UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Engineering, Built Environment and IT
Department of Computer Science

Software Modelling
COS 214

Examination Opportunity 3 (EO3)

22 October 2020

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 **10** 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 in PDF format.
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 and student number 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 on the ClickUP Discussion Board under the appropriate thread in the **Tests and EOs** Forum. Note, this forum is moderated and therefore your post will not be available for viewing to the COS214 class until is has been moderated.
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 and upload your PDF. **No late submissions will be accepted.**

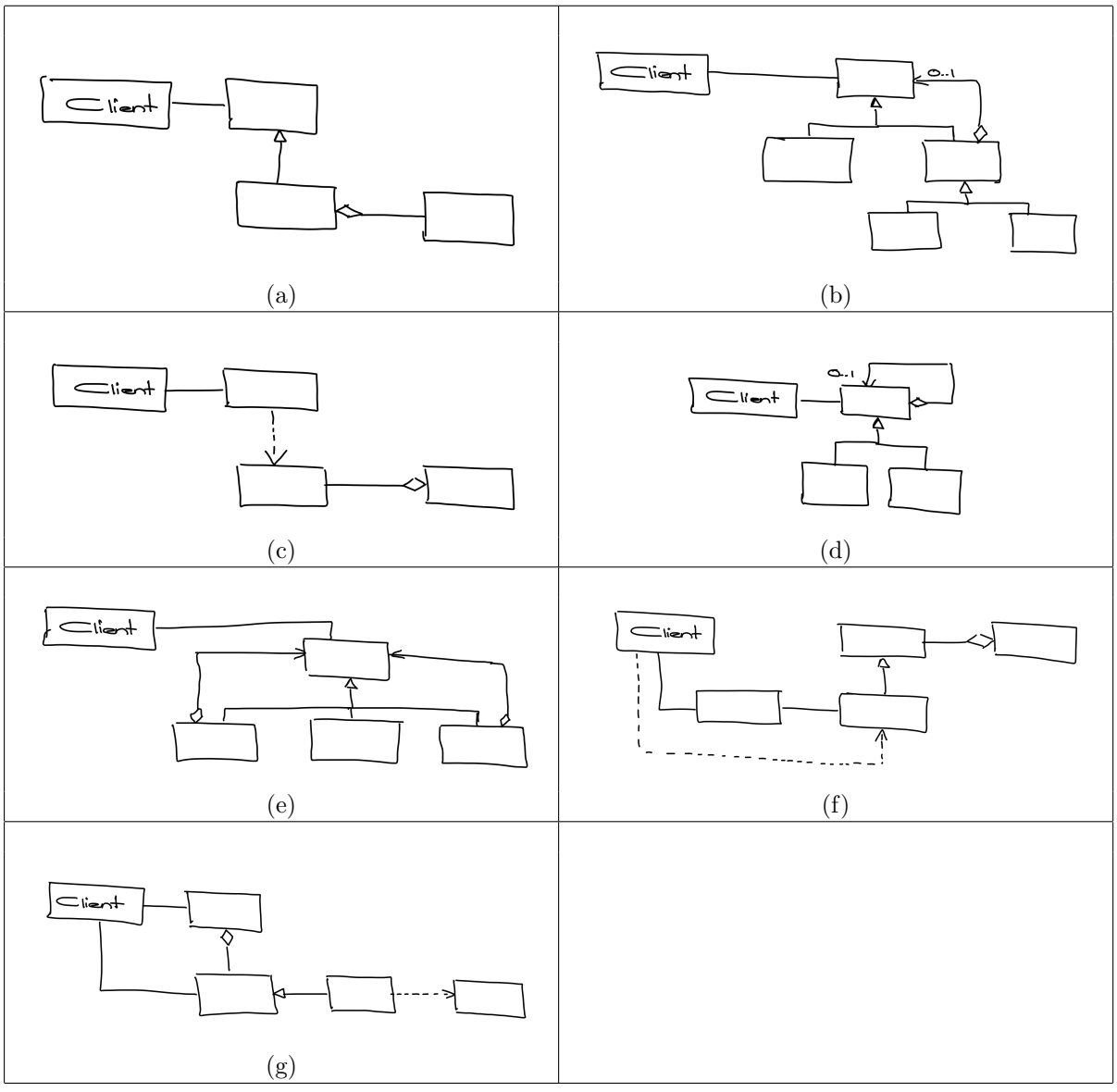
Integrity statement:
The University of Pretoria commits itself to produce academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest 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:	14	7	5	13	12	15	19	85

