Introduction to COS 214

Software Modelling

Linda Marshall

Department of Computer Science
University of Pretoria

16 August 2021

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

The module will introduce the concepts of **model-driven analysis and design** as a

mechanism to develop and evaluate complex software systems. Systems will be

decomposed into known entities, such as **design patterns, classes, relationships,**

execution loops and process flow, in order to model the **semantic** aspects of the

system in terms of structure and behaviour. An appropriate **tool** will be used to

support the software modelling. The role of the software model in the enterprise will

be highlighted. Students who successfully complete this module will be able to

conceptualise and analyse problems and abstract a solution.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

Four examination opportunities:

- Three during the semester - 30%

    Best EO will count 15%, Worst will
    count 5% and the other will count 10%
    There will be NO Aegrotats

- One during the exam period in
  November/December - 40%

Note: The supplementary examinations will
be written in January of 2022.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

Eight class tests at least counting 5% in total. These will be scheduled during the lecture times and will be completed on ClickUP.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

There will be five practical assignments counting 10% in total.

The first practical assignment will be released on 24 August 2021 and needs to be submitted by 31 August.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

A project will be released towards the end of the semester, at the end of October. The project will be a group project.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Module description
Evaluation
Software

Visual Paradigm 16.3 will be used as the modelling tool in this module. An academic

license is in the process of being acquired. You can use the following link to download

the software: `https://ap.visual-paradigm.com/university-of-pretoria`. Use

the activation code on the page and your student email to register the software. Note,

when using the software you need an internet connection when you start it up.

Thereafter you may disconnect from the internet.

- **COS 131 and COS 110** C++ knowledge is essential for this module
- **COS 212** Design patterns are used to implement generalised application of data structures. Design patterns "build" data structures such as lists and graphs.

Software and system modelling will form a large part of the COS 301 Software Engineering project.
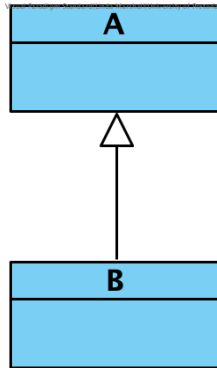
Schedule for Tuesday 24 August 2021

Basic C++ and data structure knowledge

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

```cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

```cpp
class A {
    // No access specifier implies private
    // Attributes should be private
    //    Make use of pubic setters and getters
};

class B: public A {
};
```

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

```cpp
class A {
public:
        A();
        A(string);
        void print();
private:
        string identifier;
};
class B: public A {
public:
        B();
};
A::A(string str) {
        identifier = str;
}
A::A() {
        identifier = "Class_A";
}
void A::print() {
        cout << "This_is_an_object_of_" << identifier << endl;
}
B::B() : A("Class_B"){
}
```

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

What is the output if we add destructors
that print to stdout when called and the
main program is given by:

```cpp
int main() {
        A a;
        B b;

        a.print();
        b.print();

        return 0;
}
```

Sample output:

```
This is an object of Class A
This is an object of Class B
Destructing B
Destructing A
Destructing A
```

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

What is the output if the main program is given by:

```cpp
int main() {
        A* a = new A();
        A* b = new B();

        a->print();
        b->print();

        delete b;
        delete a;

        return 0;
}
```

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

Sample output:

```
This is an object of Class A
This is an object of Class B
Destructing A
Destructing A
```

Why? How does one fix this?

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
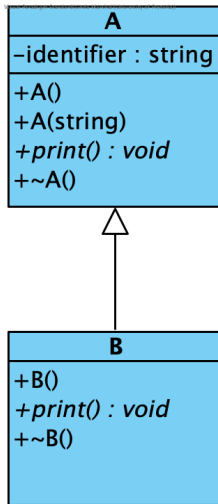**Polymorphism**
Pure virtual
Templates

What is the difference between overriding and overloading?

- **overriding** is when signature (name and parameters) of a method are the *same in the base and derived classes*.

- **overloading** is when two or more methods in the *same class have the same name, but different parameters*. If the method has a different return type, we refer to the methods has having covariant return types.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

Override `print` in B. This `print` function will not be able to access the `identifier` attribute. Which `print` will be called for the above programme? How will this be solved?

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

```cpp
class A {
public:
        A();
        A(string);
        void print();
        virtual ~A();
private:
        string identifier;
};

class B: public A {
public:
        B();
        void print();
        ~B();
};

void B::print() {
        cout << "B's_print" << endl;
}
```

Make print in A virtual.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

By making a method in a class *pure virtual*, the class becomes abstract (an interface) that cannot be instantiated.

The class name is written in *italics* in UML.

To indicate a method is pure virtual in UML, a note needs to be added to the class.

Module details
Pre-and Post-knowledge
Quick C++ refresher

Inheritance
Polymorphism
Pure virtual
Templates

```cpp
template<typename T>
T& max(T& a, T& b) {
        return a > b ? a : b;
}

cout << "Template_function" << endl;
cout << max(a,b) << endl;
cout << max<double>(24.8,13.5) << endl;
cout << max('a','b') << endl;
```

values or objects of type T must be
greater-than comparable.