CODE AND SUMMARY OF CLASS 8

Sahir Ahmed Sheikh

Saturday (2 – 5)

Teachers:

Muhammad Bilal And Ali Aftab Sheikh

Code And Summary Of Class 8 – Saturday (2 – 5) | Quarter 3

Introduction

This document 📄 summarizes Class 8 of Quarter 3, held on Saturday from 2 – 5 PM. The session was led by Sir Ali Aftab Sheikh, with Sir Muhammad Bilal. Due to low attendance and a two-week delay in the schedule, the class focused on revising **Object-Oriented Programming (OOP)** concepts covered in Class 7, with an emphasis on practical coding. The session used a biryani theme to make OOP concepts relatable, encouraging active participation and preparation for an upcoming test.
The class covered:

- **Revision of OOP Concepts**: Classes, objects, and the four pillars (Polymorphism, Inheritance, Abstraction, Encapsulation).
- **Practical Coding**: Hands-on examples using a Biryani class to demonstrate OOP principles.
- **Class Logistics**: Accessing resources on GitHub and Discord, assignment reminders, and test preparation.
- **Class Repo**: GIAIC-Q3-Saturday-2-To-5

Students were urged to join the class Discord server, review resources, and prepare for a test scheduled for the next Saturday, as there will be no regular class.

📘 Topics Covered

1. Class Logistics and Resources

- **Context**: The session began with a discussion on class logistics, addressing the two-week delay and low attendance. The teachers decided to revise OOP concepts instead of introducing new topics.
- **Resources**:

    - **GitHub Repository**: All class codes and assignments are available on the class GitHub repository. Students can access the repository to review past assignments (e.g., Secure Data Encryption from Class 6) and class code.
    - **Discord Server**: The Saturday 2 – 5 PM batch channel on Discord provides links to WhatsApp, slides, and other resources. Students who haven't joined were encouraged to do so using the provided link, even via the Discord website if mobile space is an issue.
    - **Class Form**: Session forms and assignment submissions are managed through the class form, with a reminder to complete all previous assignments and share them on LinkedIn.
- **Key Points**:
    - Ensure familiarity with GitHub and Discord for accessing class materials.
    - Complete and submit all pending assignments before the next class.
    - Prepare for the upcoming test, as the next Saturday session will be a test, not a regular class.

---

2. Revision of Object-Oriented Programming (OOP)

- **Concept**: OOP is a programming paradigm that organizes code into objects created from classes, promoting modularity and reusability. The class revisited the four pillars of OOP: Polymorphism, Inheritance, Abstraction, and Encapsulation.
- **Mnemonic**: The pillars were recalled using the acronym **PIAE** (Polymorphism, Inheritance, Abstraction, Encapsulation).
- **Key Terms**:
    - **Class**: A blueprint defining attributes (variables) and methods (functions).
    - **Object**: An instance of a class, created from the blueprint.
    - **Attributes**: Properties of a class (e.g., variables like rice, meat).
    - **Methods**: Functions defined in a class to perform actions (e.g., serve()).
- **Why OOP?**:

    - Simplifies complex problems by breaking them into reusable components.
    - Avoids unstructured “spaghetti code.”
    - Provides structure for maintainability and scalability.

2.1 Recap of the Four Pillars of OOP

The class reviewed the four pillars of OOP, using relatable analogies (e.g., biryani recipe) to reinforce understanding.

- **Polymorphism** (Different Flavors):
    - **Definition**: Allows objects of different classes to be treated as objects of a common parent class, with each object behaving uniquely via method overriding.
    - **Analogy**: Different flavors of biryani—same method name (e.g., cook()) but overridden to produce unique results (e.g., simple biryani vs. Hyderabadi biryani).
    - **Key Insight**: Enables flexibility by allowing a single interface for varied behaviors.
- **Inheritance** (Family Recipe):
    - **Definition**: A child class inherits attributes and methods from a parent class, promoting code reuse.
    - **Analogy**: A family recipe passed down—e.g., grandmother’s simple biryani (parent class) inherited by mother, who adds saffron and mint to make Hyderabadi biryani (child class).
    - **Key Insight**: Reduces code duplication and models real-world hierarchies.
- **Encapsulation** (Secret Masala):
    - **Definition**: Bundles data and methods into a class, restricting direct access using access modifiers (public, protected, private).
    - **Analogy**: Hiding the “secret masala” of biryani—only accessible within the class if private.
    - **Access Modifiers**:
        - **Public**: Accessible everywhere (e.g., account_holder).

