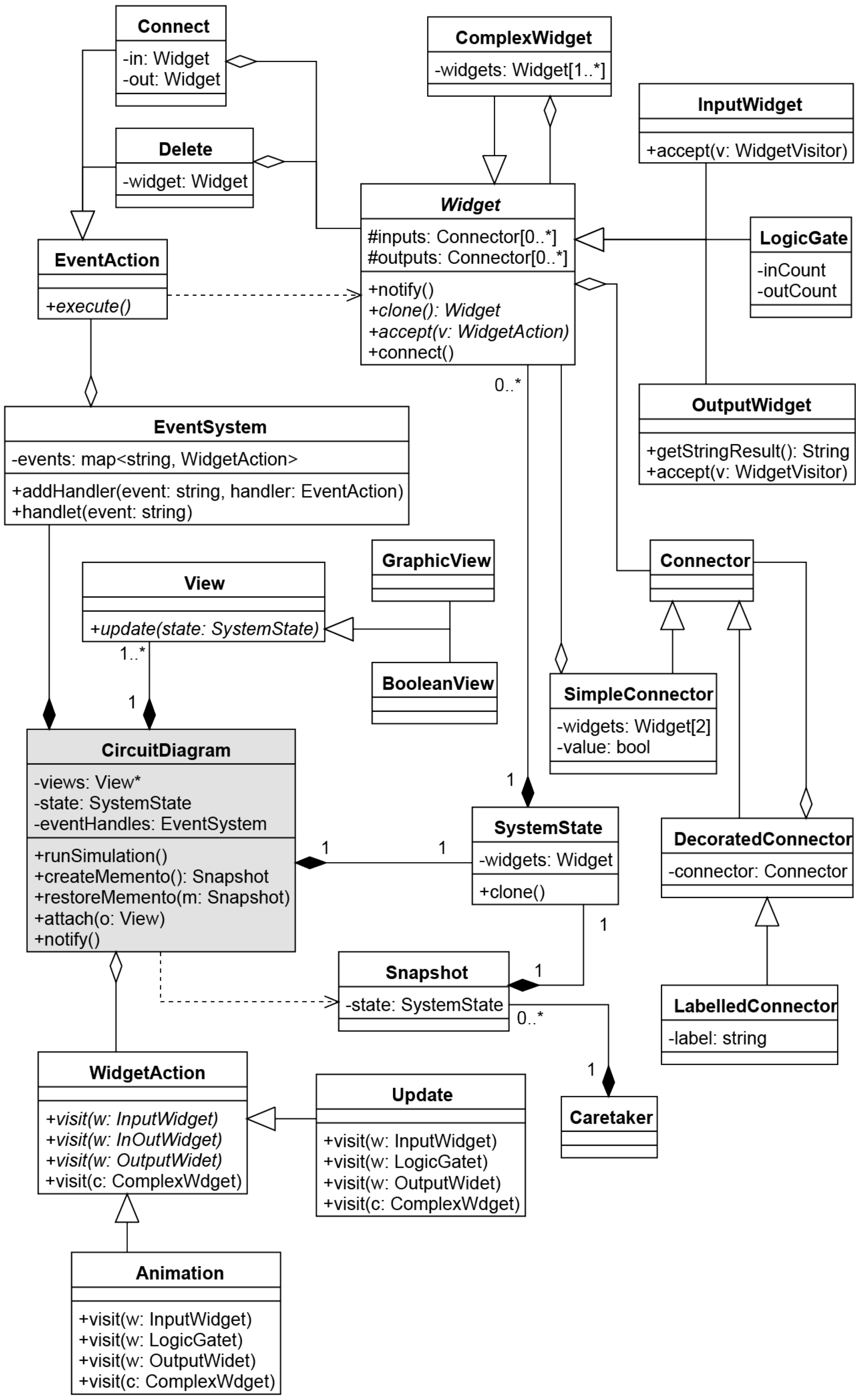


Figure 2: The Logic Circuit simulator

B. All the connector classes
C. `CircuitDiagram` and `WidgetAction`
D. `Widget`, `Connector` and `SimpleConnector`
E. `Widget` and the `EventActions`

22. Which design pattern does the `EventSystem` use to handle events? (1)
    A. Memento
    B. Prototype
    C. Factory Method
    D. Visitor
    E. Command
    F. Adapter
    G. Mediator
    H. Decorator
    I. Composite
    J. Interpreter

