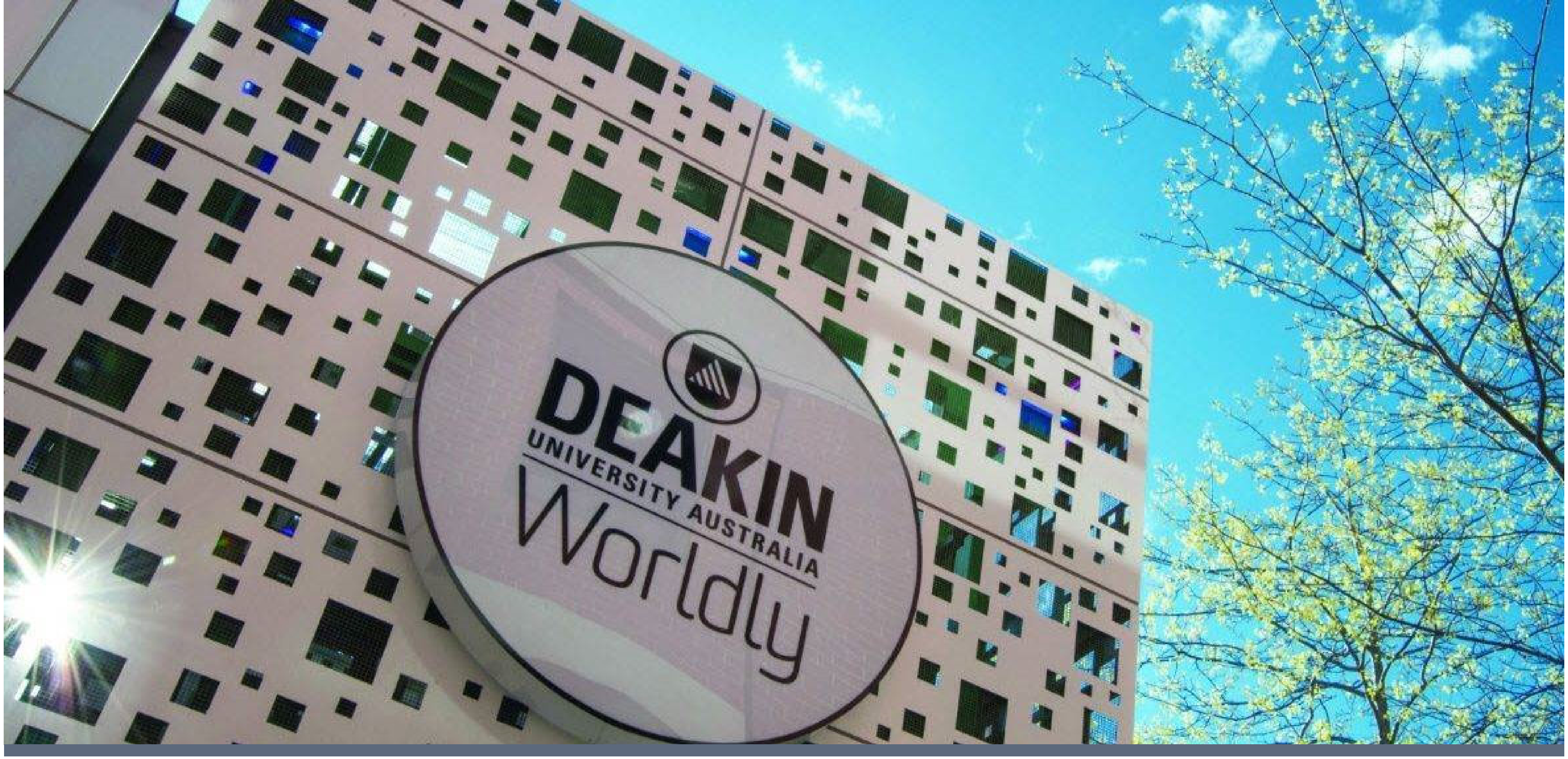


Lecture 05. Inheritance. Code Reusability

SIT232 Object-Oriented Development

Objects: What we know about them

- An object is an instance of a class, the class acts as a **template** for the object/s
- Each object has state and behavior
  - **State**: collection of attributes and their values that define the object
  - **Behavior**: what the object does
- Objects interact with each other in various ways:
  - **Communication**: objects can tell other objects to do things, that is, perform their behaviour
  - **Aggregation**: objects can contain other objects (one or many)

