Object Oriented Report

Programming

Joel

(103366239)

Table of Contents

Contents

Introduction

Object Oriented Programming
Of the many styles of programming, Object-Oriented programming (OOP) is amongst the most popular, practical, and widely used. Object-Oriented programming, as the name suggests, is aligned around representing code as objects and data instead of procedural instructions, where code is like a spaghetti, which means they are intertwined together. If one of the code breaks d, the entire codebase breaks down.
To differentiate procedural programming from object
oriented programming, OOP treats software as being composed of objects in classes instead of procedural instructions within the main/driver function.
This style of programming helps software be scalable and secure. OOP centers around using objects in code, but also to rules and guidelines to achieve cohesion or decoupling.

OOP Characteristics

1. Software is represented as "objects" which generally consists of methods (another name for functions) and data
2. Data can be encapsulated/hidden to be visible to itself or other objects. In addition, it can be set to public also
3. Objects can interact with other objects

OOP Concepts

Abstraction

- Control complexity by hiding unnecessary details from other classes
- Enables the programmer to create complex objects using the current abstractions
- Playing a movie is a good analogy of abstraction, the user just clicks the play button without worrying about what happens behind the scenes like encoding, decoding, graphic processing, etc

Encapsulation

- Keeps objects and methods in a class
- Hides internal objects from the outside world
- Getters and setters are parts of encapsulation because it makes private values private and allows access only through methods.
- Has access modifiers
    - Private -
        - Only used within the class
    - Protected,
        - Similar to private, can be accessed and overridden by child classes.
    - Public.
        - Accessed by current class and outside classes

Polymorphism

- Means "Many forms", it shows objects can be treated as another type.
- Allows method overloading, same identifier but different data type. Example int number() , double number().

Inheritance

- It is a way classes attains their properties, objects , and methods, from a parent/super class.
- Enables reusability of fields and methods from parent class and allow expansions
- Incorporates the idea of generalization and specialization
- Allows classes to be treated as parent classes (student as human, tutor as human) when a general object is needed

Roles

- Defines the purpose *"role"* that needs to be fulfilled within the program.
- Roles refer to the object and its contribution towards the program's aim
- Roles are useful for identification as they provide a viewpoint into the generalization and specialization of any role by identifying common traits

Responsibilities

- Refers to obligation of the object/class to know or do something
- Each class should have only one responsibility, sometimes known as, (responsibility-driven design)
- Responsibilities means the object, field, method must know something to fulfill its responsibilities. It must know whether it's a property, method, variable, etc

Collaboration

- Refers to how objects communicate with each another
- Separated into 3 categories:
    - Dependency
        - Defined as relationship between objects in which one object depends on the other
    - Association
        - A perpetual relationship: the object knows its association with another object, can be thought of as, "has-a" relationship.
        - 3 types of association:
            - **Association**
                - Perpetual relationship between objects
            - **Aggregation**
                - A whole object is made up of various parts, but the parts can be shared with other objects
            - **Composite Aggregation**
                - A specialized form of association in which, if the parent object is destroyed, the child object will be destroyed

- Inheritance
    - Can be referred as "is-a kind of" relationship: rock and pop are music genres but humans are not music genres

Coupling

- Refers to the degree in which an object depends on another. Less coupling means object is more independent
- Types of coupling
    - **Loose** – Objects are more independent
    - **Tight** – Objects are more dependent on each other

Cohesion

- Refers to how similar the members(methods, class, fields) of a module is to other members in the same module. More cohesion means more specific to that module
- Types of cohesion
    - **High** – Class/objects have well defined scopes. Easier to maintain, understand, and reusable
    - **Low**- Class/objects donot have well defined scopes. Harder to maintain, understand, and reuse.

Programming Artefacts

Object

- Refers to components which consists of properties and methods
- Design to mirror real world objects

Fields

- Refers to characteristics/ properties
- Initialized and are components of classes, can be public, private, or protected. Properties (accessors) are used to read/write

Method

- Refers to functions and instructions within an object to execute a task
- Allows change of fields within the object
- Allows interaction with other objects

Interface

- Acts as descriptions of what an object can or cannot perform
- Used to enforce certain properties in a class
- Are abstract in nature and defines methods which a class must implement

Class

- Acts as a template for objects
- Provides initial values of states and behavior implementation
- Scalable in nature
- An approach of structuring data so it can be reusable also

Relations

This part shows the relation of OOP concepts discussed above to programs done in this subject

In the Shape class part of the drawing program

```
public class Shape
    {
        private Color _color;
        private float _x, _y;
        private int _width, _height;
        private bool _selected;

public Shape()
        {
            _color = SplashKit.ColorWhite();
            _x = 0;
            _y = 0;
            _width = 100;
            _height = 100;

        }
```

Using OOP artefacts to relate to this class, It can be noted that these class is public, its fields are all private. In terms of objects, the Shape () is one the initialized constructor in this class and is an object of this class.

From an OOP concepts perspective, we can see that inheritance is applied as MyCircle class inherits from the Shape class. From the MyCircle constructor, the **base** keyword is used to access members of the base class Shape within this child class MyCircle. Encapsulation is applied as **_radius** is set as private which can only be interacted within this class.

```
public class MyCircle : Shape
    {
        private int _radius;

        public MyCircle(Color color, int radius) : base(color)
        {
            _radius = radius;
        }
```

Within the context of this classes, we can see that inheritance, encapsulation, access modifiers, objects member are used which demonstrates the use of object oriented principles for the Drawing program.

References:

www.javatpoint.com. 2021. *Access modifiers in java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/access-modifiers> [Accessed 18 May 2021].

Docs.microsoft.com. 2021. *Fields - C# Programming Guide*. [online] Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/fields> [Accessed 18 May 2021].

Citeseerx.ist.psu.edu. 2021. [online] Available at: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.1126&rep=rep1&type=pdf> [Accessed 18 May 2021].

O'Reilly Online Learning. 2021. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition*. [online] Available at: <https://www.oreilly.com/library/view/applying-uml-and/0130925691/0130925691_ch16lev1sec2.html> [Accessed 18 May 2021].

Kanjilal, J., 2021. *Association, aggregation, and composition in OOP explained*. [online] InfoWorld. Available at: <https://www.infoworld.com/article/3029325/exploring-association-aggregation-and-composition-in-oop.html#:~:text=Remember%20that%20aggregation%20and%20composition,may%20have%20only%20one%20owner.> [Accessed 18 May 2021].

GeeksforGeeks. 2021. *Coupling in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/coupling-in-java/> [Accessed 18 May 2021].

GeeksforGeeks. 2021. *Cohesion in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/cohesion-in-java/> [Accessed 18 May 2021].

Cs.utah.edu. 2021. *OOP - Interfaces*. [online] Available at: <https://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html> [Accessed 18 May 2021].

Brilliant.org. 2021. *Classes (OOP) | Brilliant Math & Science Wiki*. [online] Available at: <https://brilliant.org/wiki/classes-oop/> [Accessed 18 May 2021].