INFO1113/COMP9003 Week 5 Tutorial

Inheritance

Inheritance

Inheritance is a foundational feature in object oriented programming. Within Java it allows classes to subtype other classes and **inherit** the properties of its parent type. This allows for greater code reuse as the subtype is able to inherit methods and attributes from its parent type that can be used within its own class.

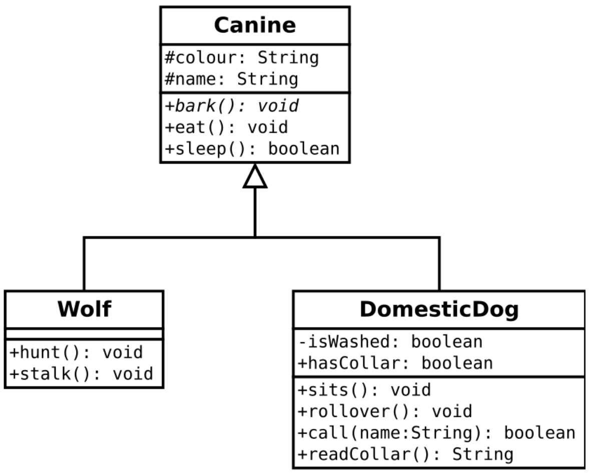

Figure 1: UML Class Diagram illustrating an inheritance heirarchy

As you may observe in the following class, that both `DomesticDog` and `Wolf` inherit from `Canine`. Any protected # or public + property is inherited by the sub-classes. This allows `Wolf` and `DomesticDog` access to `color`, `name`, `bark()`, `eat()` and `sleep()` methods from the parent class.

We would not have to rewrite any of these attributes for every subtype and any change that may occur will only occur in one place rather than multiple. The classes are also able to make a clear distinction between what properties define a `DomesticDog` in contrast to a `Wolf`.

Discussion: Cards

Consider the following attributes of various cards you might find in your wallet:

Credit Card:

- owner' name
- bank name
- card number
- expiry date
- security chip

Student Card:

- student name
- student ID
- year of issue
- magnetic strip

Driver's License:

- driver's name
- address
- card number
- state of issue
- license number

Part 1. What are the similarities between the cards?

Part 2. How are the cards used differently? How is data accessed from each of the cards?

Part 3. Discuss with your peers the the class definition below. What difficulties are going to be faced when using this class to represent all kinds of card (credit card, student card, license, etc.)?

Would this class be useful for maintaining a collection of `Cards`, for example in a Wallet?

```java
public class Card {
        private String cardType;
        private String[] data;

        public Card(String cardType, String[] cardInformation) {
                this.cardType = cardType;
                this.data = cardINformation;
        }

        public String getCardType() {
                return cardType;
        }

        public void setCardType(String type) {
                this.cardType = type;
        }

        public String[] getInformation {
                return data;
        }

        public void setInformation(String[] cardInformation) {
                this.data = cardInformation;
        }
}
```

Part 4. Define a more suitable `Card` class that contains only the common atributes from each of the card types.

Part 5. Define a subclass for each of the card types described. These subclasses should use the `super` constructor from their parent class in order to initialise the common variables, as well as initialising their own specific variables.

Discussion

One of your peers has given you a UML Class Diagram to criticise. Discuss with your class, tutor, friend or dog about the issues with the design.

