Matthew Schoeman

U17029377

BIS Multimedia

COS 214 – Semester Test 2

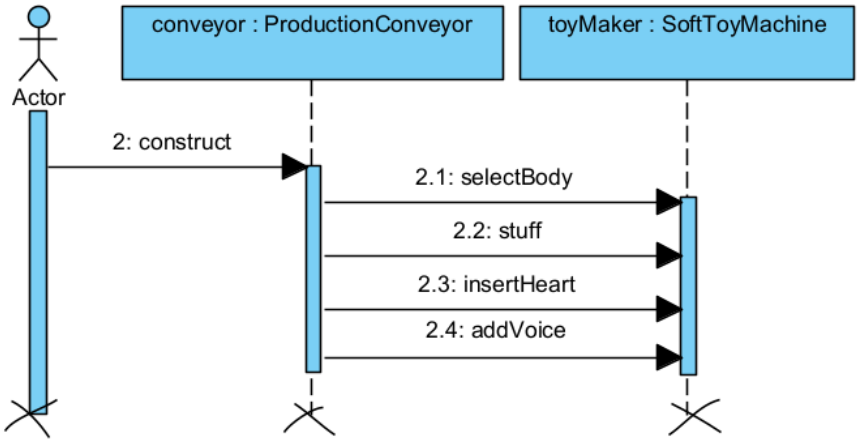
Question 1

i) Strategy
ii) State
iii) Decorator
iv) Composite
v) Observer
vi) Iterator

Question 2

a) ProductionConveyor is dependent on SoftToyMachine because it requires a SoftToyMachine in the construct method as a parameter.

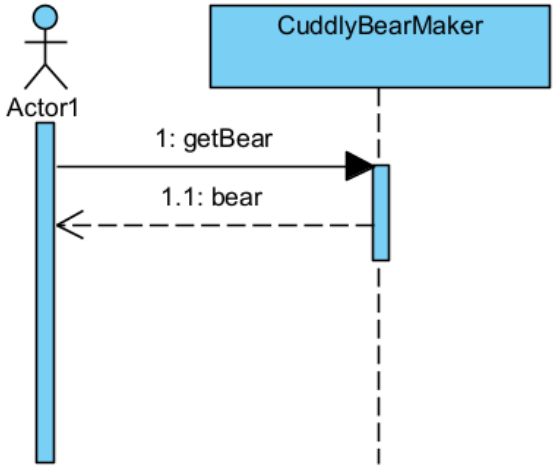
b)



c)

i) The SoftToy lifeline represents the toy object's existence, while the Actor exists so will the toy object. SoftToy* toy = new SoftToy();
ii) SoftToy* toy = new SoftToy();

d)



Question 3

a)

i) watch my_var

ii) The program will pause execution when my_var is changed. It will then show the old value of my_var and the new value of my_var.

b) This line means that there was a write that occurred out of bounds of size 20688 bytes.

c)

Question 4

(Composite, Component, Graphic)

(Composite, Leaf, Ellipse)

(Composite, Composite, CompositeGraphic)

(Decorator, Component, Graphic)

(Decorator, ConcreteComponent, Ellipse)

(Decorator, Decorator, GraphicDecorator)

(Decorator, ConcreteDecorator, {label, Box})

(Template, AbstractClass, Graphic)

(Template, ConcreteClass, CompositeGraphic)

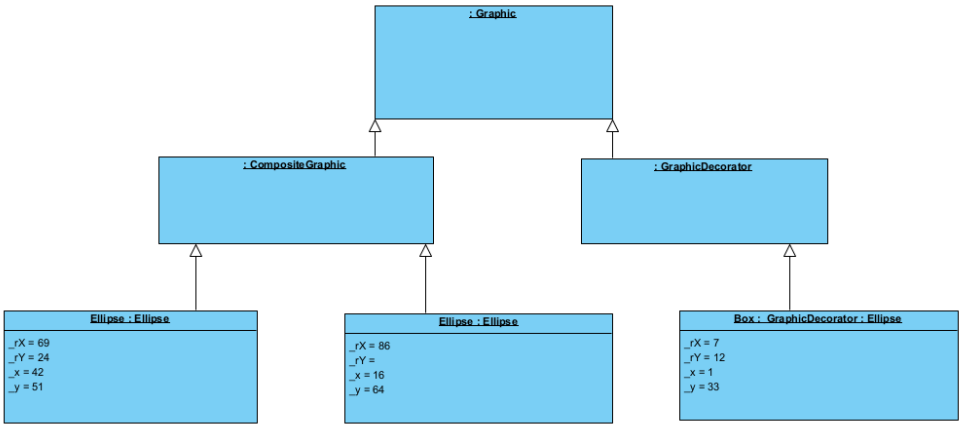
Question 5

a)

```
#ifndef BOX_H
#define BOX_H

#include "Graphic.h"

class Box : public Graphic{
    private:

    public:
        Box();
        Box(int, int, unsigned, int, unsigned int);
        void print();
};
#endif
```

b)

i)



ii)

```
int main() {
    Graphic* g = new CompositeGraphic();
    Graphic* g1 = new CompositeGraphic();
    Graphic* e1 = new Ellipse(42, 51, 69, 24);
    Graphic* e2 = new Ellipse(16, 64, 86, 33);
    g1->addGraphic(e1);
    g1->addGraphic(e2);
    g1 = new Label(g1,"Composite");

    Graphic* b = new Box(1, 33, 7, 12);
    g1 = new Label(b, "Decorator");
```