Appetizer: Sergey and Andrew as objects

- name: Andrew Cain
- position: Associate Head of School
- school: School of IT
- contact: andrew.cain@deakin.edu.au
- additional role: course director

- teach()
- set_exam()
- mark_exam()
- write_curriculum()

- name: Sergey Polyakovskiy
- position: Lecturer in Computer Science
  school : School of IT
- contact: sergey.polyakovskiy@deakin.edu.au

- teach()
- set_exam()
- mark_exam()

Appetizer: Differences between two classes

- Both Sergey and Andrew have common state information.
- The behaviours are somewhat similar but not entirely the same.
- Some behaviours and state information are new, e.g. write_curriculum(…).
- C# codes for both Sergey and Andrew are almost identical.
- **Duplicating code is a major source of errors**.

Inheritance: The Idea

Inheritance allows for keeping

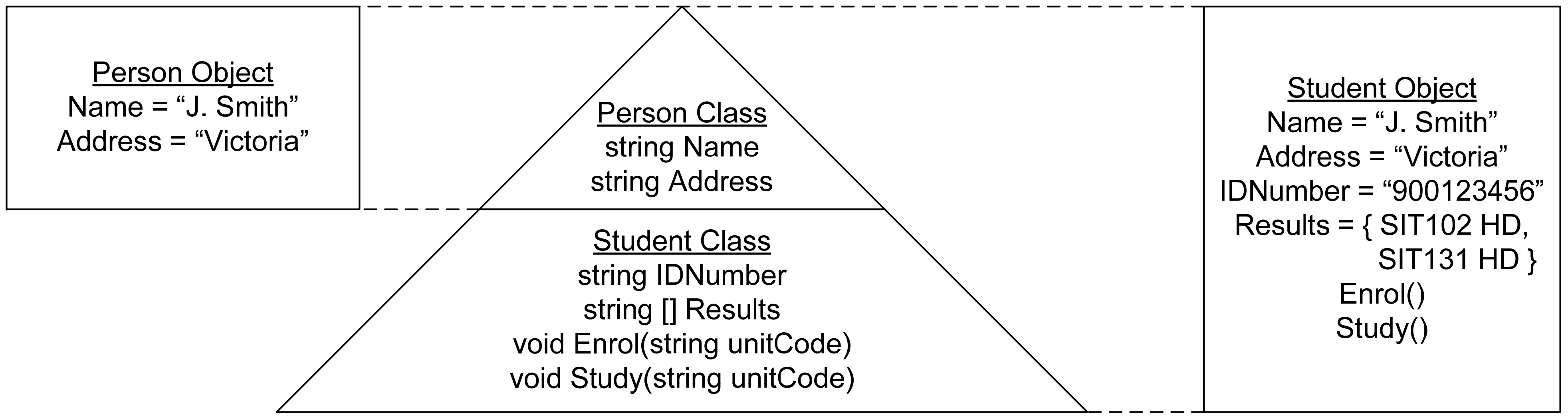
- common state and behaviour in one place (called the parent or the superclass) and
- different state and behaviour in the individual classes that inherit from the parent (called child or subclass).

- **inheritance** - the process in which common state and behaviour is passed on from the superclass to the classes that inherit from it.
- **parent/superclass** - the class that has all common state and behaviour and from which the rest inherit.
- **child/subclass** - a class that has the common state and behaviour from the superclass and adds specific state and behaviour for itself.

