UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Engineering, Built Environment and IT
Department of Computer Science

Software Modelling
COS 214

Examination Opportunity 2 (EO2)

4 October 2021

Examiners

Internal: Dr Linda Marshall and Mr Rethabile Mabaso

Instructions

1. Read the question paper carefully and answer all the questions.
2. The assessment opportunity comprises of **7** questions on **6** pages.
3. **2** hours have been allocated for you to complete this paper. An additional 30 min have been allocated for you to download the paper and upload your answer document.
4. Write your answers on a separate document (eg. using an editor or handwritten) and submit the document in **PDF format**.
5. Please make sure your **name, student number and a photograph of your student card is clearly visible** in the document you upload. Provide an e-mail address and a phone number on your paper where you can be contacted, should there be any problems.
6. This paper is **take home** and is subject to the University of Pretoria Integrity statement provided below.
   - You are allowed to consult any literature.
   - You are not allowed to discuss the questions with anyone.
   - You may not copy from online resources. All answers must be in your own words.
7. If you have any queries when writing the paper, post them in good time to the COS 214 WhatsApp group or use the chat functionality on the Blackboard Collaborate session for EO1. Make sure your post in general enough as not to give away any answers.
8. An upload slot will be open on the module ClickUP page under the **Tests and EOs** menu option for the duration of the examination opportunity (17:30 to 19:30) and then for an additional 30 min to give enough time to download this paper, create the PDF containing your answers and then upload your PDF. **No late submissions will be accepted.**

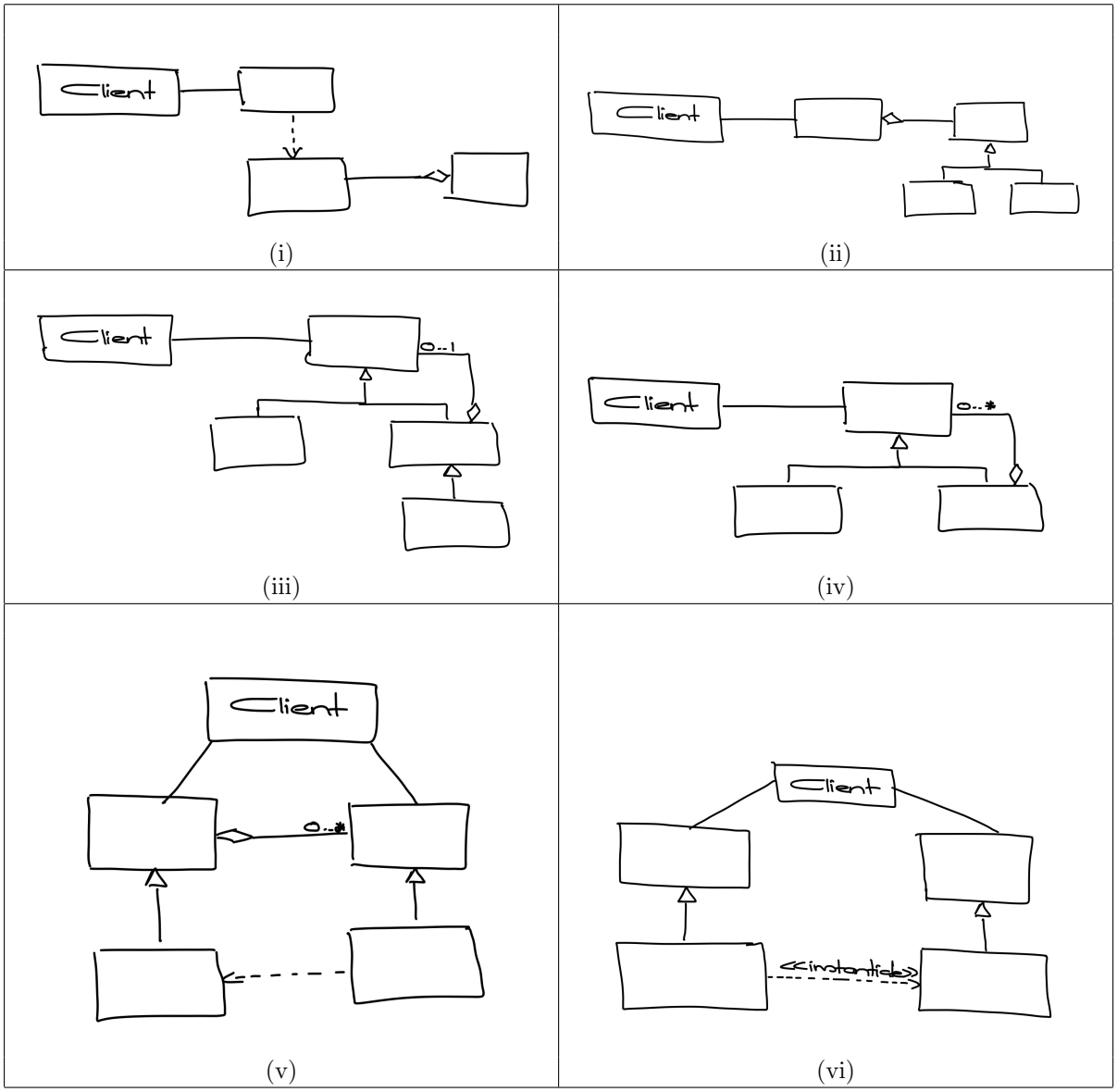
Integrity statement:
The University of Pretoria commits itself to produce academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

