**ITMD\_510 Week 2 Notes –1/17/18 Wed.**

**Java Review – OOP Basics**

**[ OOP ]**

OOP is about how objects can communicate.

**Object-oriented programming** (**OOP**) is a programming paradigm based on the **concept** of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

**Class structure / terminologies**

ClassName {

Camel case and upper case for class name

//class data members or “fields” (**static** or **non-static** fields or members)

/\*class method members (**static** vs. **instance**)\*/

Interest rate for example can be static

//constructors

Starts the members but has default values

//Getters

Get a single value

//Setters

Set a value with a parameter

//utility functions

}

**2 PRIMARY TYPES OF METHODS | Static Method or non-static method (called instance method)**

Instance methods

Methods which *require* an object of its class to be created before it can be called. To invoke or *call* a instance method, an Object of the class is needed.

**Format of function call**

**object.methodName()**

**Ex. of a function call**

         // create an instance of the class.

**Foo ob = new Foo();**

Once the object is created, the constructor is automatically fired

         // calling an instance method in the class 'Foo'.

**ob.data("frontier");**

Static methods

Methods that can be called *without* creating an object of class. They are referenced by the **class name itself** or reference to the Object of that class.

Use of keyword **static** when creating a variable or function

**Ex,**

**static int count = 0;**

**public static methodName**

**Format of function call**

ClassName.methodName(args).

**Ex. of a function call**

        // Accessing the static method data() and

         // field by class name itself.

**Foo.data("ITMD");**

**System.out.println(Foo.dataVal);**

**Important Takeaways**

* Static method(s) are associated to the class in which they reside i.e. they can be called even without creating an instance of the class i.e **ClassName.methodName(args).**
* Static data members are designed with aim to be **shared** among all Objects created from the same class.
* Static methods can not be overridden. But can be overloaded since they are resolved using **static binding** by compiler at compile time.

**Other methods**

**Constructor**

* **Carry the same name as the class**
* **Can be overloaded**
* **Cannot be inherited**
* **Place an object (data members) into an intial state**

**Getters & Setters**

Getters *gets* a data member’s value

**Ex.**

**ob.getName();**

Setters *set*s data member(s)

**Ex.**

**ob.setName(“Sammy Student”);**

 **FUTURE STUDY**

**Know differences between method overloading and method overriding!**

**Give examples!**

**For future:**

**What is polymorphism?**

**What is early binding vs. late binding?**

**[ OOD ]**

OOD is about how objects depend on each other.

Solid Object Oriented Design should do for your code follow…

1. Remove Rigidity   – Software that is not flexible takes a lot of effort to change and maintain
2. Decrease Fragility – Making code less fragile means it can adapt instead of break.
3. Increase Mobility  – Moving code should be easy, code duplication makes this difficult or even impossible. TDD!
4. Decrease Viscosity – Make the code more agile, make algorithms tight, slow code is no code
5. Remove Needless Complexity – Don’t go planning for every single case, get the bulk of it and handle ANY thing else gracefully

**[ UML (Unified Modeleing Language) ]**

The **Unified Modeling Language** (**UML**) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

|  |
| --- |
| **Class Name** |
| **Data Members** |
| **Method Member** |

**Symbols**

**+ for *public* mems.**

**- for *private* mems.**

**# for *protected* mems.**

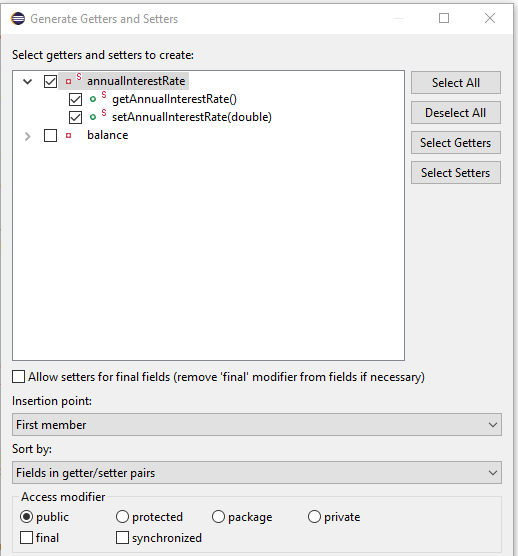
**[ IDE charms ]**

For **Eclipse** IDE:

**Source > Organize Imports**

**Source > Generate Getters and Setters…**

Ex. for **AccountHolder** class



**[ Java Docs ]**

Comments can be **single line** or **mulitline**

**//**single line comment here

**/\***Mulitline comments

Right here **\*/**

Methods should *always* be commented!!

Ex.

Place JUST before your method AFTER you’ve created it, a **/\*\***

Ex.

/\*\*

\*

\* **@param** balance

\*/

**public** AccountHolder(**double** balance)

{

}

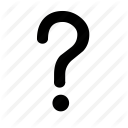
It auto creates your comments!

To view your Java Docs…

**Project > Generate Javadoc…**

Try it out and note what is generated and how to view it. It should be in the form of an HTML file!

More on this next week!



What are comments good for? Two things should come to mind!!!

**[ JUnit / TDD (Test Driven Development)]**

**Next week breakdown of Unit testing!**



**Think on what tests can be performed on Lab 1 class AccountHolder?**

**Look at the methods involved.**

**Check for balances (+ vs. -) when methods (constructor, deposit, withdrawal, monthlyInterest, etc) execute.**