        - **Protected**: Accessible within parent and child classes (e.g., _attribute).
        - **Private**: Accessible only within the class (e.g., __attribute).
    - **Key Insight**: Protects data integrity and controls access.
- **Abstraction** (Chef's Magic):
    - **Definition**: Hides complex implementation details, exposing only necessary functionality.
    - **Analogy**: Chef's magic—diners see the final biryani dish but not the cooking process.
    - **Key Insight**: Simplifies interaction by providing a clear interface while hiding complexity.
    - **Assignment**: Students were tasked with creating an abstraction example using the biryani theme and submitting it via the class form.

---

3. Practical Coding: Classes and Objects with Biryani Theme

The class focused on hands-on coding to solidify OOP concepts, using a Biryani class as a blueprint to create objects.

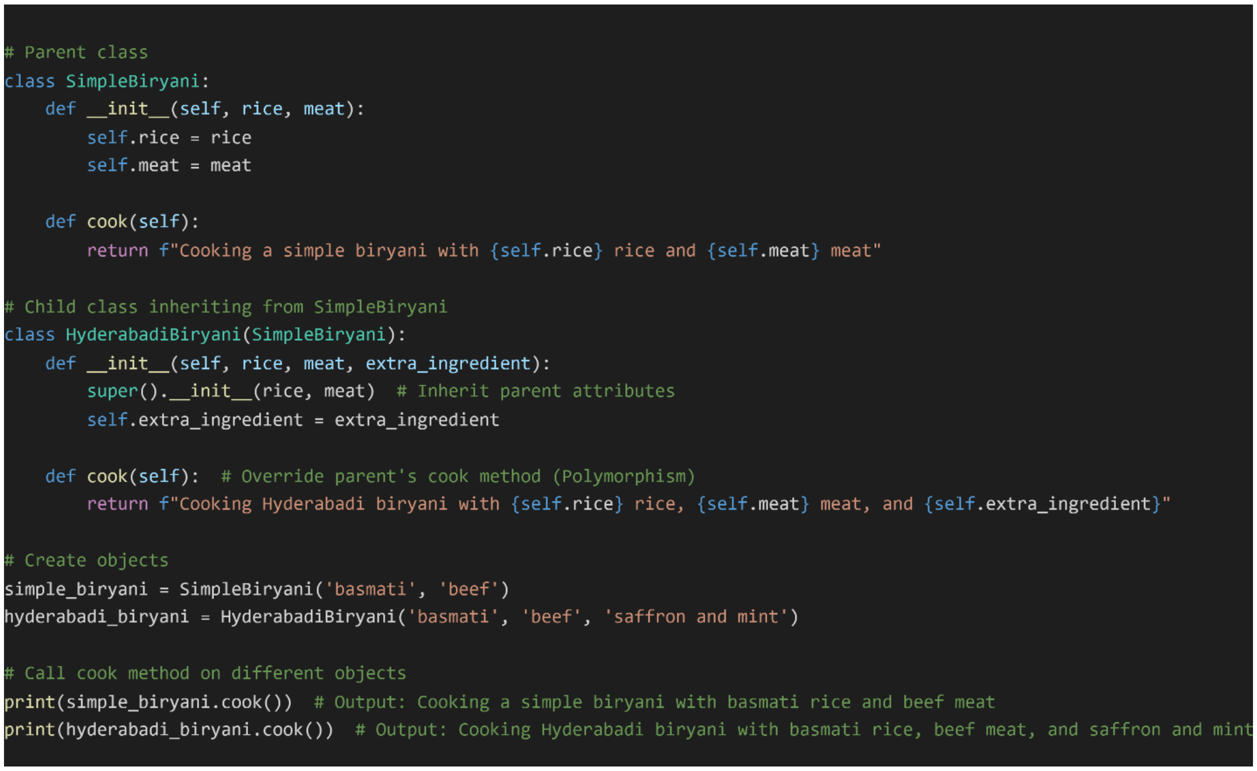
Code Example: Creating a Biryani Class

```
# Define the Biryani class (blueprint)
class Biryani:
    def __init__(self, rice, meat):
        self.rice = rice  # Attribute for rice type
        self.meat = meat  # Attribute for meat type

    def serve(self):
        return f"Serving biryani with {self.rice} rice and {self.meat} meat"

# Create an object of Biryani class
biryani = Biryani('basmati', 'beef')

# Call the serve method
print(biryani.serve())  # Output: Serving biryani with basmati rice and beef meat
```