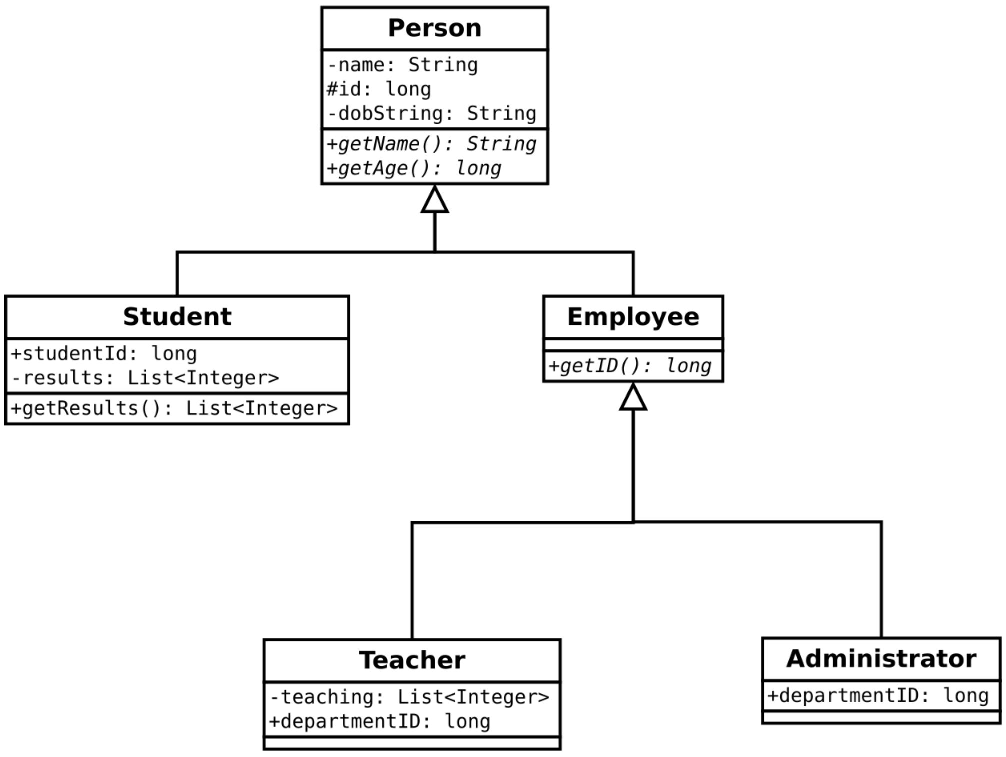

Figure 2: Your peer's work

- What attributes are not inherited?
- Outline the redundant attributes from the class diagram
- What properties of the classes would you fix?

Question 1: Bank Account

A bank account has an account number, BSB, balance and interest rate. The account can be opened with an initial balance. It can perform several transactions: earn interest, withdraw, deposit or transfer to another account.

```
Account number: 123456789
BSB: 123456
Balance: $100.0
Interest Rate: 0.5%
```

A transaction has a date, a text description, and an amount (positive for a credit, negative for a debit).

```
Date: 01/01/2022
Description: Rent
Amount: -$400
```

Create two classes - a `BankAccount` and a `Transaction` class to capture the properties described above. Add a method to describe a transaction being performed on the bank account. You should return the new balance after the transaction has been performed.

```java
public double performTransaction(Transaction transaction) {
        //write your code here
}
```

Let us consider a special type of bank account that will be a child to the normal bank account - a savings account. A savings account has all the features of a normal bank account but it can also earn bonus interest. A user earns bonus interest if they deposit more than $200 that month. A savings account would look like:

```
Account number: 123456789
BSB: 123456
Balance: \$100.0
Interest Rate: 1%
Bonus Interest Rate: 2%
```

Create a new class `SavingsAccount`. Make sure it inherits from `BankAccount` to avoid repeat a lot of the same code. Add in the new properties and methods that are unique to a `SavingsAccount`.

Question 2: SimpleDate

Download the date class `SimpleDate` from Ed resources to store date information in your program for question 1. Complete the following methods for the correct functionality.

```java
public class SimpleDate {
        private int d, m, y;

        public SimpleDate(int d, int m, int y) {
                this.d = d;
                this.m = m;
                this.y = y;
        }
        // return true when this date < other date
        public boolean isLessThan ( SimpleDate other ) {

        }

        public boolean isEqualTo ( SimpleDate other ) { ... }
        public boolean isLessThanOrEqualTo( SimpleDate other ) { ... }
        public boolean isGreaterThan ( SimpleDate other ) { ... }
        public boolean isGreaterThanOrEqual( SimpleDate other ) { ... }
}
```

Question 3: Assessed Task: Online Task 4

Remember you are required to complete a Online Task within the due date. Go to EdStem for this unit and click on Lessons to find out the task and the due date. This is a marked task. Note that you are allowed to submit multiple times but only the last one will be marked.