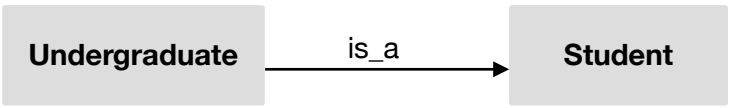
The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1
2020/2021 Semester 2
Lab 4

In this lab, we will be using several classes to illustrate the concepts of Inheritance, Method Refinement and Method Replacement. A subclass inherits state and behaviour from its super class. The state's accessibility in the subclass is determined by the access modifiers attributed to the state variables: protected or public implies full access whereas private implies no access. The behaviour of the super class that is inherited in the subclass can be:

- used as is it in the subclass
- augmented with additional behaviour (method refinement),
- completely erased and replaced with new behaviour unique to the subclass (method replacement).

Inheritance, Method Refinement and Method Replacement



1. Create a new project in BlueJ called Lab 4.
2. Retrieve the following classes from the website: **Student.java** and **StudentApp.java** Compile both java files. Run the **StudentApp** file and observe the output.
3. Create a subclass of the **Student** class called **Undergraduate.**
4. Create a new **Undergraduate** object in the **StudentApp** class with the following details:

Object	**Name**	**ID** (auto-gen)
u1	Barry Allen	30

TIP: Public methods in the parent class are inherited and can be used by a child

e.g. setName(..)

5. Print the details of the **Undergraduate** object **u1** using the **toString( )** method. What do you observe? Where did the state that is printed come from?

Answer:

Printed information in the same format as a Student object, because it inherited the toString() method from the Student (parent) class.

6. Now, add the following features to your **Undergraduate** class:
    a. Attributes: **minor (String)**, major **(String)**, credits **(int)**
    b. Accessors and mutators as necessary for each attribute
    c. A default no-argument constructor for the **Undergraduate** class.

7. Create and/or modify the following **Undergraduate** objects in the **StudentApp** class:

Object	Name	ID (auto-gen)	Major	Minor	Credits
u1	Barry Allen	30	Forensics	Athletics	25
u2	John Rambo	40	Conflict Analysis	International Affairs	20
u3	Ellen Ripley	50	Astrobiology	Conflict Analysis	15

8. Print the details of the **Undergraduate** objects **u1, u2, u3** using the **toString( )** method. What do you observe? Did all of the state print properly? Why not?

Answer:
It used the toString() method from the Student (parent) class, so only the name, ID, fees and graduated variables were printed

9. Let us now *refine* the **toString( )** method in the **Undergraduate** class. Write a **toString( )** in the **Undergraduate** class which returns the full details of an **Undergraduate** object (ID, name, fees, graduated, major, minor, credits). This method should call the **toString( )** method inherited from the parent class (Student) to achieve this.

TIP: Invoking a parent method from a child class

super.
methodName( )

10. Run the **StudentApp** class again. What do you observe? Did all of the state print properly this time? How does it differ for the objects **s1**, **s2** compared to **u1**,**u2**, **u3**?

Answer:

The format of the output was different and all of the state of the Undergraduate objects were printed properly. This was because the refined toString() method of the Undergraduate class was used when printing the state of the Undergraduate objects. It used the toString() method of its parent and added the major, minor and credits attributes.

Bonus Question: Comment off both of the toString( ) methods in the Undergraduate and Student classes. Run the StudentApp class again. Explain why the code still works and the meaning and origin of the output.

11. Overload the constructor of the **Student** class as follows:

    public Student(String name)

    The overloaded constructor should still set the state as in the original constructor.

12. Compile and run the **StudentApp** class. Did it work? Did you have to change the code for the **Student** objects **s1** and **s2** in the **StudentApp** class to suit the new constructor? Why or why not?

Answer: **Yes it worked because a no argument constructor still exists in the Student class. However, we can modify the existing code to create objects with a specified name. This is opposed to creating them with the no argument constructor and setting their names after.**

13. What happens when you compile your **Undergraduate** class? Which of the superclass constructors do you think is called by the Undergraduate constructor?

Answer:

Nothing happens really. The no argument constructor of the Student (parent) class is called when an Undergraduate object is created.

14. Comment off the no-argument constructor in the Student class. Observe the (compilation) error generated in the Undergraduate class. Explain why this happens. Explain how it can be fixed. (remove your commented code when finished

Answer:

This happens because there is no no-argument constructor present in the Student (parent) class. Therefore, we must invoke the other constructor (the one that accepts a string) to avoid this error by writing "super(name)". This ensures that the Student class can set its state properly.

15. Create a subclass of the **Student** class called **Postgraduate** with the following:
    a. Attributes: **supervisor (String), thesisTitle (String), status (String)**
    b. A constructor that accepts a **name**, **supervisor** and **thesisTitle**.
    c. A mutator for **status**
    d. The default status for a student is full-time.

Note: Do not change the access modifiers of the **Student** class from private. You need to invoke the appropriate **Student** class constructor in order to set the name.

TIP: Invoking a parent constructor from a child class

super(parameter1,..)

16. Create two **Postgraduate** objects in the **StudentApp** class with the following state

Object	Name	ID	Supervisor	Thesis Title
p1	John McClain	60	Prof. Asp Pirin	How to Die Hard
p2	Brian Mills	70	Dr. No Kia	Mobile Usage Patterns in Hostage Situations

17. In the **Postgraduate** class, *refine* the inherited **toString( )** method from the **Student** so that the full details of a **Postgraduate** object are returned. Print the details of the **Postgraduate** objects **p1** and **p2** in the **StudentApp** class using your refined **toString( )**.
18. Override the method **calculateFees( )** in the **Undergraduate** class so that **Undergraduate** tuition fees are calculated based on the number of credits. Each credit costs $200.00.
19. Override the method **calculateFees( )** in the **Postgraduate** class so that
    - part-time **Postgraduate** tuition fees amount to $1,325.00
    - full-time **Postgraduate** tuition fees amount to $2,650.00

TIP: Overriding a method in a child class is the same as replacing the method inherited from the parent class: the same method signature is used

20. Change the status of the **Undergraduate** and **Postgraduate** objects in the **StudentApp** class as follows:
    a. John McClain: full-time
    b. Brian Mills: part-time
    c. Barry Allen: full-time
    d. John Rambo: 25 credits
    e. Ellen Ripley: 20 credits

    Calculate their fees, and print their details once more.What do you observe?

Answer:
Several changes in credits and statuses led to the fees of 4 students being changed

21. What happens when you invoke the **calculateFees( )** method on the **Student** objects **s1** and **s2**?

Answer: **Nothing**