Full marks is: 80

Short questions

1. Identify the design pattern shown in each of the following skeletal UML class diagrams. Also state whether the pattern is classified as creational, behavioural or structural. (14)



Total for Question 1: 14

2. The following questions apply to the Command design pattern.
   (a) What are macro commands? (1)
   (b) Why is the Decorator not always appropriate to create macro commands? (2)
   (c) Which other design pattern can be used to create macro commands? (1)
   (d) Assume you have a collection of commands and during compile time you are unaware which command will execute which function. Which other design pattern can be used to group all the commands together and pass the request from one command to the other until one of the commands can execute the specified functionality? (1)

(e) Which other design pattern can be used to efficiently traverse through the grouped commands and move from one command to the next? (1)

(f) Which other design pattern can be used to undo the operation of a command that was mistakenly executed? (1)

Total for Question 2: 7

3. A local area network (LAN) broadcasts packets to the entire network, irrespective of who the receiving node is. Hence, if a packet has to be transmitted from `192.168.1.11` to `192.168.1.22`, the packet is sent to the router at `192.168.1.1` and then the router broadcasts the packet onto the network. All nodes on the network may receive the packet, but only node `192.168.1.22` will open and inspect the contents of the packet.

(a) Why is the Observer design pattern not appropriate to broadcast packets on the network? (1)

(b) Which design pattern should rather be used on the router to receive packets from any node on the network and then forward it by broadcasting it to all other nodes on the network? (1)

(c) You, the network administrator, suspect that certain nodes on the network are intruders controlled by hackers. Which design pattern should be used to observe certain nodes on the network and notify the router of any suspicious activity, so that the router can automatically disconnect these malicious nodes? (1)

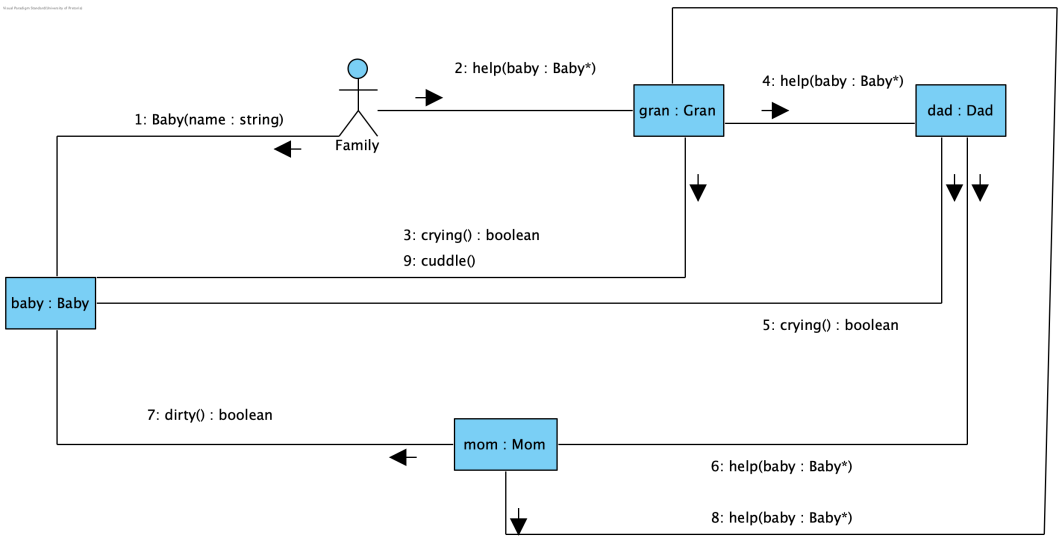
(d) You want to connect two LANs running on different technologies, one is a Windows network, the other one an Apple network. Which design pattern is most appropriate to be implemented on the router between the two LANs, in order to convert the calls from the Windows LAN into calls that are understood by the Apple LAN? (1)

(e) Routers typically have very slow processors and code has to be written efficiently to avoid overloading. Each time a new node is connected, the router creates an object in memory containing a lot of information that is exactly the same for all newly connected nodes. Only after initialisation, the nodes change certain information in this object, such as the IP address. Which design pattern can be used to reduce computational time by avoiding the creation of the large object from scratch each time a new node is connected? (1)

Total for Question 3: 5

Long questions

4. Consider the following UML Communication diagram and answer the questions that follow.



Assume that all objects, other than `baby`, are instantiations of classes which inherit from the class `Adult`. Class `Adult` implements an association which results in the `Adult` hierarchy implementing *recursive composition*.

