Object Orientation– Basic Concepts

Topic # 1

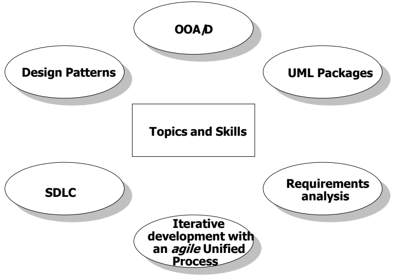
Today’s Agenda

- Administrative Stuff
- Overview of CS324
- Introduction to object oriented analysis
- Why OOAD?

About the course

- Study application of a software engineering approach that models and designs a system as a group of interacting objects.
- Various cases studies will be used throughout the course to demonstrate the concepts learnt.
- A strong in class participation from the students will be encouraged and required during the discussion on these case studies.

Major Topics of the course



OOA/D

Design Patterns

UML Packages

Topics and Skills

SDLC

Requirements analysis

Iterative development with an *agile* Unified Process

Course Outline

- – Course Introduction
- – SDLC
- – RUP
- – UML Relationships
  - Association
  - Aggregation
  - Composition
- – UML Packages
  - Use Case
  - Class Diagram
  - Activity Diagram
  - Collaboration Diagram
  - Sequence Diagram
  - State Transition Diagrams
- – Design Patterns

Pre-requisites /Knowledge assumed

- Programming language concepts
- Data structure concepts
- We assume you have the skills to code in any programming language therefore you
  - – can design, implement, test, debug, read, understand and document the programs.

Course Material

- You will have Presentations of each topic and ***reference books*** in PDF format will be available on slate.

Text Books

1. UML 2 Toolkit by Hans-Erik Eriksson, Magnus Penker, Brian Lyons, David Fado

2. Applying UML and Patterns 3rd Edition by Craig Larman

3. Software Engineering: A practioner's Approach, Roger Pressman, McGraw-Hill, Eigth Edition.

4. System Analysis & Design Methods, 9th Edition, By Whitten, Bentley, Dittman

Reference Books

1. UML and the Unified Process, Practical object-oriented analysis and design by Jim Arlow, Ila Neustadt

2. The Unified Modeling Language Reference Manual, 2nd edition by James Rumbaugh, Ivar Jacobson and Grady Booch

3. UML Distilled, 2nd Edition by Martin Flower

- Hands-on labs will be part of the course.

Course Goals/Objectives

- By the end of this course, you will be able to
  - Perform analysis on a given domain and come up with an Object Oriented Design (OOD).
  - Practice various techniques which are commonly used in analysis and design phases in the software industry.
  - Use Unified Modeling Language (UML) as a tool to demonstrate the analysis and design ideas
  - Analyze, design and implement practical systems of up to average complexity within a team and develop a software engineering mindset

Course Page

FACEBOOK GROUP PAGE:

SDA Fall 2020 by Ubaid Aftab

Basic OOP Concepts and Terms

Objects

- Most basic component of OO design. Objects are designed to do a *small*, specific piece of work.

- Objects represent the various components of a business system

Examples of Different Object Types

- GUI objects
  - objects that make up the user interface
  - e.g. buttons, labels, windows, etc.
- Problem Domain objects
  - Objects that represent a business application
  - A problem domain is the scope of what the system to be built will solve. It is the business application.
    - e.g. An order-entry system
      A payroll system
      A student system

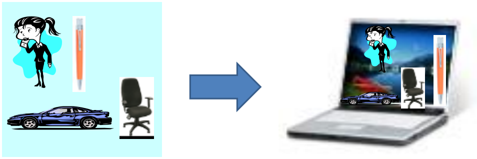
Classes and Objects

- **Classes**
  - **Define what all objects of the class represent**
  - **It is like a blueprint. It describes what the objects look like**
  - **They are a way for programs to model the real world**
- **Objects**
  - **Are the instances of the class**

What is Object-Orientation?

- What is Object?

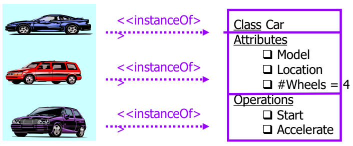
- An "object" is anything to which a concept applies, ***in our awareness***
- Things drawn from the problem domain or solution space.
  - E.g., a living person in the problem domain, a software component in the solution space.



- A structure that has identity and properties and behavior
- It is an instance of a collective concept, i.e., a class.

What is Object-Orientation?

- Class



- **What is *CLASS?***
  - a collection of objects that share common properties, attributes, behavior and semantics, in general. ***What are all these???***
  - A collection of objects with the same data structure (attributes, state variables) and behavior (function/code/operations) in the solution space.
- **Classification**
  - Grouping of common objects into a class
- Instantiation.
  - The act of creating an instance.

