Object-Oriented Programming (OOP) is a paradigm in computer science that uses objects, which are instances of classes, to design applications and software. These classes serve as blueprints for creating objects, each of which represents an entity that can be found in the real world. This approach to programming is akin to constructing a model using Lego blocks, where each block represents an object, and the instructions for a Lego set are analogous to a class. The use of classes and objects in OOP allows for a more structured and organized approach to programming, leading to code that is easier to understand, maintain, and reuse.

1. Encapsulation is like a piggy bank. You can put money in or take it out, but you can't mess with what's inside directly. In programming, this means we protect our data from accidental changes. For example, in a BankAccount class, we don't let anyone access the balance directly. They have to use deposit or withdraw methods.

2. Inheritance is like a family tree. A child inherits traits from their parents. In programming, a class can inherit features from another class. For example, a Car class could inherit from a Vehicle class. Every car is a vehicle, but cars have some extra features.

3. Polymorphism is a fancy word for flexibility. It allows us to use one interface for a general class of actions. For example, different shapes calculate area in different ways, but we can call the calculateArea() method on any shape.

4. Abstraction is about making things simpler. It's like using a car - you don't need to know how the engine works to drive. In programming, we hide complex details and show only what's necessary. For example, a Car class might have a start() method. The user doesn't need to know what happens under the hood when the car starts.

For instance, in a flight reservation system, classes could represent entities like Flight, Passenger, Ticket, etc. These classes would encapsulate data and behaviors related to these entities, promoting code organization, reusability, and maintainability. Inheritance could be used to model different types of flights or passengers, Polymorphism to handle different pricing strategies, and Abstraction to hide complex operations like seat allocation or flight scheduling.