Inheritance: Example

- We can define a superclass Person that captures common state and behaviour from which **Lecturer** (implementing Sergey) and **CourseDirector** (implementing Andrew) inherit.
- What are the common state and behaviour that persons will have?
- What about the differences in Lecturer and CourseDirector?

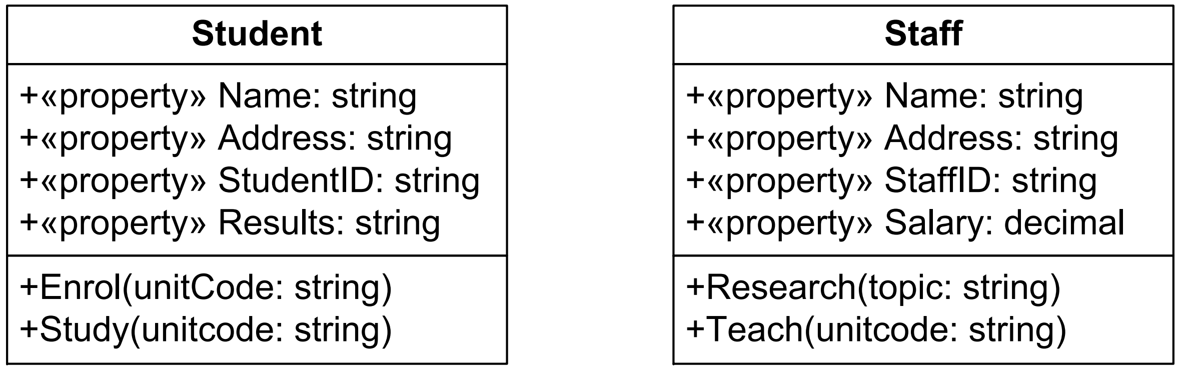
Inheritance: How to identify it

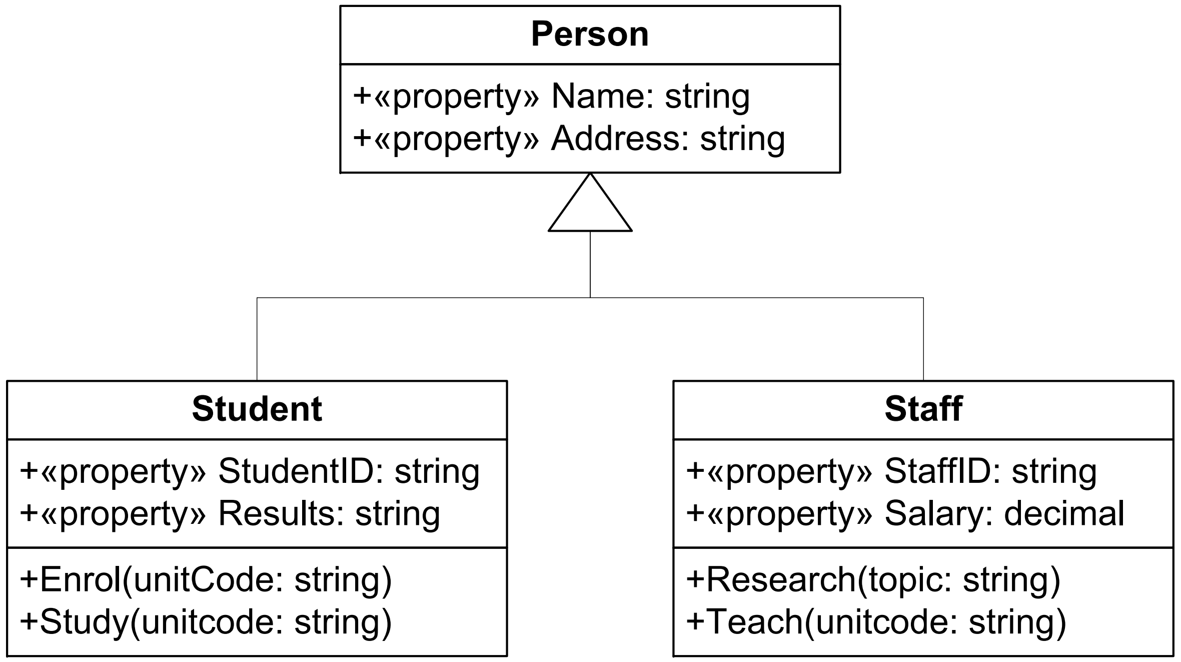


Two important phrases to identify inheritance:

- A Student is a Person
- A Student is a kind of Person

Inheritance in UML


Student
+«property» Name: string
+«property» Address: string
+«property» StudentID: string
+«property» Results: string
+Enrol(unitCode: string)
+Study(unitcode: string)
Staff
+«property» Name: string
+«property» Address: string
+«property» StaffID: string
+«property» Salary: decimal
+Research(topic: string)
+Teach(unitcode: string)



Inheritance: Syntax

```
[access_modifier] class derived_class_name : base_class_name
{
    [access_modifier] class_member
    ...
}
```

access_modifier:

- **public** - a public member is accessible from anywhere outside the class. If a variable is public, you can set/read its value directly.
- **private** (by default) - a private member is not accessible (not even for reads) from outside the class; only the class can access private members.
- **protected** - a protected member is accessible to methods of the class and sub-classes of the class.

Inheritance: Summary

- Inheritance keeps common state and behaviour in the superclass.
- Inheritance keeps different or specialized state and behaviour in the subclasses.
- Behaviour from the superclass is available in the subclasses, but not the other way around.

Code Reusability: Advantages

Reusability – the ability to exploit/reuse previously developed code in the construction of new solutions

Advantages of reusability include:

- **A reduction in development time** – reused code is already complete and does not need to be developed from scratch;
- **A reduction in testing time** – reused code has usually been tested thoroughly and can be relied upon in a new project;