(a) Which design pattern gave rise to the given UML Communication diagram? (1)

(b) Draw the UML Class diagram which includes the design pattern illustrated in the UML Communication diagram. Include all the classes, relationships and features given in the Communication diagram or needed to highlight how the pattern works. (4)

(c) Annotate the UML class diagram in the previous question with the participants of the pattern you identified. (3)

(d) Draw the corresponding UML Sequence diagram for the given UML Communication diagram. (5)

Total for Question 4: 13

5. Your friend has forwarded you the following header file and implementation file in which the class `InsaneWrapper` is defined and implemented.

- Header file - InsaneWrapper.h

```
#include <iostream>

#ifndef INSANEWRAPPER_H
#define INSANEWRAPPER_H

using namespace std;
class InsaneWrapper {
public:
    InsaneWrapper(int const& _value);
    InsaneWrapper& operator = (int _value) ;
    InsaneWrapper& operator = (InsaneWrapper &_wrapper);
    operator int const () const;
    InsaneWrapper& operator ++ ();
    InsaneWrapper& operator ++ (int);
    InsaneWrapper& operator -- ();
    InsaneWrapper& operator -- (int);
    bool operator == (InsaneWrapper const& _wrapper);
protected:
    int value;
};
#endif
```

- Implementation file - InsaneWrapper.cpp

```
#include <iostream>
#include "InsabeWrapper.h"

using namespace std;

InsaneWrapper::InsaneWrapper(int const& _value) {
    value = _value;
}

InsaneWrapper::InsaneWrapper& operator = (int _value) {
        value = _value; return *this;
}

InsaneWrapper& InsaneWrapper::operator = (InsaneWrapper &_wrapper) {
        _wrapper.value = value; return *this;
}

InsaneWrapper::operator int const () const {
        return value + 1;
}

InsaneWrapper& InsaneWrapper::operator ++ () {
        value++; return *this;
}

InsaneWrapper& InsaneWrapper::operator ++ (int) {
```

```
        value = ++value +1; return *this;
    }

    InsaneWrapper& InsaneWrapper::operator -- () {
        value--; return *this;
    }

    InsaneWrapper& InsaneWrapper::operator -- (int) {
        value = value-- +1; return *this;
    }

    bool InsaneWrapper::operator == (InsaneWrapper const& _wrapper)     {
        return value = _wrapper.value;
    }
```

The functionality required by your friend, and provided to some extent by the `InsaneWrapper` class, is defined in the following interface class `Wrapper`.

```
class Wrapper {

  public:
    virtual void print(ostream&) = 0; // print the object
    virtual void increment() = 0; // increment the wrapped object
    virtual void decrement() = 0; // decrement the wrapped object
    virtual void update(Wrapper*) = 0; // update the wrapped object
    virtual ~Wrapper() {};
};
```

Having sat through a lecture in which the **Adapter** design pattern was explained, you realise that this is exactly what is needed to use the `InsaneWrapper` to provide the required functionality, and that this can be done by implementing an **Object Adapter** design pattern. You design the `SaneWrapper` class given below.

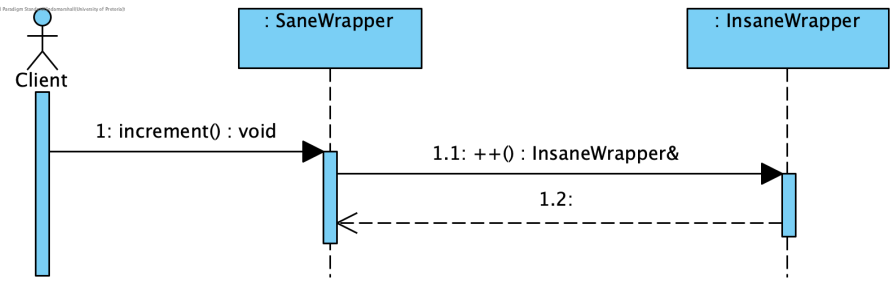
```
class SaneWrapper : public Wrapper {
public:
      SaneWrapper();
      SaneWrapper(int);
      virtual void print(ostream&);
      virtual void increment();
      virtual void decrement();
      virtual void update(Wrapper*);
      virtual ~SaneWrapper();
protected:
      InsaneWrapper* object;
};
```

