ICT167 ANS3

Information hiding:

- Allows client to understand what the method is doing but not how the method does what it does
- This involves designing a method so that in order to use it, a client doesn't need to look at the code in the method body
    - A comment at the start of the method should indicate and clearly describe to the client what the method does
- Allows a team to easily divide up their work
- Applies to whole classes as well as methods

- Use Private variables (ICT167Ans2)
- Use assessors and mutators

Pre/Post condition comments: (low level documentation comments)

- The client (user of a method of a class) will want to know- the name of method, return type of the method, number/type/order of the parameters of the method and clear description of what the method is supposed to do
- It is a *contract* between the creator of the class and the user (client) of the class

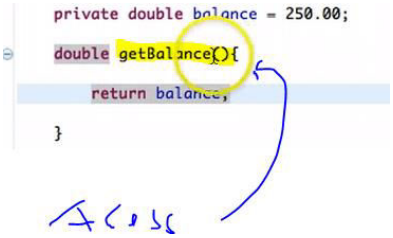
Characteristics of Pre/Post condition comments: [EXAM?]

- The client produce some arguments satisfying certain conditions, *Pre-condition*
- Pre-condition: for a method states the conditions that must be true before the method is invoked. If pre-condition is not correct then processing may be wrong. But since the client worked out the pre-conditions it's their fault. The processing was designed for their requested pre-conditions
    - EG: (To use method correctly must pass an) Years is a integer between the range of 0,100
- Once the pre-conditions are stated the creator will do some processing according to the pre-conditions, this is called the *Post-condition*
- Post-condition: describes the effect of the method call when the method is invoked. Includes the return value properties
    - Returns the projected population of the calling (receiving) object after the specified number of years

Assertions: assertions is a statement about the state of a program e.g. - pre-conditions. Used for trouble shooting

Accessor Methods: (Used to access/modify private instance variables) (getters)

- Commonly referred to as the get methods or getters, it is a method that access an object and returns some information about it without changing the state of that object (its instance variable). *So purpose is to get the value being stored in the private instance variable*



Mutator Methods: (deal with private instance variables. Just for setting objects)

- Commonly referred to as set methods, it is a method that modifies the state of an object (private instance variable)
    - EG: SetSpecifies(String newName, int newPopulation), readInput()
- Should have a return type of void

Advantages of Assesor and Mutators :

- The use of mutator methods allow you to validate whether the new value is legitimate
- Mutators allows us to update a group of variables
- Allows the client to change the name of the private instance variable but keep the old mutator method and accessor method still functioning the same correctly (a simple change name of private variable and name of accessor/mutate method)

Immutable classes: Classes that have only accessor methods and no mutator method e.g string class

Helper method: (refer to future lecture) they help other methods in the class and you don't want client to know. Any method in the class declared as private. Helper methods helps handle complex calculations. They declared as private. Think private methods

Data abstraction: Data abstraction allows the data in the program to be organised by putting related data values together and having a simple name for the big lump of data that we call

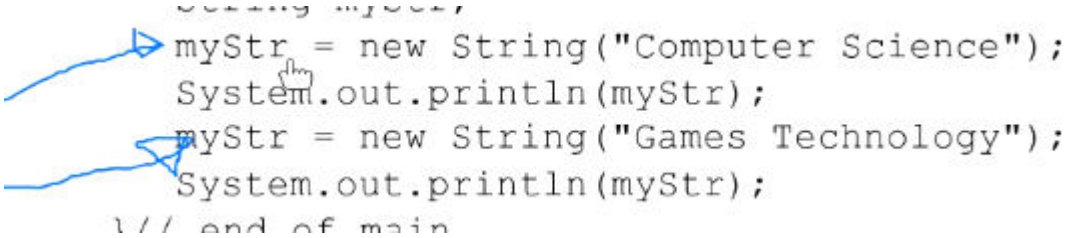
- EG: Species (data without method)

Encapsulation: Encapsulation allows the data in the program to be organised by putting related data values AND actions that manipulate with the data (i.e methods) together in one item.

- EG: string class lump characters together as a string and has methods

Explaining garbage: [IMPORTANT]

- Assigning new objects to an already assigned variable will cause the older assignments to be overridden. The first reference to the first object is now garbage unlike primitive
- Instantiating two objects with the same object reference (variable) will only allow access to the second memory location while the first memory location is treated as garbage (refer below)
- #read Lecture slide 55/56



Calling object: refers to the object that calls the method. So the object i.e [class variable]

```
[class variable].method(…);
```

Understanding multiple objects of the same class:

```
1   Fractions firstFrac = new Fractions();
2   Fractions secondFrac = new Fractions();
3
4   firstFrac.input();
5   firstFrac.display();
6
7   secondFrac.input();
8   secondFrac.display()
9
10  firstFrac.isEqual(secondFrac);
```

- Two objects. firstFrac and secondFrac.
- Both of the objects STORE/HOLD the same → Methods FROM class: Fractions
    - .input()
    - .display()
    - .isEqual()

- Both of the objects STORE/HOLD may store different values for THE class instance variables
    - firstFrac
        - Numerator = 10
        - Denominator = 12
    - secondFrac
        - Numerator = 11

- Denominator = 15

Thus to access objects of the same class instance variable-

```
firstFrac.numerator

secondFrac.numerator
```

Pass a class type variable (object) to a method-

Client program:

```
[class variable 1/Object1].method([class variable 2/Object2]);
```

Class:

```
1  public void method([Class Name] otherFrac) { //The paramater refers to class: Fraction5 but object: otherFrac
2
3          this.Numerator == otherFrac.Numerator;
4
5  }
```