Question:	1	2	3	4	5	6	7	Total
Marks:	8	17	10	8	17	12	29	101

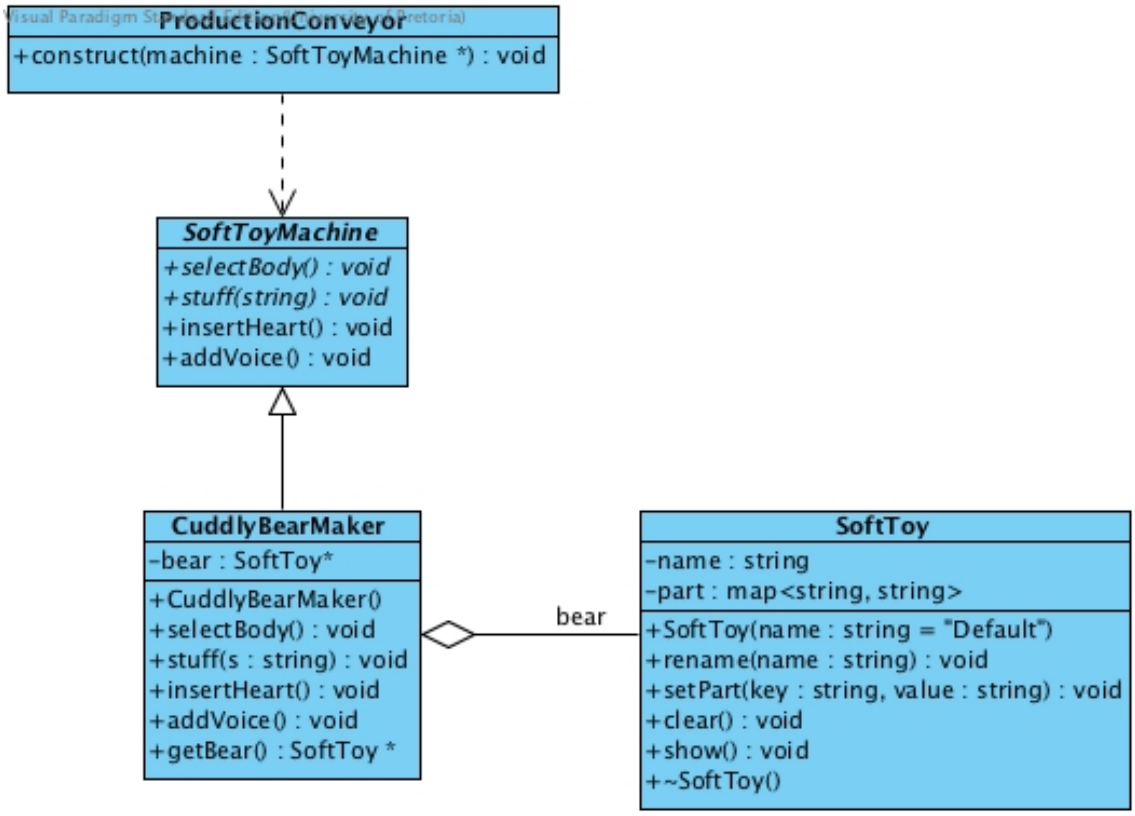
Full marks is: 95

Short questions

1. Identify the design pattern(s) from the skeletal UML class diagrams. Write down the number, for example (i), followed by the name of the pattern it represents. (8)



2. Consider the following class diagram and answer the questions that follow.



Assume that the objects `conveyor` and `toyMaker` exist and are defined as follows:

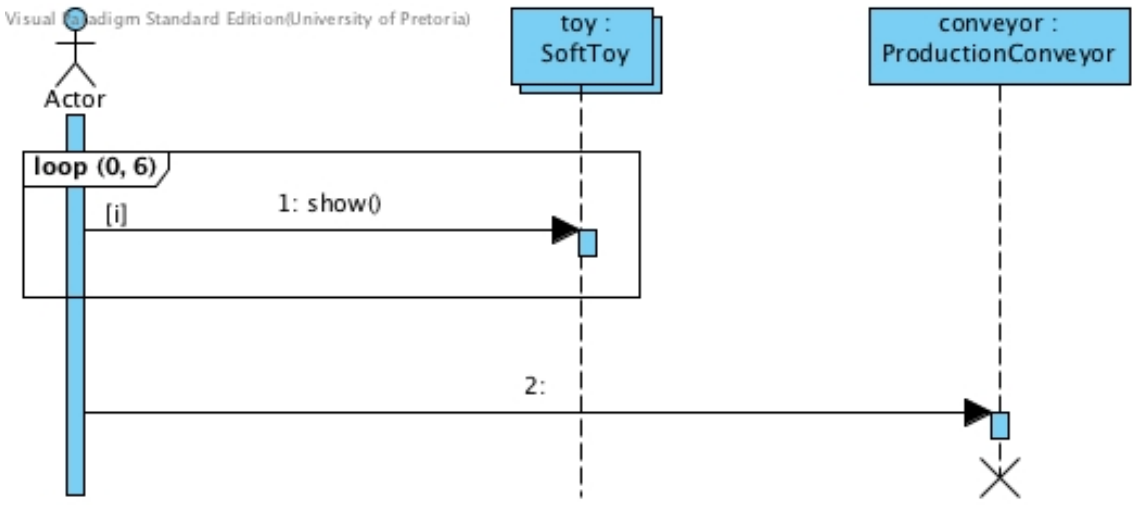
SoftToyMachine* toyMaker;
ProductionConveyor* conveyor;

(a) Why is the relationship between `ProductionConveyer` and `SoftToyMachine` a dependency relationship? (1)

(b) The implementation of the `construct` function defined in `ProductionConveyor` is given below. Draw the sequence diagram to show the interaction over time between the objects involved in the interaction. Make use of the `conveyor` and `toyMaker` objects that were given in your solution. (7)

```
void construct(SoftToyMachine* machine) {
  machine->selectBody();
  machine->stuff("soft stuff");
  machine->insertHeart();
  machine->addVoice();
};
```

(c) Consider the following sequence diagram.



i. What does the `toy:SoftToy` lifeline represent and how would you define the object `toy`? (2)

ii. Provide the code represented from the perspective of the main program. (3)

(d) Assuming the `getBear` function is defined as follows, draw a sequence diagram showing the interaction between the main program and the `CuddlyBearMaker` class when the `getBear` function is called. (4)

```
SoftToy* CuddlyBearMaker::getBear(){
  return bear;
}
```

3. (a) Consider the simple C++ program shown below:

```
int main(){
        int my_var = 10;

        return 0;
}
```

i. Write a GDB command to pause the program whenever `my_var`'s value is modified. (2)

ii. Suppose `my_var` is modified, what will happen to the program and what will be printed on the screen if the command in i. is used. (2)

(b) A single line of output from Memcheck is shown below: (3)

==20688== Invalid write of size 4

Explain what this line of output means.