(a) Provide implementations for each of the following `SaneWrapper` functions.

i. The constructor that takes an `int` as a parameter and initialises `object` accordingly. (2)

ii. The `print` function. Note that the conversion operator of the `InsaneWrapper` adds one to the value being returned. Your `print` function needs to nullify this addition. (2)

iii. The `increment` function. Make use of the given UML Sequence diagram to write the code for the function. (2)



(b) Describe what needs to be updated to convert the given implementation to a **Class Adapter** implementation of the Adapter pattern. (2)

(c) Provide the implementation for **one** of the `SaneWrapper` constructors for the Class Adapter implementation of the pattern. (2)

(d) Choose **one** of the following `SaneWrapper` operations: `print`, `increment` or `decrement`. Provide the Class Adapter implementation thereof. (2)

Total for Question 5: 12

6. The following code was written to illustrate the Builder Design pattern. Carefully read the code and then answer the questions that follow.
Note, the implementation of the functions is provided as part of the class definitions to limit paging during the examination opportunity.

```
#include <iostream>
#include <cstring>
using namespace std;

class HousePlan {
public:
        HousePlan(){};
        virtual ~HousePlan(){};
        virtual void setBasement(string basement) = 0;
        virtual void setStructure(string structure) = 0;
        virtual void setRoof(string roof) = 0;
        virtual void setInterior(string interior){};
};


class House : public HousePlan {
public:
        House(): HousePlan(),
                basement("None"),
                structure("None"),
                roof("None"),
                interior("None")
        {}

        void setBasement(string b) {
                basement = b;
        }

        void setStructure(string s) {
                structure = s;
        }

        void setRoof(string r) {
                roof = r;
        }

        void setInterior(string t) {
                interior = t;
        }

        void show() {
                cout << "Basement: " << basement << endl;
                cout << "Structure: " << structure << endl;
                cout << "Roof: " << roof << endl;
                cout << "Interior: " << interior << endl;
                cout<<"--------------------------------"<<endl;
        }

private:
        string basement;
        string structure;
```

```
        string roof;
        string interior;
};


class HouseBuilder {
public:
        virtual void buildBasement() = 0;
        virtual void buildStructure() = 0;
        virtual void buildRoof() = 0;
        virtual void buildInterior() {};
        virtual House getHouse() = 0;
};


class CivilEngineer {
public:
        CivilEngineer(HouseBuilder* p) : pantologist(p) {};
        virtual ~CivilEngineer(){};
        void construct() {
                pantologist->buildBasement();
                pantologist->buildStructure();
                pantologist->buildRoof();
                pantologist->buildInterior();
        }
private:
        HouseBuilder* pantologist;
};



class TipiBuilder : public HouseBuilder {
public:
        TipiBuilder() {}

        void buildBasement() {
                tent.setBasement("Wooden Poles");
        }

        void buildStructure() {
                tent.setStructure("Wood and Ice");
        }

        void buildRoof() {
                tent.setRoof("Wood, caribou and seal skins");
        }

        House getHouse(){
                return tent;
        }

private:
        House tent;
};

class IglooBuilder : public HouseBuilder {
public:
        IglooBuilder() {}
        virtual ~IglooBuilder(){
                if (snowHouse) delete snowHouse;
        }
```

```
        void  buildBasement() {
                if (snowHouse) delete snowHouse;
                while (!isSafe())
                {
                        cout << "Finding a safer spot" << endl;
                }
                snowHouse = new House();
                snowHouse->setBasement("Ice Bars");
        }

        void buildRoof() {
                snowHouse->setRoof("Ice Dome");
        }

        void buildInterior()  {
                snowHouse->setInterior("Ice Carvings");
        }

        void buildStructure() {
                snowHouse->setStructure("Ice Blocks");
        }

        House getHouse(){
                House* replica = new House(*snowHouse);
                return *replica;
        }
protected:
        bool isSafe()
        {
                static int counter = 1;
                return ((counter++ % 5) == 0) ? true : false;
        }

private:
        House* snowHouse;
};
```

(a) Draw a UML Class diagram of these classes and how they relate to one another. For all classes except the `IglooBuilder`, you may omit the detail of the members of the class. Include all the detail of the `IglooBuilder` class. (8)