23. Identify the Component from the Decorator Pattern (1)
    A. EventSystem
    B. View
    C. Widget
    D. CircuitDiagram
    E. Snapshot
    F. ComplexWidget
    G. SimpleConnector
    H. Connector

24. Identify the Composite class (1)
    A. EventSystem
    B. View
    C. Widget
    D. CircuitDiagram
    E. Snapshot
    F. ComplexWidget
    G. SimpleConnector
    H. Connector

25. Which pattern does `WidgetAction` and its subclasses form part of? (1)
    A. Prototype
    B. Factory Method
    C. Visitor
    D. Command
    E. Adapter
    F. Mediator
    G. Iterator
    H. Composite
    I. Interpreter

26. What other class(es) are also part of the design pattern referenced in 25? (1)
    A. Widgets and `SystemState`
    B. `CircuitDiagram` and `SystemState`
    C. Widgets

D. `CircuitDiagram`
E. `SystemState`
F. Widgets and Connectors
G. Event Actions

27. Which roles does the `CircuitDiagram` fulfill? (1)
    A. Component
    B. Composite
    C. Product
    D. Prototype
    E. Decorator
    F. Adapter
    G. Aggregate
    H. Originator
    I. Invoker
    J. Mediator

28. What pattern is the `Snapshot` and `Caretaker` classes part of? (1)
    A. Prototype
    B. Factory Method
    C. Visitor
    D. Command
    E. Adapter
    F. Memento
    G. Decorator
    H. Composite
    I. Interpreter

### Questions 29 to 32 relate to the following code

```
class RectangleA {
public:
   virtual void draw() = 0;
};

class RectangleB {
public:
   RectangleB( Coordinate x1, Coordinate y1,
         Coordinate x2, Coordinate y2 ) {
      x1_ = x1;  y1_ = y1;  x2_ = x2;  y2_ = y2;
      cout << "RectangleB:  create.  ("
           << x1_ << "," << y1_
           << ") => (" << x2_ << ","
           << y2_ << ")" << endl; }
   void oldDraw() {
      cout << "RectangleB:  oldDraw.  ("
           << x1_ << "," << y1_
           << ") => (" << x2_ << ","
           << y2_ << ")" << endl; }
private:
   Coordinate x1_;
   Coordinate y1_;
   Coordinate x2_;
   Coordinate y2_;
};

class RectangleC : public RectangleA,
                    private RectangleB {
public:
   RectangleC( Coordinate x, Coordinate y,
                           Dimension w, Dimension h )
               : RectangleB( x, y, x+w, y+h ) {

    cout << "RectangleC: create.  (" << x
           << "," << y
           << "), width = " << w
           << ", height = " << h << endl; }
   virtual void draw() {
      cout << "RectangleC: draw." << endl;
      oldDraw(); }
};
```

29. The given code is an example of the ... design pattern. (1)
    - A. Template Method
    - B. Factory Method
    - C. Mediator
    - D. Adapter
    - E. Iterator

30. Instead of `RectangleC` inheriting *privately* from `RectangleB`, the pattern can be applied as follows: (1)
    - A. `RectangleC` inheriting publicly from only `RectangleB`
    - B. `RectangleC` inheriting publicly from `RectangleB` as well as from `RectangleA`
    - C. `RectangleC` inheriting privately from only `RectangleB`
    - D. `RectangleC` inheriting privately from `RectangleB` as well as from `RectangleA`
    - E. `RectangleC` inheriting publicly from only `RectangleA`
    - F. `RectangleC` inheriting publicly from `RectangleA` and have a private feature defined as `RectangleA *rectangle;`
    - G. `RectangleC` inheriting publicly from `RectangleA` and have a private feature defined as `RectangleB *rectangle;`
    - H. `RectangleC` inheriting publicly from `RectangleA` and have a private feature defined as `RectangleC *rectangle;`

31. Assuming the correct change in Question 30 has been applied, which other lines of `RectangleC` need to change? (1)
    - A. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB`.
    - B. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB` and add `rectangle = new RectangleB` to the first line of the body of the constructor.
    - C. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB` and add `rectangle = new RectangleB(x,y,w,h)` to the first line of the body of the constructor.
    - D. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB` and add `rectangle = new RectangleB(x, y, x+w, y+h)` to the first line of the body of the constructor.
    - E. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB`, add `rectangle = new RectangleB(x, y, x+w, y+h)` to the first line of the body of the constructor and change the call to `oldDraw` to `rectangle->oldDraw()`.
    - F. The constructor of `RectangleC`. Remove the member list initialisation call to the constructor of `RectangleB`, add `rectangle = new RectangleB(x, y, x+w, y+h)` to the first line of the body of the constructor and change the call to `oldDraw` to `rectangle.oldDraw()`

