rtp

week 0_1:

object oriented programming

rajin teaches programming

OUTCOME:

The goal of this handout is to help you understand what OOP is and why Java is a preferred language for learning and applying OOP principles.

WHAT IS OOP?

Object Oriented Programming (OOP) is a programming **paradigm/concept** that uses **objects** and **classes** to **structure** software **programs.** It focuses on **modeling real-world entities** and the **interactions** between them.

- KEY CONCEPTS:

Class: A **class** is a **blueprint** for **creating objects.** It **defines** the **data** (**attributes**) and **behavior** (**methods**) that the **objects** created **from** the **class** will have.

Object: An **object** is an **instance** of a **class.** It is a **self-contained entity** that consists of **attributes** and **methods.**

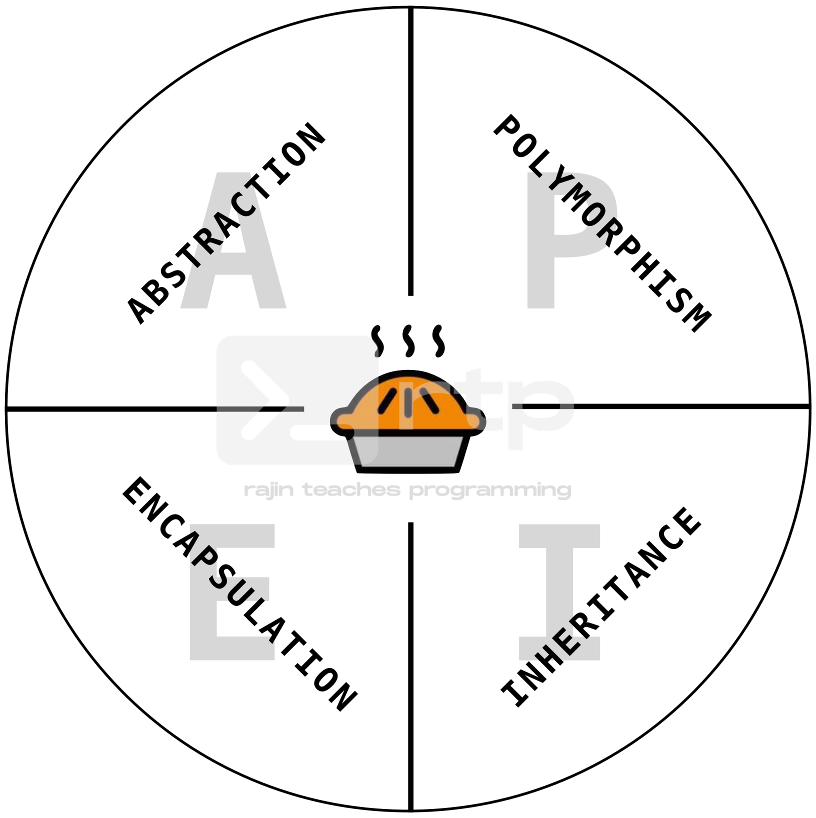
Abstraction: The **concept** of **hiding** complex **implementation details** and **showing only** the **necessary features** of an **object.** It helps in managing complexity by breaking down the program into manageable parts.

Polymorphism: The **ability** of **different classes** to **respond** to the same **method call** in **different ways.** It allows for **methods** to be **overridden** or **overloaded.**

Inheritance: A mechanism where one class **(subclass/child class) inherits attributes** and **methods** from **another** class **(superclass/ parent class).**

Encapsulation: This is **practice** of **keeping** an **object's state (its attributes) private** and providing **public methods** to **access** and modify that state. This ensures data **integrity** and hides the **internal implementation** details.

The last 4 properties learned are called the 4 pillars of Object Oriented Programming. They can be easily remembered using the contraction, **A P I E .**



a contraction and diagram to remember the 4 pillars of OOP.

- ANALOGIES:

To help you understand OOP concepts better, let us create real world analogies for each case.

Class: Think of a class as a blueprint for a house. The blueprint defines the structure and features of the house (number of rooms, layout, etc.). Each house built from this blueprint is an object. Even though they all share the same blueprint, each house (object) can have different colors, furnishings, etc.

Abstraction: When you drive a car, you use a simple interface (steering wheel, pedals) without needing to understand the complex mechanics of the engine. Abstraction in OOP allows programmers to use objects without needing to know the intricate details of their implementation.

