Object-Oriented Programming (OOP) is a programming paradigm that is based on the concept of objects. OOP uses various concepts to implement the concept of objects, which include the following:

1. Encapsulation: Encapsulation refers to the practice of keeping the internal workings of an object hidden from the outside world, and only exposing a public interface through which other objects can interact with it. This helps to ensure that an object's internal state remains consistent, and prevents other objects from making unauthorized modifications to it. In real-world applications, encapsulation is used to protect sensitive information and ensure the integrity of data.
2. Abstraction: Abstraction refers to the process of simplifying complex systems by breaking them down into smaller, more manageable pieces. In OOP, abstraction is achieved through the use of classes and objects, which allow developers to encapsulate complex functionality into reusable components. This makes it easier to maintain and modify code over time, and allows developers to focus on the high-level functionality of an application without worrying about the details of how it is implemented.
3. Inheritance: Inheritance is the ability of one class to inherit the properties and methods of another class. This allows developers to create a hierarchy of classes, where each class builds on the functionality of the classes that came before it. Inheritance is useful in real-world applications because it allows developers to create reusable code that can be easily extended and modified as needed.
4. Polymorphism: Polymorphism refers to the ability of an object to take on multiple forms. In OOP, this is achieved through the use of interfaces and abstract classes, which allow developers to define a common set of behaviors that can be shared across multiple classes. This allows developers to write code that can work with objects of different types, without needing to know the specific details of each object's implementation.

Overall, the use of OOP concepts in real-world applications provides several benefits, including improved code organization, increased code reusability, and better code maintenance over time. By encapsulating complex functionality into reusable components, developers can build more robust and scalable applications, and can more easily adapt to changing requirements over time.