(c) The Google Test statement below is a simple test of the `multiply()` function: (3)

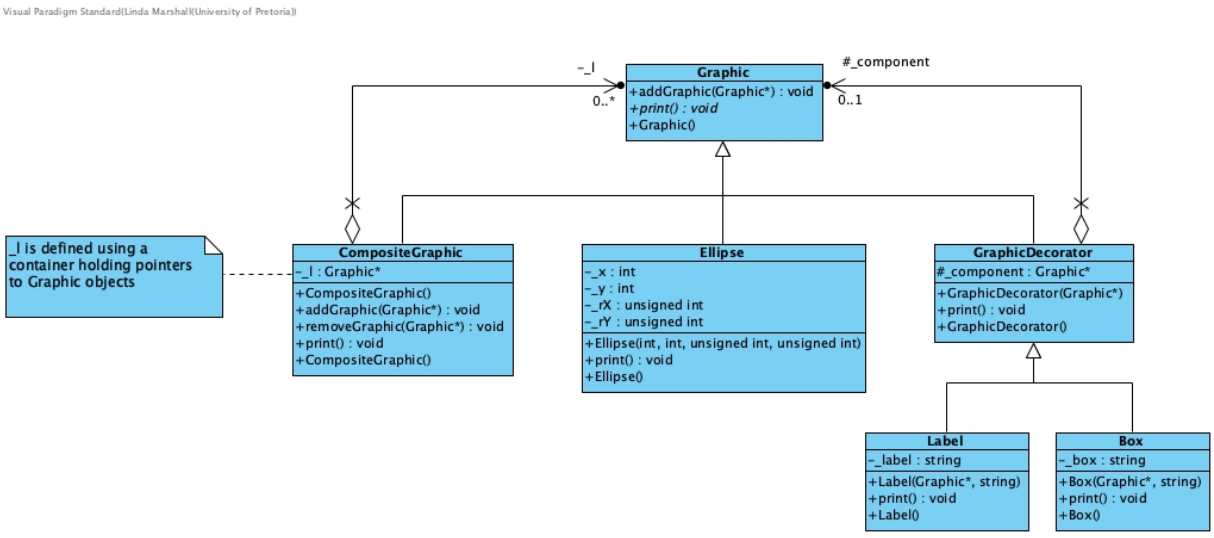
EXPECT_EQ(10, multiply(5,2)) << "Five multiplied by two must equal ten";

Rewrite the test statement to stop execution when the test fails with floating point arguments.

Long questions

Scenario

A friend of yours found COS 214 design pattern slides on the internet and came to you to ask for help in understanding the patterns. They had downloaded the slides for the Composite lecture and showed you the UML class diagram for the `Graphic` hierarchy. You searched through your files and found the following diagram that combines the Composite design pattern your friend had shown you with the Template Method and Decorator design patterns.



4. Identify the participants for each of the patterns using the diagram. Provide the pattern name, participant name and the corresponding class name as a triple. For example: (Composite, Component, Graphic). If more than one corresponding class falls under a particular participant in the pattern, list the classes as a set. That is, list the class names in curly braces, '{' and '}'. (8)

5. (a) Provide the code for the header file `Box.h` that defines the class `Box`. Assume that each class is defined in its own header file (.h) and has its own corresponding implementation (.cpp) file. (5)