Polymorphism: Think of a person who can be a teacher, a parent, and a friend. In different contexts, the person behaves differently. Similarly, polymorphism allows objects to take on multiple forms. For example, a "draw" method might behave differently depending on whether it's called on a circle, square, or triangle object.

Inheritance: Imagine you have a general blueprint for a vehicle. This blueprint includes basic features like wheels and an engine. Now, you can create a more specific blueprint for a car that inherits these basic features but also adds new features like doors and a trunk. Similarly, a motorcycle blueprint would inherit the basic vehicle features but add handlebars and two wheels instead of four.

Encapsulation: Consider a capsule pill. The medicine inside is protected and can only be accessed by taking the pill. In OOP, encapsulation ensures that an object's data is protected and can only be accessed or modified through specific methods, much like the medicine can only be accessed by taking the capsule.

BENEFITS:

Modularity: Code is organized into **objects,** making it easier to manage and maintain.
Reusability: Classes can be **reused** across different programs, reducing redundancy.
Scalability: Programs can be **scaled** easily by adding **new** objects and classes.
Maintainability: Code is **easier** to **understand** and **modify** due to its modular nature.

INTRODUCTION TO JAVA:

Java is a **high-level, class-based, object-oriented** programming language designed to have as **few implementation dependencies** as possible. It is intended to let application developers **write once, run anywhere** (**WORA**), meaning that **compiled** Java code can **run** on **all platforms** that **support** Java **without** the need for **recompilation.**

- A BRIEF HISTORY:

1991: Java was created by James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems (now acquired by **Oracle** Corporation) as a part of the Green Project.
1995: Java 1.0 was released to the public. It quickly gained popularity due to its **platform independence** and **robustness.**
2004: Java 5.0 introduced several new features, including **generics, enumerated types**, and **metadata.**
2017: Oracle introduced a faster release cycle, with a new version of Java being released every six months.

WHY JAVA?

Java is designed to be **platform-independent** at **both** the **source** and **binary** levels, making it **ideal** for **cross-platform** applications. Furthermore, Java provides a vast **standard library** (API) that supports **everything** from **data structures** to **networking**, making it **easier** to develop robust applications. To add to all that, Java has a **large, active** community and a wealth of **documentation** and **resources** available online. Finally, Java has **built-in** security features that help **protect** applications from malicious attacks. Another reason Java was chosen is that it is **widely** used in industry, making it a **valuable** skill for developers.

KEY CONCEPTS:

Class Declaration: A **class** in Java is like a **blueprint** for creating **objects.** It defines the **attributes** and **methods** that the objects will have.

Main Method: The **main method** is the **entry point** of any Java program. It's where the program **starts** execution.

Variables: Containers for storing data values.

Methods: Blocks of code that perform a **specific** task (functions).

Control Structures: Constructs that control the flow of execution, such as **if-else statements** and **loops.**

OBJECT ORIENTED CONCEPTS:

Class and Object: A **class** is a **blueprint,** and an **object** is an **instance** of that class.

Inheritance: One **class** can **inherit properties** and **behaviors** from **another** class.

Encapsulation: Keeping the **data** (**attributes**) of an **object safe** from **outside interference** and **misuse.**

Polymorphism: Different classes can be treated as **instances** of the **same class** through a **common interface.**

Abstraction: Simplifying complex reality by **modeling** classes **appropriate** to the problem.

SUMMARY:

OOP is a **programming paradigm** based on the concept of **objects.**

Java is a **versatile, platform-independent** language that is well-suited for OOP.

Key **OOP** concepts **include classes, objects, inheritance, encapsulation, polymorphism,** and **abstraction.**

Understanding these concepts is **crucial** for becoming proficient in Java and OOP.

FAQs:

What is the difference between a class and an object?
A class is a blueprint for objects, defining their attributes and methods. An object is an instance of a class.

Why is encapsulation important?
Encapsulation protects the internal state of an object and prevents unauthorized access, promoting data integrity and security.

How does inheritance work in Java?
Inheritance allows a class to inherit properties and behaviors from another class, enabling code reuse and hierarchical classification.

What is the advantage of polymorphism?
Polymorphism allows methods to be used interchangeably, enhancing flexibility and the ability to extend code without modifying existing structures.

rtp

next class 1_1:

java basics

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com