Introduction to COS 214

Software Modelling

Department of Computer Science
University of Pretoria

24 July 2023

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**.

Notes are available from:

`https://www.cs.up.ac.za/cs/lmarshall/TDP/TDP.html`

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 - 40%

At least six (6) class tests counting 5% in total. These will be scheduled during the lecture times and will be completed on ClickUP.

There will be five (5) practical assignments scheduled throughout the semester counting 10% in total.

The first practical assignment will be released on 1 August 2023 when the first practical session begins.

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

Visual Paradigm 17.1 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.

- **COS132 and COS110**

  C++ knowledge is essential for this module

- **COS212**

  Data structures are used (mostly C++ containers) in the implementation of the design patterns. Some design patterns "build" data structures such as lists and graphs.

Software and system modelling will form a large part of the **COS301** Software Engineering project.

CT01 is scheduled for Thursday 27 July 2023

Basic C++ and data structure knowledge and basic UML Class and Object diagrams

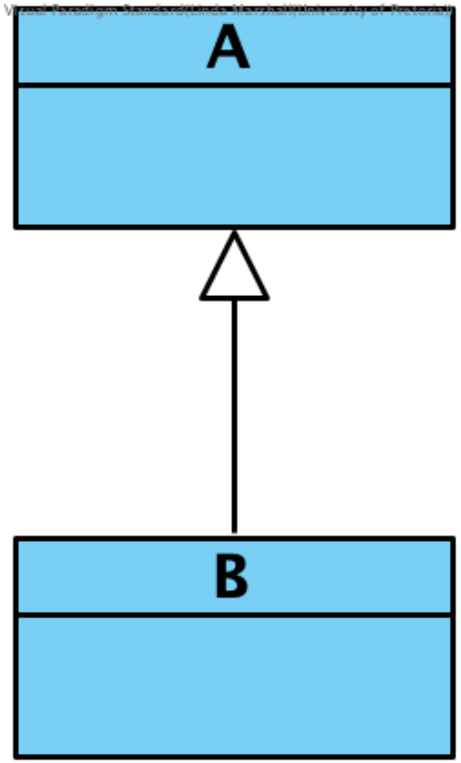
```
#include <iostream>

using namespace std;

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

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

class B: public A {
};
```


A
B

```
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"){
}
```

What is the output for the main program given by:

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

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

        return 0;
}
```

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

```
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
```

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

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

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

        delete b;
        delete a;

        return 0;
}
```

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?

What is the difference between overriding and overloading?

- **overriding** is when the signature (name and parameters) of a method are the *same in the base and derived classes*. If the method has a different return type, we refer to the methods has having **covariant return types**.
- **overloading** is when two or more methods in the *same class have the same name, but different parameters*.

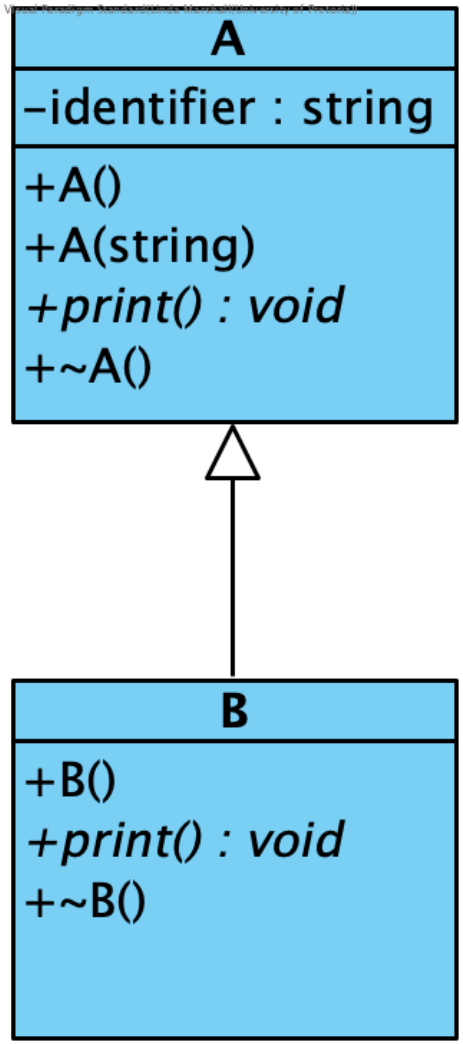
Override `print` in `B`. This `print` function will not be able to access the `identifier` attribute. Which `print` will be called for the above programme? How will this be solved?

```
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.


A
-identifier : string
+A()
+A(string)
+print() : void
+~A()
B
+B()
+print() : void
+~B()

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.

```
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*.