32. Assuming the correct change in Question 30 has been applied, what changes are necessary for classes `RectangleA` and `RectangleB`? (1)
    - A. `RectangleA` is no longer necessary and can be removed.
    - B. `RectangleB` is no longer necessary and can be removed.
    - C. Both `RectangleA` and `RectangleB` can be removed.
    - D. `RectangleA` and `RectangleB` stay the same.
    - E. texttRectangleA changes and `RectangleB` stay the same.
    - F. texttRectangleB changes and `RectangleA` stay the same.

## Questions 33 to 36 relate to the following scenario.

A new strategy game is currently under development. The developer has created three classes representing units, namely `Ghost`, `Marine` and `Firebat`. All of these classes inherit from an abstract class `Unit`. Having this arrangement will not only make it easy to handle units uniformly throughout the code, but will also allow for the easy addition of more units, like for example, if an expansion pack is created for the game at a later date. All types of units have their own "standard stats". These stats may change once upgrades are applied during the course of the game afterwhich the newly created applicable units will have new "standard stats" as dictated by the upgrade. There is also a class named `Barracks` which is able to create and return units of any type upon request. This class encapsulates three objects, one of each type of unit.

33. Which of the following options is most appropriate to describe the `Firebat`? (1)
    - A. Implements an operation for cloning itself.
    - B. Creates a new object by asking the prototype manager to ask a selected prototype to clone itself.
    - C. Declares an interface for cloning itself.
    - D. Declares the factory method which returns a product object.
    - E. Defines the product interface for the factory method to create.
    - F. Implements the interface for the product.
    - G. Is responsible for keeping prototypes and provide functionality to add and remove prototypes.
    - H. Implements the abstract operations that produce product objects that are created by the corresponding ConcreteFactory

34. Which of the following options is most appropriate to describe the `Barracks`? (1)
    - A. Implements an operation for cloning itself.
    - B. Creates a new object by asking the prototype manager to ask a selected prototype to clone itself.
    - C. Declares an interface for cloning itself.
    - D. Declares the factory method which returns a product object.
    - E. Defines the product interface for the factory method to create.
    - F. Implements the interface for the product.
    - G. Is responsible for keeping prototypes and provide functionality to add and remove prototypes.
    - H. Implements the abstract operations that produce product objects that are created by the corresponding ConcreteFactory

35. Which of the following options is most appropriate to describe the `Unit`? (1)
    - A. Implements an operation for cloning itself.
    - B. Creates a new object by asking the prototype manager to ask a selected prototype to clone itself.
    - C. Declares an interface for cloning itself.
    - D. Declares the factory method which returns a product object.
    - E. Defines the product interface for the factory method to create.
    - F. Implements the interface for the product.
    - G. Is responsible for keeping prototypes and provide functionality to add and remove prototypes.
    - H. Implements the abstract operations that produce product objects that are created by the corresponding ConcreteFactory

36. Select from the following options the intent of the pattern: (1)
    - A. Provide an interface for creating families of related or dependent objects without specifying the concrete classes.
    - B. Without violating encapsulation, capture and externalise an object's internal state so that the object can be restored to this state later.
    - C. Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
    - D. Define the skeleton of an algorithm in an operation, deferring some steps to sub- classes. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
    - E. Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

F. Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

G. Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

H. Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

## Questions 37 to 39 relate to the following scenario.

A new strategy game has been released. During development, the developers created two concrete classes representing vehicles, namely `Ornithopter` and `Devastator`. Both of these classes inherit from an abstract class `Vehicle`. Another class named `ConstructionYard`, which is able to create units of any type upon request, was also created. This class encapsulates objects, one of type `Ornithopter` and one of type `Devastator`. The `Vehicle` class' definition is as follows:

```
class Vehicle
{
        public:
                Vehicle();
                virtual ~Vehicle();
                virtual int fire() = 0;
                virtual void takeDamage(int) = 0;
                virtual void drawUnit() = 0;

        protected:
                int damage;//The amount of damage a vehicle can take

};
```

The gaming company has decided to develop an expansion pack. The expansion pack will include a new type of vehicle called `SiegeTank`. In addition, two upgrades will be available for all vehicles. These are an armor reinforcement, which will increase the amount of damage a vehicle can receive, and a rapid fire upgrade, which will allow a vehicle to fire multiple times in quick succession. These upgrades are "purchased" with the game's currency while playing. Once purchased in a level, the upgrades are applied to all units constructed after the purchase. The upgrades are reset at the start of a subsequent level.