(b) Assume the following variables are declared in the client:

```
House dwelling;
HouseBuilder* nomad = new TipiBuilder();
HouseBuilder* inuk = new IglooBuilder();
```

i. Give the output of the following statement. Assume that an instance of `TipiBuilder` was instantiated as specified in the above definition but that no other code was executed yet. (2)

```
(nomad->getHouse()).show();
```

ii. Explain why the following statement will result in a segmentation fault. Assume that an instance of `IglooBuilder` was instantiated as specified in the above definition but that no other code was executed yet. (2)

```
(inuk->getHouse()).show();
```

iii. Write code that applies the pattern correctly to assign an instantiated house that was created by an `IglooBuilder` to `dwelling`. Use the objects that were defined in the above definition, and define and instantiate any additional objects that are required for this operation. (3)

Total for Question 6: 15

7. Each time you enter the COS214 Collaborate session you see an image of the Tardis pulsating at the bottom left of your screen. Not wanting to appear the fool, you decide not to tell anyone. One day your curiosity

gets the better of you. You load a C++ compiler on your laptop, charge it, click on the image of the Tardis and quickly pack it in your backpack. After a journey that feels like an eternity you pop out in small village hearing a big fat man calling "Dogmatix, come here my little doggy". "Oh no!", you think to yourself just before being tapped on the shoulder by a man looking the splitting image of Getafix, the druid from the "Asterix and Obelix" comic books you had been reading during the online Collaborate sessions. He politely questions you, "You are Dr Who???".

Getafix has a problem. The Gauls you see are living in decimal world, numbers as you know them. They have however been intercepting messages sent between the Romans who have surrounded their village. The messages seem to be referring to positions. The problem is, all these positions are given in Roman numerals. Getafix asks you if you could convert decimal numbers to the Roman number system. Luckily, you had a similar problem when reading the comics prior to your journey and had Googled and saved the BNF for some Roman numbers beginning at 1. You show these to Getafix on your laptop which has magically taken on the appearance of Sonic Sunglasses. You glance at Getafix, "Don't look at my browser history", and smile.

```
RomanNumber ::= Hundreds Tens Units
Hundreds ::= LowHundreds | CD | D LowHundreds
LowHundreds ::= Empty | LowHundreds C
Tens ::= LowTens | XL | L LowTens | XC
LowTens ::= Empty | LowTens X
Units ::= LowUnits | IV | V LowUnits | IX
LowUnits ::= Empty | LowUnits I
```

(a) You had also worked on a basic algorithm which converts a digit from a decimal number to a roman numeral. This algorithm makes use of a multiplier for hundreds, tens and units and a lookup table. The algorithm is encapsulated in the class called `RomanNumber`. The class definition and implementation of the algorithm are given below.

```
class RomanNumber {
public:
    RomanNumber(int);
    virtual string interpret(Conversion,Context) = 0;
protected:
    string interpreter(Conversion);
    int multiplier;
    int digit;
};

string RomanNumber::interpreter(Conversion conversion) {
    string str = "";
    switch (digit) {
        case 3: str = str + conversion.get(1*multiplier);
        case 2: str = str + conversion.get(1*multiplier);
        case 1: str = str + conversion.get(1*multiplier); break;
        case 4: str = str + conversion.get(1*multiplier) +
            conversion.get(5*multiplier); break;
        case 8: str = str + conversion.get(1*multiplier);
        case 7: str = str + conversion.get(1*multiplier);
        case 6: str = str + conversion.get(1*multiplier);
        case 5: str = conversion.get(5*multiplier) + str; break;
        case 9: str = conversion.get(1*multiplier) +
            conversion.get(10*multiplier); break;
    }
    return str;
}
```

i. The class `Conversion` defines a lookup table which stores roman numerals as capital letters. The lookup table for roman numerals is populated using the following code: (4)

```
conversion.set(1,'i');
conversion.set(5,'v');
conversion.set(10,'x');
conversion.set(50,'l');
```

```
conversion.set(100,'c');
conversion.set(500,'d');
```

Define the `Conversion` class and provide implementations for the `get` and `set` functions.

ii. What does the algorithm return for `conversion` if populated as above and for each of the values for *digit* and *multiplier* given below? For example, the result for `digit = 2` and `multiplier = 1` is `II`.

