# Core Java Concepts

- What is object-oriented programming (OOP)?
- What are the four pillars of OOP?
- What is a class?
- What is an object?
- What is a method?
- What is an inheritance?
- What is polymorphism?
- What is encapsulation?
- What is abstraction?

Object-oriented programming (OOP)** is a programming paradigm that uses objects and their interactions to design applications and computer programs. OOP is based on the concept of objects, which can contain data and code. Objects can interact with each other by sending and receiving messages.

The **four pillars of OOP** are:

- **Abstraction:** Abstraction is the process of hiding implementation details and exposing only the functionality to the user. In abstraction, we deal with ideas and not events. This means the user will only know “what it does” rather than “how it does”.
- **Encapsulation:** Encapsulation is the process of binding data variables and methods together in a class. This makes it easier to manage the data and code, and it also helps to protect the data from being accidentally modified.
- **Inheritance:** Inheritance is the process of one class inheriting the properties and methods from another class. This allows us to reuse code and create new classes that are more specific.
- **Polymorphism:** Polymorphism allows objects of different classes to be treated as belonging to a common group. This makes it easier to write code that is flexible and reusable.

**Class:** A class is a blueprint for creating objects. It defines the properties and behaviors that objects of that class will have.

**Object:** An object is an instance of a class. It contains the data and code that is defined in the class.

**Method:** A method is a function that is defined within a class. Methods can be used to perform actions on the data of the object.

**Inheritance:** Inheritance is the process of one class inheriting the properties and methods from another class. The subclass inherits all of the properties and methods of the superclass.

**Polymorphism:** Polymorphism allows objects of different classes to be treated as belonging to a common group. This makes it easier to write code that is flexible and reusable.

For example, we could have a class called `Animal` with properties such as `name`, `age`, and `species`. We could then have subclasses of `Animal` such as `Dog`, `Cat`, and `Bird`. Each subclass would inherit the properties of the `Animal` class, but it would also have its own unique properties and behaviors.

We could then use polymorphism to write a method that can feed any type of animal. The method would take an `Animal` object as input and then call the `eat()` method on the object. The `eat()` method would be implemented differently for each subclass of `Animal`.

OOP is a powerful programming paradigm that can be used to create complex and reusable applications. It is the basis for many popular programming languages, such as Java, Python, and C++.

Encapsulation** is the process of binding data variables and methods together in a class. This makes it easier to manage the data and code, and it also helps to protect the data from being accidentally modified.

In encapsulation, the data of a class is hidden from the outside world and can only be accessed through the class's methods. This is known as **data hiding**. Data hiding helps to protect the data from being accidentally modified or corrupted.

For example, we could have a class called `BankAccount` with a private data variable called `balance`. The `BankAccount` class would have public methods to deposit and withdraw money from the account. However, outside code would not be able to directly access the `balance` variable.

This means that outside code could not accidentally modify the balance of the account without using the `deposit()` and `withdraw()` methods. This helps to ensure that the balance of the account is always accurate.

**Abstraction** is the process of hiding implementation details and exposing only the functionality to the user. In abstraction, we deal with ideas and not events. This means the user will only know “what it does” rather than “how it does”.

Abstraction is often used in conjunction with encapsulation to create a higher-level view of a class or system. For example, we could have a class called `Car` that encapsulates the complex implementation details of a car, such as the engine, transmission, and brakes. The `Car` class would expose public methods to start, stop, and drive the car.

Outside code would not need to know how the `Car` class works internally in order to use it. The outside code would simply need to know how to call the `start()`, `stop()`, and `drive()` methods.

Abstraction makes it easier to use and maintain complex systems. It also allows us to reuse code and create new classes that are more specific.

Here are some examples of encapsulation and abstraction in Java:

- The `java.lang.String` class encapsulates the implementation details of a string of characters. The `String` class exposes public methods to manipulate strings, such as `length()`, `charAt()`, and `substring()`. However, outside code cannot directly access the internal representation of a string.
- The `java.util.List` interface encapsulates the implementation details of a list of objects. The `List` interface exposes public methods to manipulate lists, such as `add()`, `remove()`, and `get()`. However, outside code cannot directly access the internal representation of a list.
- The `java.io.File` class encapsulates the implementation details of a file on the filesystem. The `File` class exposes public methods to manipulate files, such as `exists()`, `createNewFile()`, and `delete()`. However, outside code cannot directly access the internal representation of a file.

Encapsulation and abstraction are two of the most important concepts in object-oriented programming. They help us to create more modular, reusable, and maintainable code.