**Report on Question 1**

**Overview**

This report outlines the development process of a Tkinter application designed to allow users to open image files and classify them using object oriented programming (OOP) principles. The application demonstrates fundamental OOP concepts such as multiple inheritance, decorators, encapsulation, and method overriding, and has been designed with future flexibility in mind to potentially integrate real AI models. The project was a collaborative effort between **Phat**, **Gilbert**, **Umais**, and **Sayed**, where each member took responsibility for different aspects of the development. This report will provide details on how each concept was applied and how the project responsibilities were divided.

**Task Division**

**Phat**: Phat was responsible for constructing the main structure of the program. He ensured that the different components, including window management and file operations, were correctly connected and integrated into the final application. Phat also built the user interface using Tkinter, focusing on making it simple, interactive, and intuitive. His contribution laid the foundation for the app’s usability by making sure the window design and navigation were cohesive.

**Gilbert**: Gilbert’s primary responsibility was ensuring that the program followed good coding practices, specifically related to **encapsulation**. He made sure that parts of the code, such as the classification method, were hidden from direct access outside of the class (a concept called encapsulation). This prevents unintended interference with the internal workings of the program. He also contributed to the functionality of the image classification process, working closely with Sayed to ensure the classifier function was well-structured.

**Umais**: Umais added a feature called a **decorator**, which was used to log classification actions whenever an image was classified by the program. This functionality makes it easy to monitor when and which images were classified. This added an extra layer of transparency, showing how decorators can be useful in adding functionality to existing methods without altering their core logic.

**Sayed**: Sayed focused on making the program more flexible, ensuring that the code could be easily adapted to support multiple AI models in the future (a feature known as **polymorphism**). For now, the program uses a mock up classification system that randomly assigns an image to one of four categories: "Dog", "Cat", "Bird", or "Car". He also collaborated with Gilbert on the development of the basic classification method that simulates an AI-driven image recognition system.

**Code Structure and Explanation**

1. **Multiple Inheritance**:

In the development of this application, we employed **multiple inheritance** by creating an App class that inherits from both the BasicWindow and FileOperations classes. This allows the App class to combine the functionality of setting up the user interface (inherited from BasicWindow) and handling file-related operations, like selecting an image file (inherited from FileOperations). This approach makes the code modular and easier to maintain, as different features are handled by separate classes, but they are combined into a cohesive whole.

Phat led this part of the project, ensuring that the inheritance structure was properly implemented and that the App class could use both the window setup and file management features seamlessly.

A screen shot of a computer program

Description automatically generated

1. **Encapsulation**:

The principle of **encapsulation** was demonstrated by making certain methods private, specifically the \_simple\_image\_classifier method. This method, responsible for the image classification logic, was intentionally designed to be private (indicated by the leading underscore \_) to prevent it from being accessed or modified outside the class. Encapsulation ensures that the internal workings of this method are hidden from other parts of the program, reducing the risk of unintended changes or misuse of the classification logic.

Gilbert took the lead on this aspect, ensuring that the internal logic of the classifier remained protected and that only the relevant methods and variables were exposed to the rest of the application. This practice follows the OOP principle of data hiding, improving the security and integrity of the code.

A computer screen shot of text

Description automatically generated

1. **Method Overriding**:

**Method overriding** was used in the program to customize how certain inherited functions behave. In the App class, the classify\_image method overrides any behavior it might inherit from the parent classes, specifically adding functionality that updates the display with the classification result after an image is selected. This ensures that when the user interacts with the application, the correct functionality is triggered—displaying the classification result on the screen.

Phat and Gilbert worked together on this, ensuring that the classify\_image method was properly overridden to update the interface with the classification result once an image is opened and processed.

A screen shot of a computer code

Description automatically generated

1. **Decorators**:

A **decorator** is a feature that allows additional functionality to be added to existing functions. In our application, Umais implemented a decorator to add logging functionality to the classify\_image method. Every time an image is classified, this decorator logs the event, including the file path of the image being processed. This is useful for tracking user actions or debugging the program by keeping a record of which images have been classified.

Umais' contribution here made the program more transparent, as it logs essential details whenever the image classification process occurs, making it easier to trace any issues or analyze how the program is being used.

A screenshot of a computer screen

Description automatically generated

1. **Polymorphism**:

The principle of **polymorphism** allows the program to be flexible enough to handle different types of objects or processes in the future. In this application, although the classification process currently uses a simple random choice, Sayed ensured that the structure supports polymorphism. This means that in the future, different AI models could be plugged into the classification system, and the classify\_image method would still work, regardless of which model is used. This flexibility is a crucial feature, as it allows the program to evolve and integrate more sophisticated classification methods as needed.

Sayed's work on this part ensures that the app can easily be expanded to include actual AI models for tasks like object detection, image recognition, or language translation, without needing to restructure the entire program.

A screen shot of a computer code

Description automatically generated

**Conclusion**

This project demonstrates how fundamental object-oriented programming concepts can be applied to create a simple yet functional Tkinter application. The use of multiple inheritance, encapsulation, method overriding, decorators, and polymorphism allowed the team to design a flexible and modular program that can be easily expanded in the future.

Each team member played a vital role in ensuring the success of the project. Phat focused on connecting the main components and user interface, Gilbert ensured proper encapsulation and helped with the classification logic, Umais added the logging feature through decorators, and Sayed contributed to the program’s flexibility by designing it to support polymorphism.

The application works as expected, displaying selected images and classifying them, albeit using a random mock-up classification model. The future potential of this application lies in its abili

This screenshot shows the Tkinter-based AI Image Classifier application in action. The application allows users to upload an image file, which is then classified using a simple mock-up classification system. In this example, the user has uploaded an image of a dog, and the program has randomly classified it as a "Bird." The interface consists of an image display area, an "Open Image" button for selecting files, and a label that shows the classification result below the image.