α) `digit = 5` and `multiplier = 1` (1)

β) `digit = 9` and `multiplier = 1` (1)

γ) `digit = 3` and `multiplier = 10` (1)

δ) `digit = 7` and `multiplier = 100` (1)

(b) From the BNF it is clear that the symbol `RomanNumber` is a nonterminal and comprises of `Hundreds Tens Units`. The decimal value of `111` is equivalent to `100 + 10 + 1` or `C + X + I = CXI`. The algorithm defined by `RomanNumber::interpreter(Conversion)` takes care of the details for the `Hundreds`, `Tens` and `Units` rules for expressions on the right-hand-side of the rule.

Assuming that `RomanNumber` is the *AbstractExpression* participant of the *Interpreter* design pattern. Answer the questions which follow.

i. Class `Context` provides a lookup table for the digits of the decimal values. This means that a number from 1 to 9 is stored in a lookup table for hundreds (`'H'`), tens (`'T'`) and units (`'U'`) respectively. The class definition for `Context` is given. (2)

```
class Context {
public:
    Context();
    void set(char key, int value);
     int get(char key);
private:
    map<char,int> valueMap;
};
```

Provide the statements to create an object of `Context` and to assign the digits representing the decimal number `217` to the lookup table.

ii. Assume the `interpret` function of `Tens` is implemented as follows, provide the definition for the `Tens` class. (4)

```
string Tens::interpret(Conversion conversion, Context context) {
    digit = context.get('T');
    return interpreter(conversion) +
        units->interpret(conversion,context);
}
```

iii. Identify the *TerminalExpression* and *NonTerminalExpression* participants of the pattern. (2)

iv. α) The largest number that can be converted using the Interpreter design pattern as defined above is 899. What single addition needs to be made so that all numbers from 1 to 999 can be converted? Note: The roman numeral `M` represents 1000. (1)

β) Assume that you wish to convert decimal values from 1000 to around 5000. What changes or additions would you need to make to the existing code? (2)

Total for Question 7: 19

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Engineering, Built Environment and IT
Department of Computer Science

Software Modelling
COS 214

Examination Opportunity 3 (EO3)

22 October 2020

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.

Personal Details:

Student Number : u18050362
Surname : Graaff
Name(s) : Werner
Email : u18050362@tuks.co.za
Cell : 0716407441

Current Address Where Exam is Taken:

Street : Balboa Place 11
Suburb : Eldoraigne
City : Pretoria
Province : Gauteng
Country : South Africa

Please record the times below:

Time you downloaded the paper : 17:30
Time you started writing : 17:40
Time you completed the paper :
Time you will be submitting the paper :

Please highlight blocks where you had load shedding during the EO time (if any):

17:00-17:30	17:30-18:00	18:00-18:30	18:30-19:00	19:00-19:30	19:30-20:00	20:00-20:30

For Examiners' Use Only:

Question	Marks	Score
1	14	
2	7	
3	5	
4	13	
5	12	
6	15	
7	19	
Total	**85 (80 full marks)**	

Question 1

(a) Object Adapter – Object Structural

(b) Decorator – Object structural

(c) Memento - Behavioural

(d) Chain of responsibility - Behavioural

(e) Interpreter - Behavioural

(f) Command - Behavioural

(g) Builder - Creational

Question 2

(a) Commands that are built from a set of other commands in a given order.

(b) As it already consists of a lot of components which would make it hard to keep track of the commands through all the different subclasses.

(c) Composite design pattern.

(d) The chain of responsibility design pattern

(e) The iterator design pattern

(f) The memento design pattern.

Question 3

(a) Because all the nodes would continuously have to attach and detach to the subject (which is the packet) which can become cumbersome.

(b) The mediator design pattern.

