10 September 2024 19:43

Object Oriented Programming Structure / System [OOPS] - Alan Kay

- It is not a syntax.
- It is a process / methodology which is used to solve real world problems using class and object
••
- 3 phases of OO s/w development
    1. Object Oriented Analysis (OOA) - Pen / paperor UMLDiagrams
    2. Object Oriented Design (OOD) - UML and Design Patterns
    3. Object Oriented Programming (OOP) - Syntax of OO programming language.

- Object Oriented Analysis and Design with Application: Graddy Booch [BOOK NAME]
- According to Graddy Booch there are 4 major parts/pillars of oops:
    1. Abstraction: To achieve simplicity
    2. Encapsulation: To hide the implementation & provide data security.
    3. Modularity: To reduce module dependency.
    4. Hierarchy: To achieve reusability.

- According to Graddy Booch there are 3 minor parts/pillars of oops:

    1. Typing / Polymorphism: To reduce maintenance of the system.
    2. Concurrency: To utilize h/w resources efficiently.
    3. Persistance: To maintain state of the instance in file/HDD

What is Abstraction?
-> Abstraction is major pillar of OOPS
-> process of getting essential things from a system
-> purpose >> achieve simplicity(think only about essentials)

•Abstraction and Encapsulation are Complementary Concepts

•Abstraction is the process of getting essential things from a System.
-> It focuses on observable behaviour.

•Encapsulation focuses on implementation that gives rise to this behaviour
->The process of making fields private is Data Encapsulation.

•Why getter setter are used?
=> to access class in web apps(that's y preferred),to access data in console (Scanner can be used).

•Hierarchy > 4 types

1. Has-a / whole-part (Association) >> has 2 forms
• Aggregation represents loose coupling(weak relationship)
•Composition

2. Is-a / Kind-of (Inheritance) represents tight coupling (strong relationship)

3. Use-a (Dependency)

4. Creates-a (Instantiation):

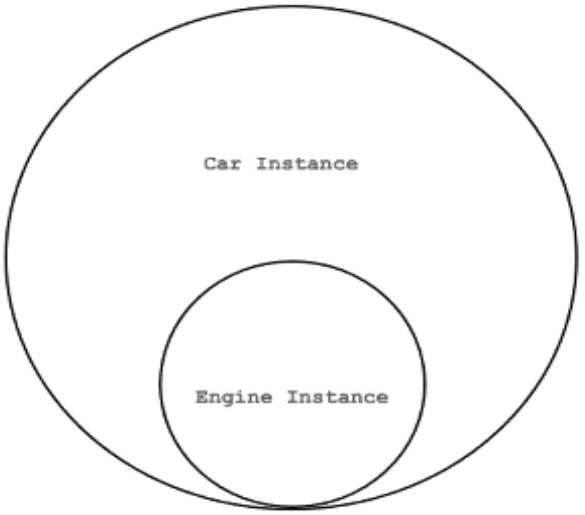
•Typing (polymorphism):
>> The ability to take any form is called polymorphism.
>> Compile Time -- Method Overloading (between classes)
>> Run Time -- Method Overriding (between super class and base class)

•Concurrency(achieved by threads / multithreading)
>> utilise hardware resources efficiently

•Persistence:
>>The maintain state of instance in file/HDD.

10 September 2024 19:43

•Hierarchy(Simple topic) >> understand the question and ask question to it
->For ex:
1) Car 'has -a' engine
>> ◘ Car has - a Engine
>> ◘ Engine is part of car

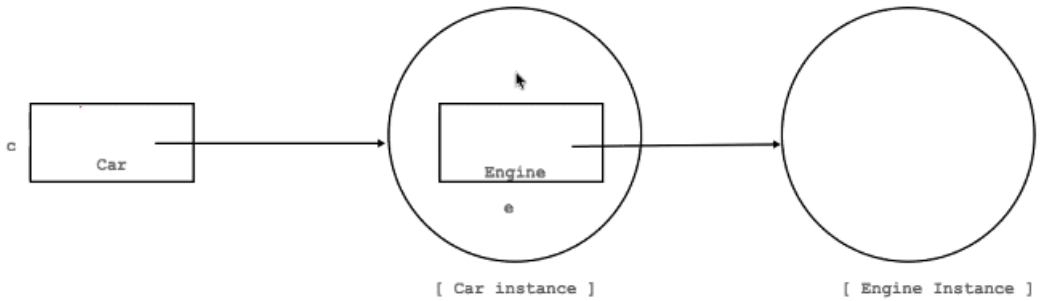



```
class Engine{

class Car{
Engine e = new Engine ( ); //Association

Car c = new Car( ) ;
```

2)Room 'has-a' wall
3)Saving account 'is a' account

○ In case of hierarchy, a object is a part of class,it is Association

Detailed Example: (Loose Coupling wrt to Inheritance)




```
class Engine{
}
class Car{
private Engine e;
public Car( ) {
   this.e = new Engine( ); //Association
    }
```

```
 }
class Program{
public static void main ( String[] args ) {
            Car c = new Car () ;
        }
}
```

♣ Association has 2 forms
1.Aggregation (Loose coupling)
2.Composition (Tight Coupling)

Examples:

1. Department has a Faculty

- Dependent - Department

- Dependency: Faculty

```
class Faculty{
}
class Department{
private Faculty faculty;
public void setFaculty ( Faculty faculty ) {
this. faculty = faculty;
}
}

class Program{
 public static void main ( String[] args ) {
   Department d =new Department ( );
   d.setFaculty ( new Faculty ( ) );
 }
}
```

2.Car has a Engine ()

Dependent - Car (Instance)
Dependency - Engine (Instance)

```
Class Engine {

}

Class Car{
 private Engine e = new Engine ();  //Association
}

Class Program{
```

```
  public static void main(String args[])
 Car c = new Car();
}
```

3.Employee has a join date

```
class Date{
private int day;
private int month;
private int year;
//ctor:
//getter & setter
//toString
}
class Employee{
private String name;
private int empid;
private float salary;
private Date joinDate = new Date () ; //Association -> Composition

   //ctor:
 / /getter & setter
 //toString
}
 class EmployeeUtil{
}
class Program{
public static void main ( String[] args ) {

}

}
```