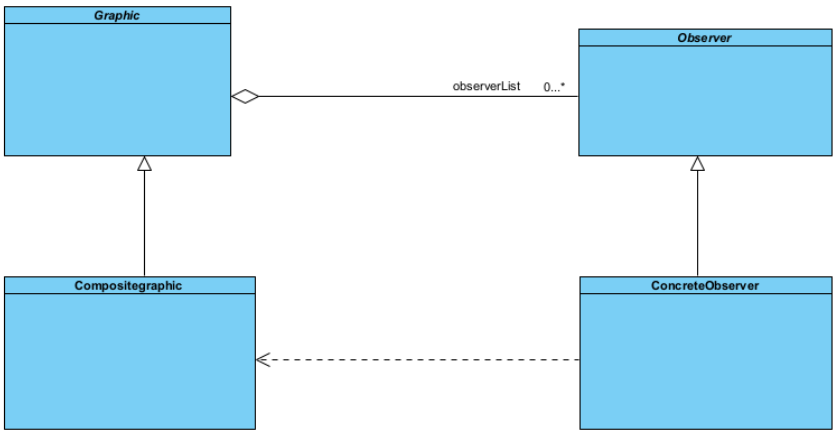
```
    g->addgraphic(g1);

    g->print();
    cout<<endl;
    delete g;
}
```

c)

```
Compositegraphic::~CompositeGraphic(){
    list<Graphic*>::iterator it = _l.begin();
    for( ; it != _l.end(); ++it){
        delete *it;
    }
    delete _l;
}
DecoratorGraphic::~DecoratorGraphic(){
    list<Graphic*>::iterator it = _component.begin();
    for( ; it != _component.end(); ++it){
        delete *it;
    }
    delete _component;
}
```

Question 6

a) Graphic class as abstract, therefore both the compositeGraphic and Decoratorgraphic can overwrite it to print out there different representations.
b) The subject hierarchy in the Observer Pattern.
c) The compositeGraphic class as it stores a list of subjects/graphic objects for which the observer can observer.
d) compositeGraphic
e)

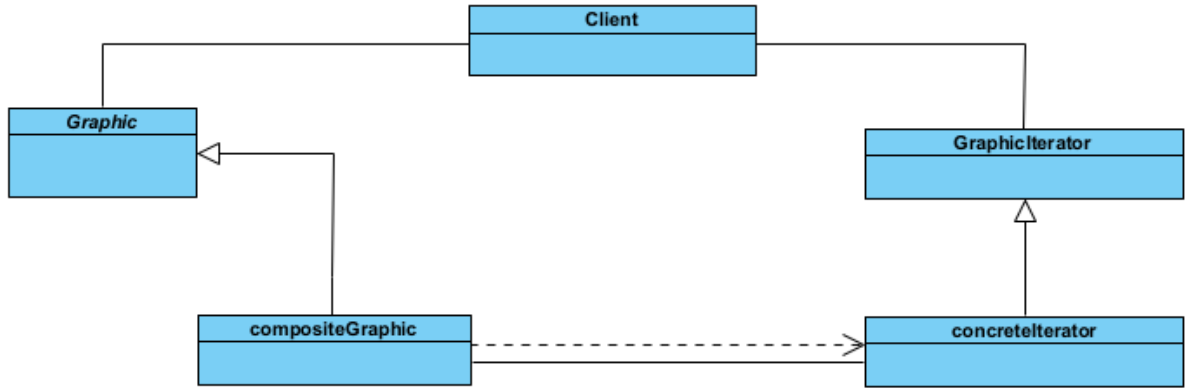


Question 7

a) The graphic class needs to declare a method called creatIterator which returns a GraphicIterator pointer : GraphicIterator* createIterator();

b)

c)



d)

```
Class Iterator {
    public:
      Iterator();
     ~Iterator();
      Graphic* next();
      Graphic*current();
      Graphic* first();
}
```

e) Graphic* start, would have the start method from the Iterator class. Stack<Graphic*> nextStack would utilize the next method from the Iterator class;

f)

```
Graphic* GraphicIterator::operator++(){
    if(this != nullptr){this->current = this->current->next;}
     return *this;
}
```

g)