(c) The observer design pattern

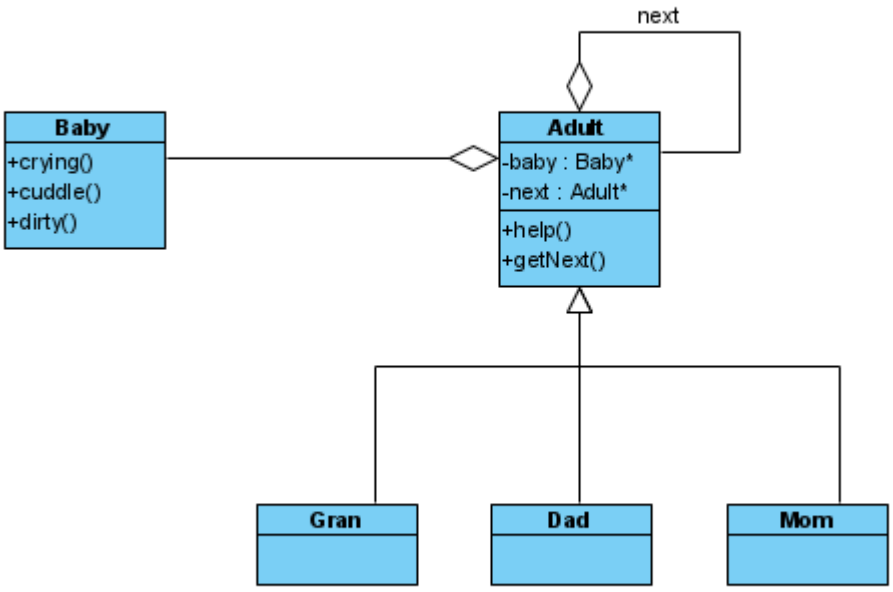
(d) The interpreter design pattern. //wrong

(e) The prototype design pattern.

Question 4

(a) The chain of responsibility design pattern.

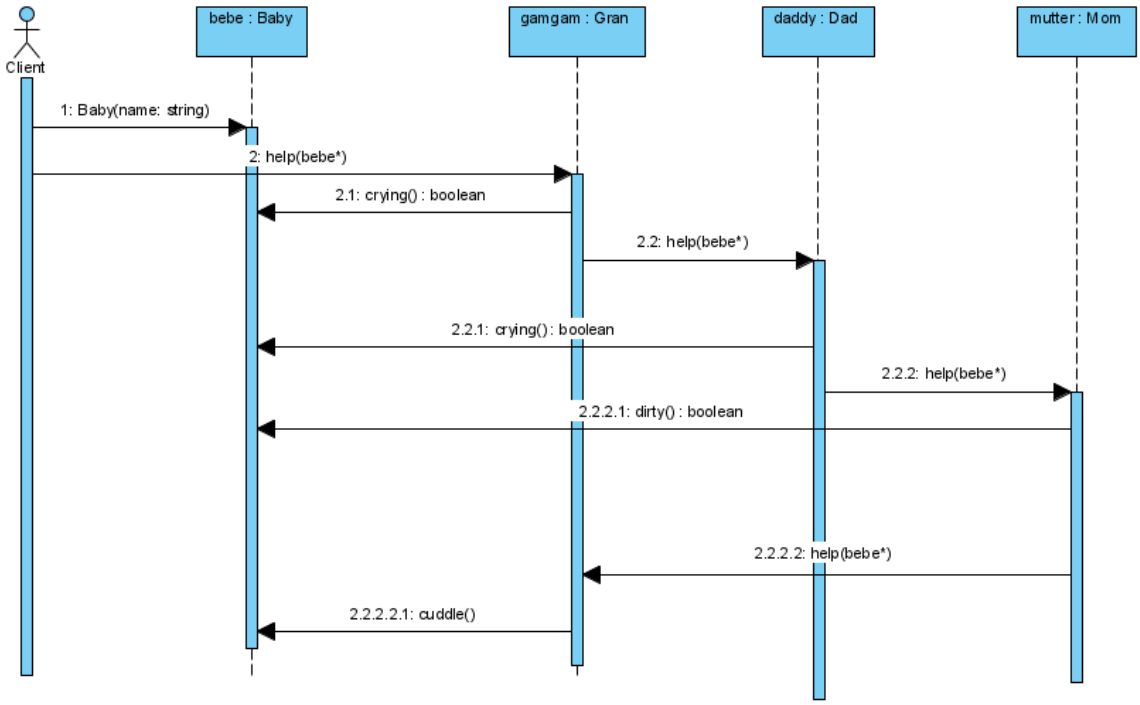
(b)



(c) Handler: Adult

Concrete Handlers: Gran, Dad, Mom

(d)



Question 5

(a) i.

```
SaneWrapper::SaneWrapper(int i)
   {
       this->object = new InsaneWrapper(i);
   }
```

ii.

```
virtual void SaneWrapper::print(ostream&)
   {
       cout<<"Printing InsaneWrapper's value"<<++(&this) - 1<<endl;
   }
```

iii.

```
virtual void SaneWrapper::increment()
   {
       ++(&this);
   }
```

(b) The SaneWrapper class would need to inherit from InsaneWrapper privately while still inheriting publicly from the Wrapper class. The SaneWrapper class will also not contain an object pointer anymore, but instead call the InsaneWrapper class's functions directly.