Explanation:

- **Class Definition**: Biryani is the blueprint with attributes rice and meat.
- **Initializer (__init__)**: Sets the initial values of rice and meat for each object using self.
- **Method (serve)**: Returns a string describing the biryani, accessing attributes via self.
- **Object Creation**: biryani is an instance of the Biryani class, with basmati rice and beef meat.
- **Key Insight**: The class allows multiple objects to be created with the same structure but different values (e.g., biryani2 = Biryani('pulao', 'chicken')).

Code Example: Inheritance and Polymorphism with Biryani



```python
# Parent class
class SimpleBiryani:
    def __init__(self, rice, meat):
        self.rice = rice
        self.meat = meat

    def cook(self):
        return f"Cooking a simple biryani with {self.rice} rice and {self.meat} meat"

# Child class inheriting from SimpleBiryani
class HyderabadiBiryani(SimpleBiryani):
    def __init__(self, rice, meat, extra_ingredient):
        super().__init__(rice, meat)  # Inherit parent attributes
        self.extra_ingredient = extra_ingredient

    def cook(self):  # Override parent's cook method (Polymorphism)
        return f"Cooking Hyderabadi biryani with {self.rice} rice, {self.meat} meat, and {self.extra_ingredient}"

# Create objects
simple_biryani = SimpleBiryani('basmati', 'beef')
hyderabadi_biryani = HyderabadiBiryani('basmati', 'beef', 'saffron and mint')

# Call cook method on different objects
print(simple_biryani.cook())  # Output: Cooking a simple biryani with basmati rice and beef meat
print(hyderabadi_biryani.cook())  # Output: Cooking Hyderabadi biryani with basmati rice, beef meat, and saffron and mint
```

Explanation:

- **Inheritance**: HyderabadiBiryani inherits from SimpleBiryani using SimpleBiryani in parentheses, accessing rice and meat via super().__init__().
- **Polymorphism**: The cook() method is overridden in HyderabadiBiryani to include extra_ingredient, demonstrating how the same method name produces different results.

- **Super Keyword**: super().__init__(rice, meat) ensures the parent class's attributes are initialized in the child class.
- **Key Insight**: Inheritance reuses the parent's attributes, while polymorphism allows the child to customize behavior (e.g., adding saffron and mint to Hyderabadi biryani).

Code Example: Classes with Methods and Attributes

The class also explored creating methods to access object attributes dynamically, addressing a common issue where hardcoding values leads to identical outputs.

```python
# Define a Student class
class Student:
    def __init__(self, name, age, roll_number):
        self.name = name
        self.age = age
        self.roll_number = roll_number

    def student_info(self):
        return f"My name is {self.name}, my age is {self.age}, and my roll number is {self.roll_number}"

# Create an object with dynamic values
student1 = Student("Ali", 23, 1)

# Call the method
print(student1.student_info())  # Output: My name is Ali, my age is 23, and my roll number is 1

# Create another object
student2 = Student("Hassan", 22, 2)
print(student2.student_info())  # Output: My name is Hassan, my age is 22, and my roll number is 2
```