- **A reduction in time/effort to maintain existing code** – bug fixes, etc., to code can quickly and easily be carried to all projects in which it was reused; and
- **Improved quality of code** – code that has been developed to be reusable will usually have been more carefully designed, coded, and tested.

Code Reusability: Disadvantages

Disadvantages of reusability include:

- **Reusable code takes longer to develop** than purpose-built code
- **Reused code can be restrictive**, i.e., if the code does not support a particular feature you may not be able to extend/easily extend that code to support that feature
- **Bugs in reused code will be present in all projects** using that code.
- **Reusable code is sometimes rarely, if ever, reused**, thus wasting the extra effort to develop it in the first place;
- **Reusable code can take a long time to learn and/or adapt** for (or plug-in to) a new for project;

Reusability: Guideline

- **Keep methods coherent** – methods should perform either a single function or a group of closely related functions
- **Keep methods small** – smaller more general functions are much easier to reuse than larger more application specific functions
- **Keep methods consistent** – the same basic functions should have the same names, parameter lists, etc.
- **Separate policy and implementation** – keep decision making (application specific) separate from the mechanisms/logic to implement those decisions (implementation)
- **Provide uniform coverage** – provide methods to handle all possible input possibilities, not just the expected/common ones
- **Avoid global information** – minimise the use of data read from outside of a method

Before you go: Method Overloading

- Method overloading refers to having several methods with the same name
- Such methods are differentiated by their signature
  - Number of parameters
  - Data type of parameters
  - How they are passed (input/reference/output)
- Method overloading can improve code readability, e.g.,

```
Network.SendInt32(123);
Network.SendFloat(1.23);
Network.SendString("123");
```

$\longrightarrow$

```
Network.Send(123);
Network.Send(1.23);
Network.Send("123");
```

- **Do not mix it up with overriding, the topic which we will discuss in our next lecture.**