A picture containing text

Description automatically generatedMatthew Schoeman

U17029377

BIS Multimedia

COS 214 – Semester Test 2

# Question 1

1. Strategy
2. State
3. Decorator
4. Composite
5. Observer
6. Iterator

# Question 2

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

Diagram

Description automatically generated

1. 1. The SoftToy lifeline represents the toy object’s existence, while the Actor exists so will the toy object. SoftToy\* toy = new SoftToy();
   2. SoftToy\* toy = new SoftToy();

Diagram

Description automatically generated

# Question 3

2. 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.

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

# 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



#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



Diagram

Description automatically generated



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;

}



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

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

Diagram

Description automatically generated

# Question 7

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

Diagram

Description automatically generated



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

1. Graphic\* start, would have the start method from the Iterator class. Stack<Graphic\*> nextStack would utilize the next method from the Iterator class;
2. Graphic\* GraphicIterator::operator++(){  
    if(this != nullptr){this->current = this->current->next;}  
    return \*this;  
   }