With respect to the expansion pack, answer the following questions:

37. Which of the following class definitions is most appropriate minimal class definition for the `SiegeTank` class? (1)

A.
```
class SiegeTank  : public Vehicle
{
        public:
                SiegeTank();
                virtual ~SiegeTank();
                virtual int fire();
                virtual void takeDamage(int);
                virtual void drawUnit();

        protected:
                int damage;
};
```

B.
```
class SiegeTank
{
        public:
                SiegeTank();
                virtual ~SiegeTank();
                virtual int fire();
                virtual void takeDamage(int);
                virtual void drawUnit();

        protected:
                int damage;
};
```

C.
```
class SiegeTank : public Vehicle
{
        public:
                SiegeTank();
                virtual ~SiegeTank();
                virtual int fire();
                virtual void takeDamage(int);
};
```

D.
```
class SiegeTank : public Vehicle
{
        public:
                SiegeTank();
                virtual ~SiegeTank();
                virtual int fire();
                virtual void takeDamage(int);
                virtual void drawUnit();

        protected:
                int damage;
};
```

38. Suppose an `Upgrade` class was created to represent all of the subsequent concrete upgrades which can be applied to vehicles. Select the most appropriate option for a class definition for the `Upgrade` class: (1)

A.
```
class Upgrade
{
        public:
                Upgrade();
                virtual ~Upgrade();
                virtual int fire() = 0;
                virtual void takeDamage(int) = 0;
                virtual void drawUnit() = 0;

        protected:
                int damage;

};
```

B.
```
class Upgrade : public Vehicle
{
        public:
                Upgrade(Vehicle*);
                virtual ~Upgrade();

        private:
                Vehicle* vehicle;

};
```

C.
```
class Upgrade : public Vehicle
{
        public:
                Upgrade(Vehicle*);
                virtual ~Upgrade();
                virtual int fire();
                virtual void takeDamage(int);
                virtual void drawUnit();

        private:
                Vehicle* vehicle;

};
```

D.
```
class Upgrade : public Vehicle
{
        public:
                Upgrade(Vehicle*);
                virtual ~Upgrade();

        protected:
                Vehicle* vehicle;

};
```

E. All of the options are valid.

F. Both `A` and `B` are valid.

G. Both `C` and `D` are valid.

H. `B`, `C` and `D` are all valid.

39. Select from the following options the intent of the pattern: (1)

A. Provide an interface for creating families of related or dependent objects without specifying the concrete classes.

B. Without violating encapsulation, capture and externalise an object's internal state so that the object can be restored to this state later.

C. Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

D. Define the skeleton of an algorithm in an operation, deferring some steps to sub- classes. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithms structure.

E. Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

F. Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

G. Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

H. Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

## Questions 40 to 43 relate to the following scenario.

Assume that a program which will support Boolean algebra is currently under development. The first order of business is to come up with a way to construct Boolean expressions. The following classes have thus far been proposed by the development team, each with a descriptive class name indicating its functionality:

- `Expression`, which is an abstract class.
- `BinaryOperator`, which inherits from `Expression`.
- `Negate`, which inherits from expression.
- `And` and `Or`, which both inherit from BinaryOperator.
- `Variable`, which inherits from Expression and represents a single variable. This class also encapsulates the value of the variable that it represents.
- `Constant`, which inherits from Expression.

