Exploring Fundamental Object-Oriented Concepts and Their Applications in Software Development

Topics:

- Classes & Objects
- Inheritance

1. Research:

I. Classes & Objects:

Definition:

Classes & Objects are the concept in Object Oriented Programming which reduces redundancy in a codebase and helps you keep things simple. It helps developers follow the DRY principle which stands for **Don't Repeat Yourself**.

Class: A class is a blueprint for creating objects.
Objects: Used to implement the functionalities of a class.

Importance:

A class is a blueprint for creating objects, like if you've implemented some kind of functionality or logic you don't have to copy the exact same logic everywhere in the codebase and make it look like a copy-paste hell. Instead, what you can do is create a different class, implement that functionality there, and create objects of the class anywhere in the codebase where you need it. This improves code readability and doesn't make your codebase look like hell.

Real-world example:

Here's a real word example of classes and objects which make it easy to understand the reason behind the concept:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
```

```
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Output:

5
5

What this example does is create a variable called x in a class. Now if we wanted it in any other place, instead of rewriting it, we just create an object and call it wherever we want it to.

Another example that is actually real word and not some fictional mambo-jumbo like the one above is:

```
const mongoose = require('mongoose')

const carSchema = new mongoose.Schema({
   name: { type: String, required: true },
   brand: { type: String, required: true },
   pricePerDay: { type: Number, required: true },
   description: { type: String, required: true },
   image: { type: String, required: true }
})

carSchema.virtual('reviews', {
   ref: 'Review',
   localField: '_id',
   foreignField: 'car',
   options: {
       sort: { createdAt: -1 }
   }
})

carSchema.set('toJSON', { virtuals: true })
carSchema.set('toObject', { virtuals: true })

module.exports = mongoose.model('Car', carSchema);
```

This example is from mongodb schema implemented inside Javascript backend code (NodeJS/express).

What it does is that it creates the object of the Mongoose Schema as the Schema you want it to be and it lets you implement the functionality by just calling a method/function from the class which was prebuilt by mongodb dev.

This saves thousands of hours in development and lets you implement the logic inside your project without you needing to reinvent the wheel.

Benefits:
The one key benefit of using classes & objects is that they promote code-reusability and reduces the chances of code redundancy

Additionally, classes enable data hiding and encapsulation. By declaring certain data members as private or protected, classes can restrict direct access to sensitive information, enhancing security and preventing unintended modifications

Another advantage of classes is the ability to implement inheritance. Classes can inherit properties and methods from other classes, allowing for the creation of hierarchical relationships. This promotes code reuse and facilitates the development of complex systems by building upon existing classes.

In summary, classes and objects offer benefits such as code reusability, data encapsulation, inheritance, and the ability to model real-world entities, making them fundamental concepts in object-oriented programming.

II. Inheritance:

Inheritance is a concept which lets you inherit the attributes of other classes into your class. It’s like you inheriting your parents property or anything. We group the "inheritance concept" into two categories:

- subclass (child) - the class that inherits from another class

superclass (parent) - the class being inherited from

Importance:
Inheritance in object-oriented programming (OOP) offers several benefits, including code reusability, improved code organization.

Instead of you typing all the stuff again and again in your codebase it lets you just inherit the attributes or properties of the class into your class.

Real World code example:

```
class Vehicle {
  protected String brand = "Ford";         // Vehicle attribute
  public void honk() {                     // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}
class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {
    // Create a myCar object
    Car myCar = new Car();
    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();
    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Output:

```
Tuut, tuut!

Ford Mustang
```

Benefits:
The benefits of inheritance include the ability to inherit from the parent class, which reduces redundancy and lets you follow the DRY principle.

Limitations:
The limitations of inheritance is that sometimes it makes it harder to change a codebase without affecting the other codebases. If you happen to make a change to a class it could cause the other instances inheriting your properties to crash.

It may not look that horrifying for your simple todo-app project but that is a really big problem for industrial-scale software codebases.

2. Comparative Analysis:

Classes & Objects	**Inheritance**
Classes define structure and behaviour; Objects are instance of classes	Inheritance allows a class to acquire properties and methods of another class
Organize code and represent real world entities	Promotes code reuse and hierarchical relationships
Blueprint → House; `Car` class → `myCar` object	Child inherits traits from a parent; `Dog` inherits `Animal`
Basic unit of OOP; foundation for	Used after class/object design to

building structure	reduce duplication
Straightforward, beginner-friendly	More advanced; can lead to tight coupling if misused

Relationship between the Two:

Classes and objects are the **foundation** of object-oriented programming — without them, you can't even start thinking about inheritance. Inheritance is essentially a feature **built on top** of classes. It allows one class to derive from another, making it easier to reuse existing code structures. In short: **you need classes before you can inherit from them**, and objects are the final outcome you interact with in a program.

3. Critical Reflection:

When I first started coding, concepts like “class” and “inheritance” felt like fancy textbook terms. But over time, especially working with JavaScript and Node, I realized they’re actually pretty practical — and kinda essential if you want your code to make sense six months later.

Classes and objects help you build your code like it’s made of building blocks. Instead of repeating the same stuff over and over, you define a structure once (a class), then use it to create as many versions (objects) as you need. It’s like making a template for a car, then building different models from it. This makes your code easier to understand, reuse, and debug.

Inheritance takes that a step further. It lets you take a base class and extend it with more specific features. Like, if you have a `Vehicle` class, you can make `Car`, `Truck`, or `Bike` inherit from it instead of writing everything from scratch each time. That’s a huge win in bigger projects where things repeat a lot.

But inheritance isn’t always perfect. If your base class is messy, every child class suffers — kind of like a family curse in code form. That’s why a lot of modern devs recommend using composition more carefully or keeping inheritance shallow. Still, when used right, it saves a lot of time.

In modern languages:

- **Java** uses inheritance a lot, especially with abstract classes and interfaces.

- **Python** is super flexible — you can inherit from multiple classes, though that can also get messy if not handled well.

- **C++** gives you full control, which is powerful but also dangerous (you can shoot yourself in the foot fast if you're not careful).

- In **JavaScript**, it’s all prototype-based under the hood, but ES6 classes make it easier to think in OOP terms. I’ve used them with Mongoose models or even to structure things in React back when class components were common.

Honestly, Understanding these concepts means you can avoid building spaghetti code that even you don’t understand after a week. Because trust me — legacy code is your worst nightmare.