What is Object-Orientation

- Abstract Class vs. Concrete Class

- Abstract Class.
  - An *incomplete* superclass that defines common parts.
  - Not instantiated.
- Concrete class.
  - Is a *complete* class.
  - Describes a concept completely.
  - Is intended to be instantiated.
  - methods may be:
    - defined in the class or
    - inherited from a super-class

What is Object-Orientation

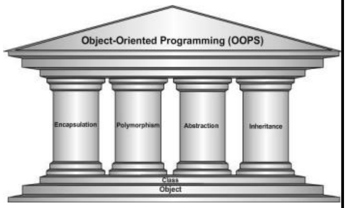
- Abstraction and Encapsulation

Abstraction
Focus on the essential
Omits tremendous amount of details
...Focus on what an object "is and does"

Encapsulation
a.k.a. information hiding
Objects encapsulate:
property
behavior as a collection of methods invoked by messages
…state as a collection of instance variables

OO Principles

1) Abstraction
2) Encapsulation
3) Inheritance
4) Polymorphism



Principle 1: Abstraction

- Its main goal is to handle complexity by hiding unnecessary details from the user.
- Applying abstraction means that each object should only expose a high-level mechanism for using it.
- This mechanism should hide internal implementation details. It should only reveal operations relevant for the other objects

Principle 2: Encapsulation

- Encapsulation separates implementation from users/clients.
- Objects have attributes and methods combined into one unit
- Encapsulation is the ability to package data, related behavior in an object bundle and control/restrict
- access to them (both data and function) from other objects.
- It is all about packaging related stufftogether and hide them from external elements.
- Encapsulation is achieved when each object keeps its state private, inside a class. Other objects don't have direct access to this state. Instead, they can only call a list of public functions — called methods.

Principle 3: Inheritance

- It is the mechanism of basing an object or class upon another object or class, retaining similar implementation.
- One class of objects takes on characteristics of another class and extends them
- Superclass → subclass
  - The derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class.
  - The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class or can add additional implementation of existing.
- Generalization/specialization hierarchy
  - Also called an inheritance hierarchy
  - Result of extending class into more specific subclasses
- **This is an important concept!!**

Principle 4: Polymorphism

- Literally means "many forms"
- Polymorphism means to process objects differently based on their data type.
  - In other words it means, one method with multiple implementation, for a certain class of action.
  - And which implementation to be used is decided at runtime depending upon the situation (i.e.,
  - data type of the object)
- This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.
- In Java means using the same message (or method name) with different classes
  - different objects can respond in their own way to the same message

What is Object-Orientation

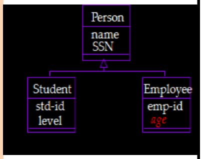
- Interfaces

- Information hiding - all data should be hidden within a class,
- make all data attributes private
- provide public methods to get and set the data values
  - e.g. Grade information is usually confidential, hence it should be kept private to the student. Access to the grade information should be done through *interfaces*, such as setGrade and getGrade

What is Object-Orientation

- Inheritance and Polymorphism

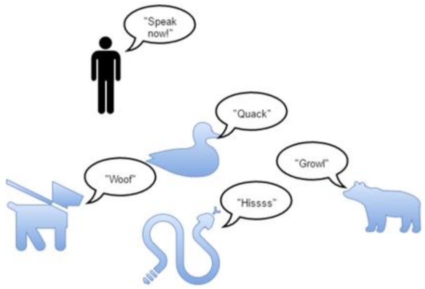
- **Inheritance:** Automatic duplication of superclass attribute and behavior definitions in subclass.
- **Specialization:** The act of defining one class as a refinement of another.
- **Subclass:** A class defined in terms of a specialization of a superclass using inheritance.
- **Superclass:** A class serving as a base for inheritance in a class hierarchy



Polymorphism:
Objects of different classes respond to the same message differently.
ability to dynamically choose the method for an operation at run-time or service-time

Polymorphism

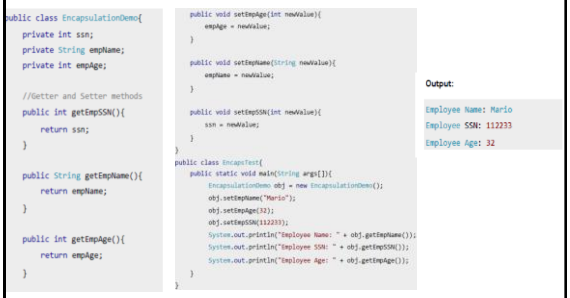
- Objects of different classes respond to the same message differently.
- ability to dynamically choose the method for an operation at run-time or service-time



Method overloading v/s overriding

- **Overloading** in simple words means two methods having same method name but takes different input parameters(number/type or both). This called static because, which method to be invoked will be decided at the time of compilation
- **Overriding** means a derived class is implementing a same method name of its super class. This is runtime because, which method to be invoked will be decided at the run time based on object type.