(b) The implementations of the `print` operation for each of the classes is given by:

```
void Ellipse :: print () {
  std :: cout << "<Ellipse("
              << _x << ", "
              << _y << ", "
              << _rX << ", "
              << _rY << ")>";
}

void CompositeGraphic::print() {
   std :: cout<<"[";
   for (std :: list <Graphic*>::iterator it = _l.begin(); it != _l.end(); ++it)
     (*it)->print();
   std :: cout<<"]";
}

void GraphicDecorator::print() {
  _component->print();
}

void Box::print() {
  std :: cout<<"{"<<_box<<" ";
  GraphicDecorator::print();
  std :: cout<<"}";
}

void Label::print () {
  std :: cout<<"{"<<_label<<" ";
  GraphicDecorator::print();
  std :: cout<<"}";
}
```

The client program that instantiates the objects of the Composite and Decorator patterns, prints the following output:

```
[{Composite [<Ellipse(42, 51, 69, 24)><Ellipse(16, 64, 86, 33)>]}{Decorated {Box <Ellipse(1,
33, 7, 12)>}}]
```

i. Draw the "object tree" that will result in this output. (4)

ii. Complete the client (in this case a main program), that will result in the given output. (4)

```
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");

  / / Add your code here....

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

}
```

(c) Provide the code for the `GraphicDecorator` and `CompositeGraphic` destructors in order to delete the tree. Note: you need to consult the code given in Part b of this question to see how `_l` is defined. (4)

6. Assume that other than the `print` functionality provided in the `Graphics` hierarchy, there is a GUI which allows for two views of the resulting graphics object hierarchy. The one view is an object-relationship view and the other a CAD-like view. Both these views allow the user to manipulate the view, which results in the representation of the underlying object to change. For example, in the CAD-like view, graphic elements can be `move`ed You may assume `move` takes 4 parameters, `from_x`, `from_y`, `to_x` and `to_x`. Additionally you may assume that a `Graphic` has an `x` and a `y` attribute added, representing the co-ordinate which indicates the top left-most corner of an invisible box around the graphic element.

   (a) In which classes does the `move` operation need to be defined. Explain your answer. (2)

   (b) Which participant(s) of the Observer Design pattern does the `Graphic` hierarchy represent? (1)

   (c) Which class will the Observer need to register with? (2)

   (d) Which class will define the list of Observers and manage the registration and deregistration of the Observers? (1)

   (e) Draw the UML Class diagram, showing how the Observer Design pattern fits together with the classes in the UML class diagram. Do not include all the details of the classes. Only the class details required by the Observer pattern need to be included. (6)

7. Assume that the following function has already been added to `Graphic` and has been implemented in its subclasses.

```
// Element type
virtual std::string elementType() = 0;
```

   (a) What else needs to be added to `Graphic` for the implementation of the Iterator Design pattern? You may assume a class with the name `GraphicIterator`, representing the *ConcreteIterator* participant, has been defined. (2)

   (b) Provide an implementation for one of the *createIterator()* equivalent functions. Again, you may assume that `GraphicIterator` has been defined. (4)

   (c) Design the Iterator and ConcreteIterator participants using the functions that are defined by the structure of the Iterator Design pattern as given in the slides. Provide only the class names and the relationships between the classes that are relevant to the Iterator design pattern. (4)

   (d) Define the `Iterator` participant class. (4)

   (e) The `GraphicIterator` class includes two private attributes, `start` and `nextStack` – defined as follows, what are their respective functions? (4)

```
Graphic* start;
std::stack<Graphic*> nextStack;
```

   (f) Write the "`next`" function of the `GraphicIterator` class. (6)

   (g) Complete the following main program to make use of the GraphicIterator. For each element in `g2`, print the string that is returned by the call to `elementType`. (5)

```
int main(){
        Ellipse* e1 = new Ellipse(42, 51, 69);
        Ellipse* e2 = new Ellipse(16, 64, 86);
        Ellipse* e3 = new Ellipse(1, 33, 7);
        CompositeGraphic* g1 = new CompositeGraphic();
        CompositeGraphic g2;
        g1->addGraphic(e1);
        g1->addGraphic(e2);

        g2.addGraphic(e3);
        g2.addGraphic(g1);

        // Your code goes here .....

        return 0;
}
```