Explanation:

- **Dynamic Values**: The __init__ method takes parameters (name, age, roll_number) to set unique attributes for each object.
- **Self Parameter**: self is required in methods to access object attributes (e.g., self.name).
- **Method (student_info):** Returns a formatted string with the object's attributes, ensuring each object produces unique output.
- **Common Errors Fixed**:

    - **Missing self**: Forgetting self as the first parameter in methods causes errors (e.g., TypeError: student_info() takes 0 positional arguments but 1 was given).
    - **Indentation**: Incorrect indentation (e.g., extra spaces) leads to IndentationError, as seen in the transcript.
- **Key Insight**: Using self ensures attributes are tied to the object, and dynamic parameters make objects unique (e.g., student1 and student2 have different values).

Why This Is Exciting

Revisiting OOP with a biryani theme made the concepts engaging and memorable:

- **Classes as Blueprints**: Like a biryani recipe, classes define a structure for creating multiple objects (plates of biryani) without redefining the recipe each time.
- **Inheritance as Family Recipes**: Passing down recipes (attributes/methods) to child classes, with the ability to add new ingredients (e.g., saffron in Hyderabadi biryani).
- **Polymorphism as Different Flavors**: Overriding methods to create unique behaviors, like customizing biryani flavors.
- **Encapsulation as Secret Masala**: Hiding implementation details to protect the "recipe" from unauthorized access.
- **Abstraction as Chef's Magic**: Focusing on the final dish (interface) without revealing the cooking process (implementation).

Together, these concepts enable structured, reusable, and creative coding, preparing students for real-world programming challenges.

Key Messages:

- **Revise Thoroughly**: Review all OOP concepts (Classes, Polymorphism, Inheritance, Abstraction, Encapsulation) before the test, as they are foundational for future topics (e.g., AI Agents in Quarter 4).
- **Practice Actively**: Engage with practical examples (e.g., biryani-themed code) to understand OOP, avoiding copy-pasting to build genuine skills.
- **Join Discord**: Access class resources and communicate via the Saturday 2–5 PM batch channel on Discord.
- **Prepare for the Test**: The next Saturday session will be a test, not a regular class. Review all topics and complete assignments.
- **Skills Over Degrees**: The IT industry values practical problem-solving, so focus on mastering Python and OOP.

Additional Notes

- **Google Colab**: Used for coding exercises. Ensure familiarity with its interface for running class code.
- **OOP Importance**: Essential for Quarter 4 topics (e.g., AI Agents) and industry projects.
- **Test Preparation**: The upcoming test is critical; failing may result in course discontinuation.
- **Career Focus**: Stay committed to learning Python for opportunities in high-paying IT roles.
- **Biryani Theme**: The biryani analogy (e.g., recipes, flavors, chef's magic) was used to make OOP relatable, reinforced by AI-generated images and poetry shared in class.

⚠ Important Reminder

Please make sure to review and understand all 9 steps/topics before the exam. Due to limited class time, we couldn’t cover everything in class — so it's essential to go through the remaining topics on your own.

Stay consistent and keep learning!

🏁 Final Words:

The rest is up to you now! Learn actively, engage consistently, and participate confidently in class—even wrong answers help you grow. Practice coding daily to master OOP concepts. The test is coming soon, so stay prepared. The next regular class will be on the Saturday after the test, where we’ll revise these topics further.

Allah Hafiz — See you in the next class!

Repo Link: GIAIC-Sat-Afternoon

This is my repository where I have uploaded all the class codes, assignments, and detailed summaries. You can go through each class's code and its full detailed summary without any hesitation, and it will also help you prepare well for your quiz.

Thank You for Reading!

Hope you understood Class 8 well.

"The capacity to learn is a gift; the ability to learn is a skill; the willingness to learn is a choice." – *Brian Herbert*