Encapsulation in Java



```
public class EncapsulationDemo{
    private int ssn;
    private String empName;
    private int empAge;

    //Getter and Setter methods
    public int getEmpSSN(){
        return ssn;
    }

    public String getEmpName(){
        return empName;
    }

    public int getEmpAge(){
        return empAge;
    }
```

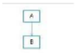
```
    public void setEmpAge(int newValue){
        empAge = newValue;
    }

    public void setEmpName(String newValue){
        empName = newValue;
    }

    public void setEmpSSN(int newValue){
        ssn = newValue;
    }
}
public class EncapsTest{
    public static void main(String args[]){
        EncapsulationDemo obj = new EncapsulationDemo();
        obj.setEmpName("Mario");
        obj.setEmpAge(32);
        obj.setEmpSSN(112233);
        System.out.println("Employee Name: " + obj.getEmpName());
        System.out.println("Employee SSN: " + obj.getEmpSSN());
        System.out.println("Employee Age: " + obj.getEmpAge());
    }
}
```

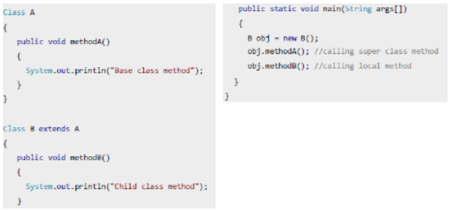
Output:

```
Employee Name: Mario
Employee SSN: 112233
Employee Age: 32
```

Inheritance in Java

- **Single Inheritance:** When a class extends another one class only then we call it a single inheritance.



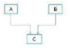


```
Class A
{
   public void methodA()
   {
     System.out.println("Base class method");
   }
}

Class B extends A
{
   public void methodB()
   {
     System.out.println("Child class method");
   }
```

```
   public static void main(String args[])
   {
     B obj = new B();
     obj.methodA(); //calling super class method
     obj.methodB(); //calling local method
   }
}
```

Inheritance in Java

- **Multiple Inheritance:** It refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent.



- **Small Talk, Java, C# do not support Multiple inheritance**. Multiple Inheritance is supported in C++.
- *"JAVA omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit. This primarily consists of operator overloading (although it does have method overloading), **multiple inheritance**, and extensive automatic coercions."*

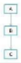
Inheritance in Java

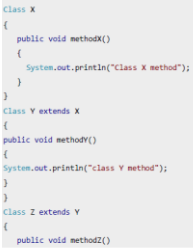
Why not multiple inheritance?

- We have two classes B and C inheriting from A. Assume that B and C are overriding an inherited method and they provide their own implementation. Now D inherits from both B and C doing multiple inheritance. D should inherit that overridden method, which overridden method will be used? Will it be from B or C? Here we have an ambiguity.

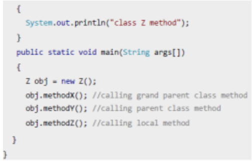
Inheritance in Java

- **Multilevel inheritance:**





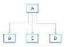
```
Class X
{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
Class Z extends Y
{
   public void methodZ()
```

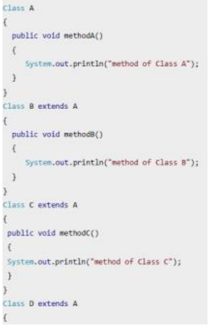


```
   {
     System.out.println("class Z method");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

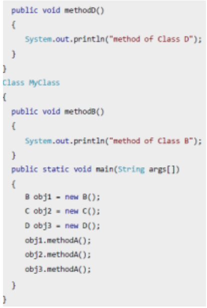
Inheritance in Java

- **Hierarchal Inheritance:** Many classes inherits from one class.





