Task 1:

Disadvantages Of OOP

- **Steep expectation to learn and adapt:** The perspective engaged with object-situated programming may not be normal for certain individuals, and it can invest in some opportunity to become accustomed to it. It is complex to make programs in view of the cooperation of articles. A portion of the key programming procedures, like inheritance and polymorphism, can be tested to appreciate at first.
- **Bigger program size:** Object-arranged programs commonly include more lines of code than procedural projects.
- **More slow projects:** Object-arranged programs are normally slower than procedure-based programs, as they ordinarily require more guidelines to be executed.
- **Not appropriate for a wide range of issues:** There are issues that loan themselves well to useful programming style, rationale programming style, or strategy based programming style, and applying object-arranged programming in those circumstances will not bring about effective projects.

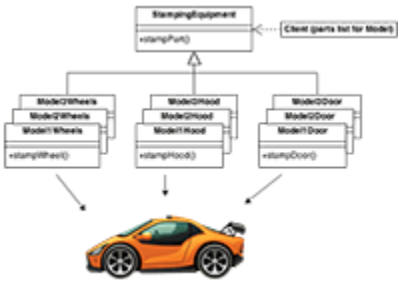
Task 2:

Types of Design Patterns?

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

1- Creational design patterns

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

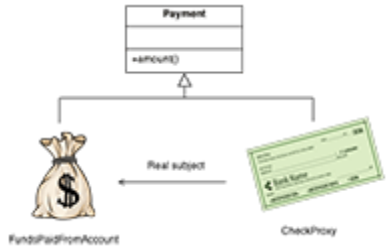


- **Abstract Factory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation
- **Factory Method**
  Creates an instance of several derived classes
- **Object Pool**
  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**
  A fully initialized instance to be copied or cloned
- **Singleton**
  A class of which only a single instance can exist

2- Structural design patterns

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

- **Adapter**
  Match interfaces of different classes
- **Bridge**
  Separates an object's interface from its implementation
- **Composite**
  A tree structure of simple and composite objects
- **Decorator**
  Add responsibilities to objects dynamically
- **Facade**
  A single class that represents an entire subsystem
- **Flyweight**
  A fine-grained instance used for efficient sharing

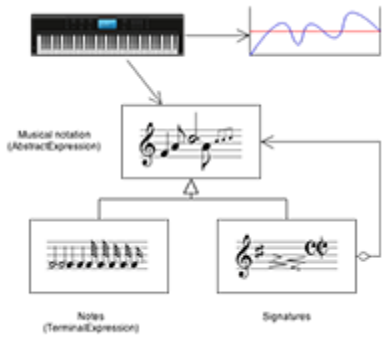


- **Private Class Data**
  Restricts accessor/mutator access

- **Proxy**
  An object representing another object

3- Behavioral design patterns

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.



- **Chain of responsibility**
  A way of passing a request between a chain of objects
- **Command**
  Encapsulate a command request as an object
- **Interpreter**
  A way to include language elements in a program
- **Iterator**
  Sequentially access the elements of a collection
- **Mediator**
  Defines simplified communication between classes
- **Memento**
  Capture and restore an object's internal state
- **Null Object**
  Designed to act as a default value of an object
- **Observer**
  A way of notifying change to a number of classes

- 

  State
  Alter an object's behavior when its state changes

- **Strategy**
  Encapsulates an algorithm inside a class
- **Template method**
  Defer the exact steps of an algorithm to a subclass
- **Visitor**
  Defines a new operation to a class without change

ببساطة الـ Design Patterns هي ناتج التطور التطبيعي لتاريخ البرمجيات , ازاي ؟ نجيب الموضوع من الاول , مع بداية ظهور البرمجيات كانت بتتكتب بطريقة Sequential بمعني إنك بتكتب البرنامج بتاعك كله عبارة عن مجموعة سطور من الـ Code تحت بعضها بتتنفذ بالترتيب و دي كان اسمها الـ Sequential Programming.

