# Introduction To Object-Oriented Programming

Part III: You will learn the standard graphical notation for representing classes (UML), why should attributes be set to private in a class definition, what documentation to write for each class, good style conventions for writing functions or methods.

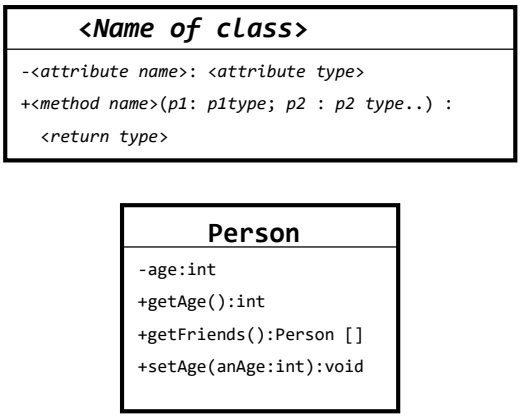
James Tam

## Graphical Summary Of Classes

- UML (Unified modeling language) class diagram
  - Source "*Fundamentals of Object-Oriented Design in UML*" by Booch, Jacobson, Rumbaugh (Dorset House Publishing: a division of Pearson) 2000
  - UML class diagram provides a quick overview about a class (later you we'll talk about relationships between classes)
- There's many resources on the Safari website:
  - http://proquest.safaribooksonline.com.ezproxy.lib.ucalgary.ca/
  - Example "**Sams Teach Yourself UML in 24 Hours, Third Edition" (concepts)**
  - Hour 3: Working with Object-Orientation
  - Hour 4: Relationships
  - Hour 5: Interfaces (reference for a later section of notes "hierarchies")

James Tam

# UML Class Diagram




James Tam

# Why Bother With UML?

- It combined a number of different approaches and has become the standard notation.
- It's the standard way of specifying the major parts of a software project.
- Graphical summaries can *provide a useful overview* of a program (especially if relationships must be modeled)
  - Just don't over specify details

James Tam

## Back To The 'Private' Keyword

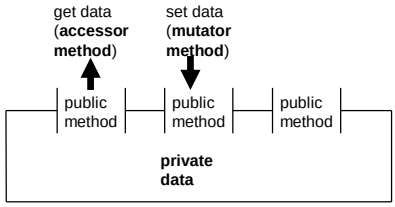
- It syntactically means this part of the class cannot be accessed outside of the class definition.
  - You should **always** do this for variable attributes, *very rarely do this* for methods (more later).
- Example

```
public class Person {
     private int age;
     public Person() {
         age = 12;   // OK – access allowed here
     }
}
public class Driver {
     public static void main(String [] args) {
         Person aPerson = new Person();
         aPerson.age = 12;  // Syntax error: program won't
                            // compile!
     }
}
```

James Tam

## New Term: Encapsulation/Information Hiding

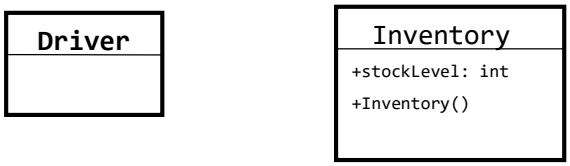
- Protects the inner-workings (data) of a class.
- Only allow access to the core of an object in a controlled fashion (use the *public* parts to access the *private* sections).
  - Typically it means public methods accessing private attributes via accessor and mutator methods.
  - Controlled access to attributes:
    - Can prevent invalid states
    - Reduce runtime and logic errors




James Tam

## How Does Hiding Information Protect Data?

- Protects the inner-workings (data) of a class
  - e.g., range checking for inventory levels (0 – 100)
- **Name of the folder containing the complete example**: `fifth_noProtection`




James Tam

## Class Inventory

```
public class Inventory
{
    public int stockLevel;

    public Inventory()
    {
        stockLevel = 0;
    }
}
```

James Tam

# Class Driver

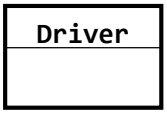
```
public class Driver
{
    public static void main(String [] args)
    {
        Inventory chinook = new Inventory();
        chinook.stockLevel = 10;
        System.out.println("Stock: " + chinook.stockLevel);
        chinook.stockLevel = chinook.stockLevel + 10;
        System.out.println("Stock: " + chinook.stockLevel);
        chinook.stockLevel = chinook.stockLevel + 100;
        System.out.println("Stock: " + chinook.stockLevel);
        chinook.stockLevel = chinook.stockLevel - 1000;
        System.out.println("Stock: " + chinook.stockLevel);
    }
}
```

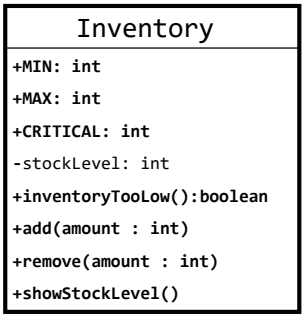
```
Stock: 10
Stock: 20
Stock: 120
Stock: -880
```

James Tam

# Utilizing Information Hiding: An Example

- **Name of the folder containing the complete example:** `sixth_encapsulation`




James Tam

# Class Inventory

```
public class Inventory
{
    public final int CRITICAL = 10;
    public final int MIN = 0;
    public final int MAX = 100;
    private int stockLevel = 0;

    public boolean inventoryTooLow()
    {
         if (stockLevel < CRITICAL)
              return(true);
         else
              return(false);
    }
```