40. Suppose all classes representing binary operations have a single constructor which takes two pointers to `Expression` objects, where the first parameter represents the left hand side of the operator and the second the right hand side. Assume further that the constructor of a class representing a unary operator takes as a parameter a pointer to an Expression object, which is its only operand. The `Variable` class' constructor accepts a single character which represents the name of the variable. Consider the following Boolean expression , where `x` and `y` are variables: (1)

```
(!x && y) || (x && !y)
```

Identify from the following options a valid statement to construct the given expression:

A. Expression* e = new Or(And(new Variable('y'),Negate(new Variable('x'))),And(new Variable('x'),new Negate(new Variable('y'))))

B. Expression* e = new Or(And(new Variable('y'),Negate(new Variable('x'))),And(new Negate(new Variable('y'),new Variable('x'))))

C. Expression* e = new Or(And(Negate(new Variable('x')),new Variable('y')),And(new Variable('x'),new Negate(new Variable('y'))))

D. Expression* e = new And(Or(Negate(new Variable('x')),new Variable('y')),Or(new Variable('x'),new Negate(new Variable('y'))))

E. Expression* e = new Or(And(Negate(new Variable('x')),new Variable('y')),Or(new Variable('x'),new Negate(new Variable('y'))))

F. Expression* e = new Or(And(new Variable('x'),new Variable('y')),Or(new Variable('x'),new Variable('y')))

41. Which of the following statements is most true for the `And` class? (1)

A. Implements an Interpret operation for nonterminal symbols in the grammar. Interpret typically calls itself recursively on the variables representing R1 through Rn.

B. Contains children that are either composites or leaves.

C. Do not have children, define the primitive objects of the composition.

D. Implements an Interpret operation associated with terminal symbols in the grammar.

E. Both `A` and `B` are applicable.

F. Both `C` and `D` are applicable.

G. All of the options are applicable.

42. Which of the following statements is most true for the `Variable` class? (1)

A. Implements an Interpret operation for nonterminal symbols in the grammar. Interpret typically calls itself recursively on the variables representing R1 through Rn.

B. Contains children that are either composites or leaves.

C. Do not have children, define the primitive objects of the composition.

D. Implements an Interpret operation associated with terminal symbols in the grammar.

E. Both `A` and `B` are applicable.

F. Both `C` and `D` are applicable.

G. All of the options are applicable.

43. Select from the following options the intent of the pattern: (1)

A. Provide an interface for creating families of related or dependent objects without specifying the concrete classes.

B. Without violating encapsulation, capture and externalise an object's internal state so that the object can be restored to this state later.

C. Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

D. Define the skeleton of an algorithm in an operation, deferring some steps to sub- classes. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

E. Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

F. Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

G. Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

H. Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

44. Consider the following code: (1)

```
class Iterator
{

        public:
                virtual bool hasMore() = 0;
                virtual int* next() = 0;
                virtual int* current() = 0;
                virtual void reset() = 0;
                virtual ~Iterator(){}

                //Special
                virtual void back() = 0;
};

class Collection
{
        public:
                virtual Iterator* getForwardIterator() = 0;
                virtual Iterator* getBackwardIterator() = 0;
                virtual ~Collection(){}
};

class MatrixForwardIterator : public Iterator
{
        public:
                MatrixForwardIterator(int rows, int cols, int*** m);
                virtual ~MatrixForwardIterator();

                virtual bool hasMore();
                virtual int* next();
                virtual int* current();
                virtual void reset();
                virtual void back();

        private:
                int maxRow;//Max rows in matrix
                int maxCol;//Max columns in matrix
                int currentRow;
                int currentCol;
                int*** data;//Shared with a Matrix object
};

class MatrixBackwardIterator : public Iterator
{
        public:
                MatrixBackwardIterator(int rows, int cols, int*** m);
                virtual ~MatrixBackwardIterator();

                virtual bool hasMore();
                virtual int* next();
                virtual int* current();
                virtual void reset();
                virtual void back();

        private:
                int maxRow;//Max rows in matrix
                int maxCol;//Max columns in matrix
                int currentRow;
                int currentCol;
                int*** data;//Shared with a Matrix object
};

class Matrix : public Collection
{
        public:
                friend ostream& operator<<(ostream&,Matrix&);

                Matrix(int r, int c);
                virtual ~Matrix();

                virtual Iterator* getForwardIterator();
                virtual Iterator* getBackwardIterator();

        private:
                int numRows;
                int numCols;
                int*** items;
};

//Matrix implementation
Matrix::Matrix(int r, int c)
{
        numRows = r;
        numCols = c;
        items = new int**[numRows];
        for(int i = 0; i < numRows; ++i)
        {
                items[i] = new int*[numCols];
                for(int j = 0; j < numCols; ++j)
                {
                        items[i][j] = new int(0);
                }
        }
}

Matrix::~Matrix()
{
        for(int i = 0; i < numRows; ++i)
        {
                for(int j = 0; j < numCols; ++j)
                {
                        delete items[i][j];
                }
                delete [] items[i];
        }
        delete [] items;
        items = 0;
}

Iterator* Matrix::getForwardIterator()
{
        return new MatrixForwardIterator(numRows, numCols,items);
}

Iterator* Matrix::getBackwardIterator()
{
        return new MatrixBackwardIterator(numRows, numCols,items);
}

ostream& operator<<(ostream& os,Matrix& m)
{
        os<<"=============="<<endl;
        for(int i = 0; i < m.numRows; ++i)
        {
                for(int j = 0; j < m.numCols; ++j)
                        os<<*(m.items[i][j])<<' ';
                os<<endl;
        }
        os<<"=============="<<endl;

        return os;
}
```

