Object-Oriented Programming, the possible advantages and disadvantages of using various OOP features in the design and implementation of software.

By- Mohammad Huzaifa Shaikh

Overview-

This aim of this article is to highlight, in-depth, the possible benefits and drawbacks of using OOP and its features in designing and implementing software. This article is also designed to be used as a report by companies to make decisions on whether to adopt and use various object-oriented features in the design and implementation of that company's software.

For the ease of the readers understanding, while elaborating key points in this article, high-level (surface level) information is highlighted first followed by "***moreover***," after which the information goes into more detail and more towards the lower-level (in-depth) aspects of the point of discussion.

---

table of contents

1- Brief description of OOP in creation of software

A programming paradigm known as object-oriented programming (OOP) is based on the ideas of classes and objects. It is used to organise software into straightforward, reusable classes of code blueprints, which are then utilised to build distinct instances of things also known as objects.

2- Advantages of OOP

Some of the biggest advantages of OOP are actually the features/concepts of itself. OOP has four fundamental features: - abstraction, encapsulation, inheritance and polymorphism. These features are what combine to make OOP such an appealing programming paradigm and a no-brainer for when it comes to developers to choose their method/strategy for creating major software. Let's see what makes these certain features of OOP into considerable advantages. [M]

2.1 Abstraction.

Many would agree that it is far more straightforward to reason and design a program when you can separate the interface of a class from its implementation and focus on the interface. This is like treating a system as a "Black Box" where it is not important to understand the gory inner workings in order to reap the benefits of using it. This Process is called abstraction in OOP as we are abstracting away the implementation details of a class and only presenting a clean, easy-to-use interface via the class's functions/methods [1].

Example: A real world example-

Imagine you have to go to your friend's birthday party. You order an UBER and once the app finalizes a driver nearby for you, it gives you the details of that driver and his/her car such as the drivers name, the plate number, the make of the car and the color of the car. These are the details you will need for things such as, but not limited to, identifying the car, making sure you are getting the right car designated to you by Uber or sending a screenshot to one of your family members in case of an emergency. However, the rest of the data about the driver and their car is pretty much irrelevant to you as a passenger. Like the Social Insurance number of the driver, the birth date of the driver, the mileage on the car or the drivers' mothers name, such information is hidden from you as it is pretty much irrelevant to you but can still be critical information be used elsewhere or later in time for other purposes. [M]

Moreover, Abstract classes like the one in the example above can also have concrete methods as well as, obviously, abstract methods. Unlike a normal class which cannot have an abstract method, all methods must be predefined [M].

Abstract methods are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations. It also extends the same Abstract class and offers different implementations of the abstract methods. Abstract classes help to describe generic types of

behaviors and object-oriented programming class hierarchy. It also describes subclasses to offer implementation details of the abstract class. [2]

Example:

Let's say we are working on the backend of a banking software that compares different banks. We would create an abstract Bank class that would store the data/methods that would then be used by all instances of the class (different banks) [M].

```
4   //Written by Mohammad Huzaifa Shaikh
5   abstract class Bank {
6       abstract void getName();
7       abstract void getAddress();
8       abstract void getTransit();
9       abstract void getInterest();
10
11  }
12
13
```

[M]

```
13
14  class TD extends Bank {
15      private int transit = 67390471;
16      private int rate = 5;
17
18      void getName(){System.out.println("TD Canada Trust");}
19      void getAddress(){System.out.println("3252 Imaginary Street, Toronto, Ontario");}
20      void getTransit(){System.out.println("Transit Number = " + this.transit);}
21      void getInterest(){System.out.println("Interest Rate = " + this.rate + "%");}
22  }
23
24  class BMO extends Bank {
25      private int transit = 7396381;
26      private int rate = 4;
27
28      void getName(){System.out.println("Bank of Montreal");}
29      void getAddress(){System.out.println("1234 Random Street, Toronto, Ontario");}
30      void getTransit(){System.out.println("Transit Number = " + this.transit);}
31      void getInterest(){System.out.println("Interest Rate = " + this.rate + "%");}
32  }
33
34  class MHS extends Bank {
35      private int transit = 6942069;
36      private int rate = 9;
37
38      void getName(){System.out.println("Bank of Huzaifa");}
39      void getAddress(){System.out.println("6969 Huzaifa Street, Toronto, Ontario");}
40      void getTransit(){System.out.println("Transit Number = " + this.transit);}
41      void getInterest(){System.out.println("Interest Rate = " + this.rate + "%");}
42  }
43
```

[M]

```
45
46
47  class Main {
48      public static void main(String[] args) {
49          System.out.println ("\n\n");
50
51          Bank A = new TD();
52          A.getName();
53          A.getAddress();
54          A.getTransit();
55          A.getInterest();
56
57          System.out.println ("\n\n");
58
59          Bank B = new BMO();
60          B.getName();
61          B.getAddress();
62          B.getTransit();
63          B.getInterest();
64
65          System.out.println ("\n\n");
66
67          Bank C = new MHS();
68          C.getName();
69          C.getAddress();
70          C.getTransit();
71          C.getInterest();
72
73      }
74  }
```
[M]

Above, we can see that when a regular class extends the abstract bank class, it can now use all the methods that were initially declared in the abstract class and override them to better fit the specific class. Here we create a default abstract Bank class that contains all the methods and then use they keyword "extends" after we declare the class which enables each separate bank class to override the abstract methods and add its own data. [M]

2.2 Encapsulation.

Encapsulation describes the idea of bundling data and methods that work on that data within one unit, like a class in Java. This concept is also often used to hide the internal representation, or state of an object from the outside. This is called information hiding. [3]

The general idea of this mechanism is simple. For example, you have an attribute that is not visible from the outside of an object. You bundle it with methods that provide read or write access. Encapsulation allows you to hide specific information and control access to the internal state of the object [3].

Many programming languages use encapsulation frequently in the form of classes. A class is a program-code-template that allows developers to create an object that has both variables (data) and behaviors (functions or methods). A class is an example of encapsulation in computer science in that it consists of data and methods that have been bundled into a single unit. [4]

Encapsulation is extremely advantageous for when it comes to things like hiding data, reusability and flexibility. It gives you the ability to decide whether the variables of an object will be read/write or read only/write only. [M]

2.2.1 Hiding data

Better known as data hiding/information hiding is a technique of hiding internal object details. Data hiding ensures that only members of the class have access to the data. The data integrity is preserved. While data hiding limits the use of the data to ensure data security, encapsulation wraps up complex data to give the user a simplified perspective.

References.

[M] This is my own personal work. 100% my own ideas and thinking about the topic and examples labeled with 'M' are also created by me.

[1] Kyle Herrity: What Is Object-Oriented Programming? 4 Basic Concepts of OOP. available at https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming.

[2] James Hartman: What is Abstraction in OOPs? Java Abstract Class & Method. available at https://www.guru99.com/java-data-abstraction.html

[3] Thorben Janssen. OOP Concept for Beginners: What is Encapsulation. available at:- https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/

[4] Encapsulation - Definition & Overview. available at https://www.sumologic.com/glossary/encapsulation/#:~:text=What%20does%20encapsulation%20mean%3A%20In,in%20the%20form%20of%20classes.