The 4 pillars of OOP

OOP is a programming concept based on the idea of objects. Objects can contain data in the form of attributes or properties, and actions in the form of methods. The 4 pillars of OOP are:

1. Abstraction

Abstraction means to only show the necessary information to the user. For example, when you drive a car, you don't need to know how the engine works, you just need to know how to drive it. In programming, abstraction is achieved by using abstract classes and interfaces. This focuses on the essential characteristics of an object while ignoring irrelevant details

Abstraction allows you to create classes that provide a clear and concise interface to interact with objects, hiding the underlying complexity. By defining abstract classes and interfaces, you can establish contracts that specify the behavior and structure expected from implementing classes. Abstraction enhances code modularity, maintainability, and flexibility.

Example:

```
public abstract class Animal {
    public abstract void makeSound();
}
```

```
public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }
}
```

```
public class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.makeSound();
```

```
        cat.makeSound();
    }
}
```

---

2. Inheritance

Inheritance allows you to have code reusability. If you have an existing class and you want to build a new class that uses the things from the existing but you want to add functionality to it. Inheritance allows you to do that. Inheritance is achieved by using the `extends` keyword this makes the class a sub-class. Inheritance is a way for one class to take on the attributes and methods of another parent class. Child classes can override methods defined in their parent classes, but they can also define new methods and attributes (variables). Inheritance is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

Example:

```
public class Animal {
    public void eat() {
        System.out.println("Eating...");
    }
}
```

```
public class Dog extends Animal {
    public void bark() {
        System.out.println("Barking...");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.bark();
    }
}
```

---

3. Polymorphism

Polymorphism means many of the same type. It allows objects of different types to be treated interchangeably based on their common interface or superclass. Polymorphism enables code to be written that can operate on objects of multiple types without explicitly knowing their specific class. This flexibility simplifies code maintenance and promotes extensibility and reusability.

Example:

```
public class Animal {
    public void makeSound() {
        System.out.println("Animal is making a sound");
    }
}
```

```
public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }
}
```

```
public class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.makeSound();
        cat.makeSound();
    }
}
```

---

4. Encapsulation

Encapsulation is the process of hiding the internal details and state of an object to what ever is calling that object. To hide things you use access modifiers. You can hide data by using the `private` keyword. To access the data you must use getter and setter methods. Encapsulation is a way to achieve data hiding. Encapsulation promotes information hiding, data protection, and modularization, making code more robust, maintainable, and reusable.

Example:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
        return this.age;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person("John", 20);
        System.out.println(person.getName());
        System.out.println(person.getAge());
    }
}
```

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No