```
Class A
{
  public void methodA()
  {
     System.out.println("method of Class A");
  }
}
Class B extends A
{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
}
Class C extends A
{
 public void methodC()
 {
 System.out.println("method of Class C");
 }
}
Class D extends A
{
```



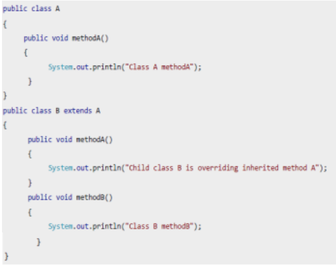
```
  public void methodD()
  {
     System.out.println("method of Class D");
  }
}
Class MyClass
{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

output

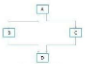
```
method of Class A
method of Class A
method of Class A
```

Inheritance in Java

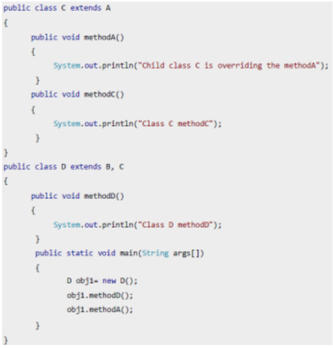
- **Hybrid Inheritance:** Combination of single and multiple.



```
public class A
{
     public void methodA()
     {
          System.out.println("Class A methodA");
      }
}
public class B extends A
{
      public void methodA()
      {
          System.out.println("Child class B is overriding inherited method A");
      }
      public void methodB()
      {
          System.out.println("Class B methodB");
        }
}
```



Inheritance in Java

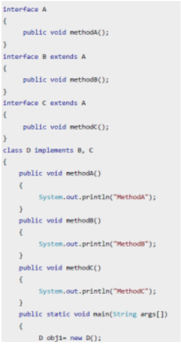


```
public class C extends A
{
       public void methodA()
       {
            System.out.println("Child class C is overriding the methodA");
        }
       public void methodC()
       {
            System.out.println("Class C methodC");
        }
}
public class D extends B, C
{
       public void methodD()
       {
            System.out.println("Class D methodD");
        }
        public static void main(String args[])
        {
               D obj1= new D();
               obj1.methodD();
               obj1.methodA();
        }
}
```

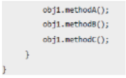
Output:

Error!!

Solution to hybrid inheritance

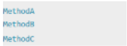


```
interface A
{
     public void methodA();
}
interface B extends A
{
     public void methodB();
}
interface C extends A
{
     public void methodC();
}
class D implements B, C
{
    public void methodA()
    {
         System.out.println("MethodA");
    }
    public void methodB()
    {
         System.out.println("MethodB");
    }
    public void methodC()
    {
         System.out.println("MethodC");
    }
    public static void main(String args[])
    {
         D obj1= new D();
```



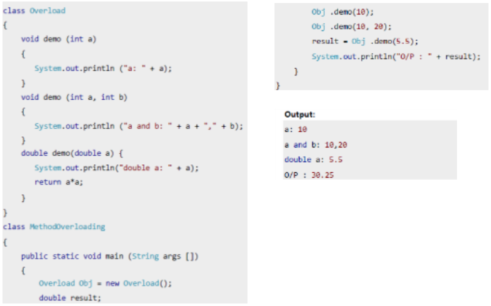
```
              obj1.methodA();
              obj1.methodB();
              obj1.methodC();
         }
}
```

Output:



```
MethodA
MethodB
MethodC
```

Method overloading



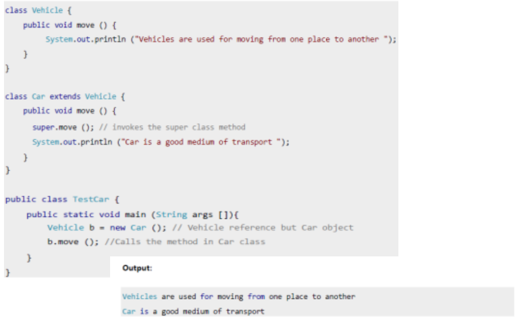
```
class Overload
{
    void demo (int a)
    {
       System.out.println ("a: " + a);
    }
    void demo (int a, int b)
    {
       System.out.println ("a and b: " + a + "," + b);
    }
    double demo(double a) {
       System.out.println("double a: " + a);
       return a*a;
    }
}
class MethodOverloading
{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
```

```
          Obj .demo(10);
          Obj .demo(10, 20);
          result = Obj .demo(5.5);
          System.out.println("O/P : " + result);
    }
}
```

Output:

```
a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
```

Method overriding in Java



```
class Vehicle {
    public void move () {
         System.out.println ("Vehicles are used for moving from one place to another ");
    }
}

class Car extends Vehicle {
    public void move () {
      super.move (); // invokes the super class method
      System.out.println ("Car is a good medium of transport ");
    }
}

public class TestCar {
     public static void main (String args []){
         Vehicle b = new Car (); // Vehicle reference but Car object
         b.move (); //Calls the method in Car class
     }
}
```

Output:

```
Vehicles are used for moving from one place to another
Car is a good medium of transport
```

END OF TOPIC 1

-COMING UP!!!!!!

-SDLC

-Software Development Approaches

-SAD v/s OOAD