James Tam

# Class Inventory (2)

```
    public void add(int amount)
    {
        int temp;
        temp = stockLevel + amount;
        if (temp > MAX)
        {
            System.out.println();
            System.out.print("Adding " + amount +
                             " item will cause stock ");
            System.out.println("to become greater than " + MAX + "
                                units (overstock)");
        }
        else
        {
            stockLevel = temp;
        }
    }
```

James Tam

# Class Inventory (3)

```
    public void remove(int amount)
    {
        int temp;
        temp = stockLevel - amount;
        if (temp < MIN)
        {
             System.out.print("Removing " + amount +
                              " item will cause stock ");
            System.out.println("to become less than " + MIN + " units
                               (understock)");
        }
        else
        {
            stockLevel = temp;
        }
    }

    public String showStockLevel()
    {
        return("Inventory: " + stockLevel);
    }
}
```

James Tam

# The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
         Inventory chinook = new Inventory();
         chinook.add(10);
        System.out.println(chinook.showStockLevel());
         chinook.add(10);
        System.out.println(chinook.showStockLevel());
         chinook.add(100);
        System.out.println(chinook.showStockLevel());
         chinook.remove(21);
        System.out.println(chinook.showStockLevel());
         // JT: The statement below won't work and for good reason!
        // chinook.stockLevel = -999;
    }
}
```

```
Inventory: 10
Inventory: 20
Inventory: 20
Inventory: 20
```

James Tam

## Add( ): Try Adding 100 items to 20 items

```
public void add(int amount)
{
    int temp;
    temp = stockLevel + amount;
    if (temp > MAX)
    {
        System.out.println();
        System.out.print("Adding " + amount +
                         " item will cause stock ");
        System.out.println("to become greater than " + MAX +
                           " units (overstock)");
    }
    else
    {
        stockLevel = temp;
    }
} // End of method add
```

```
Adding 100 item will cause stock to
become greater than 100 units (overstock)
```

James Tam

## Remove( ): Try To Remove 21 Items From 20 Items

```
public void remove(int amount)
{
    int temp;
    temp = stockLevel - amount;
    if (temp < MIN)
    {
         System.out.print("Removing " + amount +
                          " item will cause stock ");
        System.out.println("to become less than " + MIN + " units
            (understock)");
    }
    else
    {
        stockLevel = temp;
    }
}

public String showStockLevel()
{
    return("Inventory: " + stockLevel);
}
}
```

```
Removing 21 item will cause stock to
become less than 0 units (understock)
```

James Tam

## Documenting Methods

- Course requirement: Methods are a 'mini' program so the manner in which an entire program is documented should also repeated in a similar process for each method:
  - Features list.
  - Limitations, assumptions e.g., if a function will divide two parameters then the documentation should indicate that the method requires that the denominator is not zero.
  - (Authorship and version number may or may not be necessary for the purposes of this class although they are often included in actual practice).
- Another common requirement:
  - The number and type of parameters e.g. `display(int,String)`
  - The return type: //returns(int)

James Tam

## Good Style: Functions/Methods

1. Each function should have one well defined task. If it doesn't then this may be a sign that the function should be decomposed into multiple sub-functions.
   a) Clear function: A function that squares a number.
   b) Ambiguous function: A function that calculates the square and the cube of a number.

   Writing a function that is too specific makes it less useful (in this case what if we wanted to perform one operation but not the other).
   - Also functions that perform multiple tasks can be harder to test.

James Tam

## Good Style: Functions/Methods (2)

2. (Related to the previous point). Functions should have a self-descriptive action-oriented name (verb/action phrase or take the form of a question – the latter for functions that check if something is true): the name of the function should provide a clear indication to the reader what task is performed by the function.
   a) Good: `drawShape()`, `toUpper()`
      `isNum()`, `isUpper()` **# Boolean functions: ask questions**
   a) Bad: `doIt()`, `go()`, `a()`

James Tam

## Good Style: Functions (3)

3. Try to avoid writing functions that are longer than one screen in length.
   a) Tracing functions that span multiple screens is more difficult.
4. The conventions for naming variables should also be applied in the naming of functions.
   a) Lower case characters only.
   b) With functions that are named using multiple words capitalize the first letter of each word except the first (so called "camel case") - most common approach or use the underscore (less common). Example: `toUpper()`

James Tam

## Example Method For Decomposing A Long Function/Method

- Look for 'blocks' of code that can be moved to a separate function (e.g. body of a branch, loop etc.)
- Move that block of code to it's own separate function.
- Be careful of scope! (You may need to pass local variables into the new function and return updates).

**Before**

```
def fun1():
    if ():
        ...
    else:
        while():
        # many statements
```

**After**

```
def fun1():
    if ():
        ...
    else:
        fun2()

def fun2():
    while ():
    # many statements
```

James Tam

## New Terms And Definitions

- Encapsulation/information hiding

James Tam

## After This Section You Should Now Know

- How to represent a class using class diagrams (attributes, methods and access permissions) and the relationships between classes
- What is encapsulation/information-hiding, how is it done and why is it important to write programs that follow this principle

James Tam

## Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."



James Tam