Assume the following `main`:

```
int main()
{
        Matrix m(2,3);
        Iterator* iter = m.getBackwardIterator();

        while(iter->hasMore())
        {
                (*iter->next()) = 9;
                cout<<m;
        }

        cout<<"DONE"<<endl;

        delete iter;
        iter = 0;

        return 0;
}
```

From the options below, select the correct implementation for the `next` function of the `MatrixBackwardIterator` class to generate the following output:

```
===========
0 0 0
0 0 9
===========
===========
0 0 0
0 9 9
===========
===========
0 0 0
9 9 9
===========
===========
0 0 9
9 9 9
===========
===========
0 9 9
9 9 9
===========
===========
9 9 9
9 9 9
===========
DONE
```

A.
```
int* MatrixBackwardIterator::next()
{
        if(!hasMore())
                return 0;

        int* current = data[currentRow][currentCol];

        int rowReset = currentRow-1;
        currentCol++;
        currentRow++;

        if(currentCol == maxCol)
        {
                currentCol = 0;
        }
```

```
          if(currentRow == maxRow)
          {
                  currentRow = rowReset;
          }

          return current;
  }
```

B.
```
  int* MatrixBackwardIterator::next()
  {
          if(!hasMore())
                  return 0;

          int* current = data[currentRow][currentCol];
          currentCol--;

          if(currentCol == -1)
          {
                  currentRow--;
                  currentCol = maxCol;
          }

          return current;
  }
```

C.
```
  int* MatrixBackwardIterator::next()
  {
          if(!hasMore())
                  return 0;

          int* current = data[currentRow][currentCol];

          currentCol++;
          if(currentCol == maxCol)
          {
                  currentRow++;
                  currentCol = 0;
          }

          return current;
  }
```

D.
```
  int* MatrixBackwardIterator::next()
  {
          if(!hasMore())
                  return 0;

          int* current = data[currentRow][currentCol];
          currentCol--;

          if(currentCol == 0)
          {
                  currentRow--;
                  currentCol = maxCol;
          }

          return current;
  }
```

E.
```
  int* MatrixBackwardIterator::next()
  {
          if(!hasMore())
                  return 0;
```

```
                int* current = data[currentRow][currentCol];
                currentCol++;

                if(currentCol == -1)
                {
                        currentRow--;
                        currentCol = maxCol;
                }

                return current;
        }
```

45. Consider the following code: (1)

```
class A
{
        public:
                virtual ~A();
                virtual void doSomething(string t, int num) = 0;
};

class B : public A
{
        public:
                friend class D;
                virtual ~B();
                virtual void doSomething(string t, int num);

        protected:
                B(string c);
                string c;
                string w;
};

class C
{
        public:
                virtual ~C();
                virtual A* something(char m) = 0;
};

class D : public C
{
        public:
                virtual ~D();
                virtual B* something(char m);
};
```

Select from the following options the intent of the pattern:

A. Provide an interface for creating families of related or dependent objects without specifying the concrete classes.

B. Without violating encapsulation, capture and externalise an object's internal state so that the object can be restored to this state later.

C. Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

D. Define the skeleton of an algorithm in an operation, deferring some steps to sub- classes. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

E. Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

F. Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

G. Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

H. Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

46. Which of the following is considered the *original* Singleton? (1)

A. **GoF** - It uses a static pointer and an if-statement to check if it has been initialised

B. **Larman** - Like GoF, but uses eager initialisation instead of lazy initialisation

C. **Mulder** - Like GoF, but hides the copy constructor and assignment operator

D. **Meyers** - Uses a static local variable instead of a pointer.