بعد فترة ومع وزيادة احجام البرامج بدأوا يلاقوا ان الكود بيتكرر كتير فده وصلهم انهم ممكن يجمعوا الأكواد اللي بتتكرر دي في مكان واحد وكل ما نحتاجهم نستخدمهم وهنا ظهر مفهوم الـ function و دي كان اسمها الـ Procedural programming.

ومع ازدهار عصر البرمجيات بطريقة كبيرة وسريعة بدأ يتجه التفكير الي إن ازاي نسهل البرمجة علي المبرمجين بأنها تكون اقرب لطريقة تفكير الإنسان و فالوقت ده ظهر الـ OOP - object oriented programming وهو انك بتقسم السوفت وير لمجموعة classes و بتحدد علاقاتها ببعضها و الـ behavior بتاعها ودي اقرب حاجه للحياة الطبيعية و طريقة تفكير الانسان قدروا يوصلولها لحد دلوقتي.

لحد هنا كويس , طبيعي وانت شغال كـ software engineer بتواجهك مشاكل وبتفكرلها في حلول وبتحلها , ومع الوقت في مجموعة مشاكل بقت بتتكرر مع ناس كتير وبقت مشاكل مشتركة , وبدأ المبرمجين يعملولها طرق حلول (patterns) ويطوروا طرق الحلول دي وتبقي طرق موحدة بتسهل عليك حل مشاكل معينة بعينها لما تقابلك (وهتقابلك) وهي دي الـ design patterns.

فهي ببساطة مجموعة حلول لمشاكل common هتقابلك بطريقة متكررة وانت شغال. بس لازم يكون واضح انها مش أكواد او حاجه تتحول لكود هي مجموعة خطوات لحل مشكلة معينة , انت بترجمها لكود بعد كده.

- [+]ايه الفايدة اللي هتعود عليا لما استخدم الـ design patterns ؟

لما تستخدم الـ design patterns هتعمل اغلب الـ best practices اللي تخليك professional software engineerزي:

- هتلاقي نفسك بتكتب كود اقل بكتير.Less code
- هتلاقي ان الـ software بتاعك سهل تطور وتعدل فيه بأقل مجهود.maintainable software
- هتلاقي ان قدراتك فالـ Problem Solving زادت بشكل ملحوظ جدا

Task 3:

What is API in Python?

In this Python API, we'll learn how to retrieve data for data science projects. There are millions of APIs online which provide access to data. Websites like Reddit, Twitter, and Facebook all offer certain data through their APIs.

To use an API, you make a request to a remote web server, and retrieve the data you need.

But why use an API instead of a static CSV dataset you can download from the web? APIs are useful in the following cases:

- **The data is changing quickly**. An example of this is stock price data. It doesn't really make sense to regenerate a dataset and download it every minute — this will take a lot of bandwidth, and be pretty slow.
- **You want a small piece of a much larger set of data**. Reddit comments are one example. What if you want to just pull your own comments on Reddit? It doesn't make much sense to download the entire Reddit database, then filter just your own comments.
- **There is repeated computation involved**. Spotify has an API that can tell you the genre of a piece of music. You could theoretically create your own classifier, and use it to compute music categories, but you'll never have as much data as Spotify does.

In cases like the ones above, an API is the right solution. In this blog post, we'll be querying a simple API to retrieve data about the International Space Station (ISS).

Task 4:

Difference between multicore and multithreading?

Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.

تنفيذ الدوال بشكل متوازي

Each user request for a program or system service is tracked as a thread with a separate identity. As programs work on behalf of the initial thread request and are interrupted by other requests, the work status of the initial request is tracked until the work is completed. In this context, a user can also be another program.

Fast central processing unit (CPU) speed and large memory capacities are needed for multithreading. The single processor executes pieces, or threads, of various programs so fast, it appears the computer is handling multiple requests simultaneously.

A multicore processor is an integrated circuit that has two or more processor