(c)

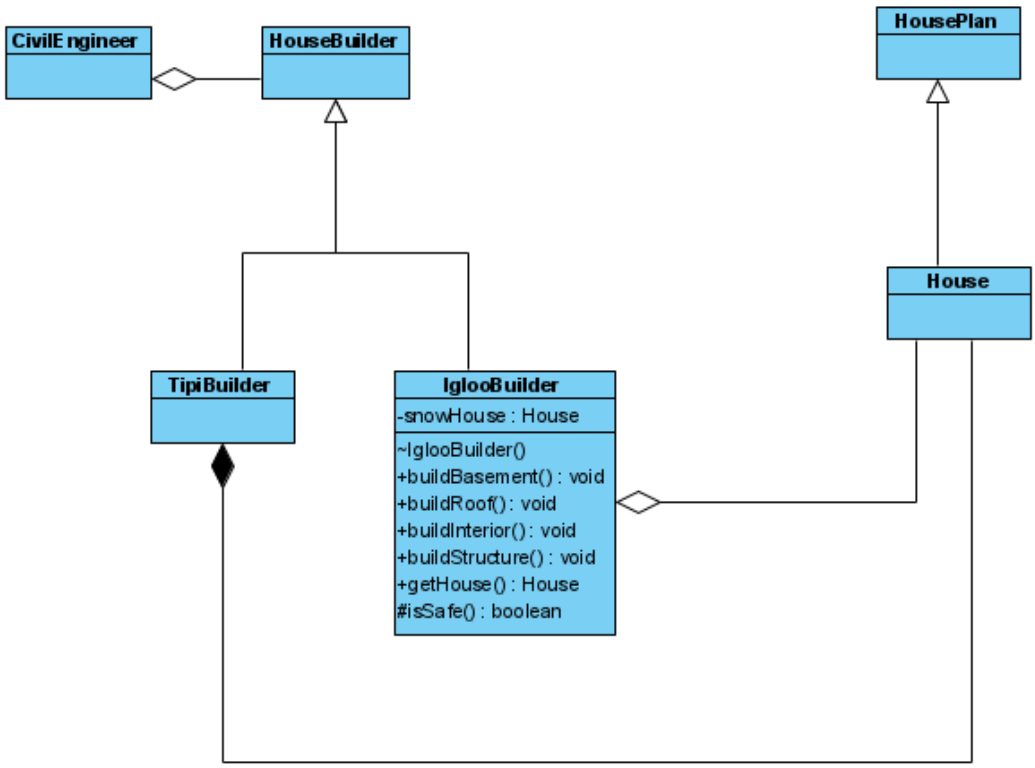
```
SaneWrapper::SaneWrapper(int i) : InsaneWrapper(i)
   {

   }
```

(d)

```
virtual void SaneWrapper::print(ostream&)
   {
       cout<<"Printing InsaneWrapper's value"<<++(&this)<<endl;
   }
```

Question 6

(a)



(b) i. Basement: None

Structure: None

Roof: None

Interior: None

---

ii. Because it's deleting the instance of snowHouse inside the buildBasement function and trying to make a replica inside getHouse, which isn't possible since House doesn't have a copy constructor specified.

iii.
```
HouseBuiler* inuk = new IglooBuilder;
CivilEngineer* eng = new CivilEngineer(inuk);
eng->construct();
dwelling = inuk->getHouse();
```

Question 7

(a)

```
Class Conversion {
    Public:
        Conversion();
    Void set(int key, char value){
        Conversion[key] = toupper(value);
    }
    String get(int key){
        return conversion[key]
    }
    Private:
    map<int,string> conversion;
    }
```

ii. α) V

β) IX

γ) XXX

δ) DCC

(b) i.

Context context;

context.set('H', 2);

context.set('T', 1);

context.set('U', 7);

ii.

```
class Tens : public RomanNumber{
        public:
        Tens();
        Virtual string interpret(Conversion, Context);
        ~Tens();
        Private:
        RomanNumber* units;

}
```

iii.

Terminal Expression: units

Nonterminal expressions: Tens and hundreds

iv. α) The value 'M' needs to be included in the conversion lookup table. That is: conversion.set(1000, 'M');

β) Add a class Thousand which has an aggregation relationship with RomanNumber representing the Hundreds. The context also needs to be updated inside the digit value for thousands as answered in the previous question.