Topic:

USAGE OF Aggregation, COMPOSTION**,** REFLECTION and Dependency IN OOP.

SURESH KUMAR

# MS (SOFTWARE ENGINEERING)

# MS- 3-15-60247

Aggregation By Reference:

Example Of Aggregation by reference

If Class StudentClass contains a reference to Class Address and Class Address contains a reference to Class StudentClass then no clear ownership can be determined and the relationship is simply one of association.

**Aggregation by reference Example**

class Address{

int street;

String city;

String state;

String country;

Address(int str1, String city1, String state1,String country1){

this.street = str1;

this.city =city1;

this.state= state1;

this.country = country1;

}

}

class StudentClass{

int rollNum;

String studentName;

Address studentAddr;

StudentClass(int rollNum1, String studentName1, Address addr){

this.rollNum = rollNum1;

this.studentName = studentName1;

this.studentAddr = addr;

}

public static void main(String args[]){

Address ad = new Address(55, "KARACHI", "SINDH", "PAKISTAN");

StudentClass obj= new StudentClass(123, "SURESH KUMAR", ad);

System.out.println(obj.rollNum);

System.out.println(obj.studentName);

System.out.println(obj.studentName);

System.out.println(obj.studentAddr.street);

System.out.println(obj.studentAddr.country);

}

}

Association:

Association establishes relationship between two **classes** through their **objects**. The relationship can be one to one, One to many, many to one and many to many.

**Association Example**

class CarClass{

String carName;

double carSpeed;

int carId;

CarClass(String name, double speed, int Id)

{

this.carName=name;

this.carSpeed=speed;

this.carId=Id;

}

}

class Driver{

String driverName;

int driverAge;

Driver(String name, int age){

this.driverName=name;

this.driverAge=age;

}

}

class TransportCompany{

public static void main(String args[])

{

CarClass obj= new CarClass("Ford", 180.15, 9988);

Driver obj2 = new Driver("Andy", 45);

System.out.println(obj2.driverName+" is a driver of car Id: "+obj.carId);

}

}

Output:

Andy is a driver of car Id: 9988

In the above example, there is a one to one relationship (**Association**) between two classes: Car and Driver. Both the classes represent two separate entities.

Composition:

**Composition** is the design technique to implement **has-a** relationship in classes.

Composition takes the relationship one step further by ensuring that the containing object is responsible for the lifetime of the object it holds. If Object B is contained within Object A, then Object A is responsible for the creation and destruction of Object B.

**Composition Example**

public class Car {

//final will make sure engine is initialized

private final Engine engine;

public Car(){

engine = new Engine();

}

public static void main(String args[]){

System.out.println("Engine create");

}

}

class Engine {

private String type;

}

Reflection

Process of examining or modifying the run time behavior of a class at run time.

**Reflection By Class Example**

class Simple{}

class Test{

void printName(Object obj){

Class c=obj.getClass();

System.out.println(c.getName());

}

public static void main(String args[]){

Simple s=new Simple();

Test t=new Test();

t.printName(s);

}

}

Dependency

also called a using relationship, which means, one class is dependent on another class. Whenever a class A uses another class or interface B, then A depends on B. A cannot carry out it's work without B, and A cannot be reused without also reusing B. In such a situation the class A is called the "dependant" and the class or interface B is called the "dependency". A dependant depends on its dependencies.

**Dependency By Example**

package com.example.e4.rcp.todo.parts;

import java.util.logging.Logger;

public class MyClass {

private static Logger logger;

public MyClass(Logger logger) {

this.logger = logger;

// write an info log message

logger.info("This is a log message.");

}

}