**OOP WITH JAVA**

**Beginner Level:**

1. **Class and Object Basics:** Create a simple class (e.g., Person) with attributes (e.g., name, age) and methods (e.g., getters and setters).
2. **Inheritance:** Implement a hierarchy of classes, such as a base class (e.g., Shape) and subclasses (e.g., Circle and Rectangle) that inherit properties and methods from the base class.
3. **Polymorphism:** Create an interface (e.g., Drawable) and implement it in multiple classes (e.g., Circle, Rectangle) to demonstrate polymorphism.
4. **Encapsulation**: Design a class with private attributes and provide public methods to access and modify those attributes.
5. **Constructor Overloading**: Practice creating multiple constructors with different parameter lists for a class.

**Intermediate Level:**

1. **Composition:** Create a complex object by composing it from simpler objects. For example, build a Car class that contains instances of Engine, Wheel, and Battery classes.
2. **Abstraction:** Design an abstract class (e.g., Animal) with abstract methods (e.g., eat() and sound()) and concrete subclasses (e.g., Dog and Cat) that implement these methods.
3. **Interfaces:** Implement multiple interfaces in a class (e.g., Comparable and Serializable) and use them to demonstrate interface-based polymorphism.
4. **Enum Types:** Define an enumeration type (e.g., DayOfWeek) to represent a set of named constants.
5. **Association and Aggregation:** Model real-world relationships between classes using association and aggregation. For example, create a University class with Department objects.

**Advanced Level:**

1. **Design Patterns:** Implement common design patterns like Singleton, Factory, Observer, or Strategy in your Java applications.
2. **Inversion of Control (IoC):** Use a dependency injection framework like Spring to manage object creation and dependencies in a complex application.
3. **Exception Handling:** Implement custom exceptions and handle exceptions appropriately in your application.
4. **Database ORM:** Utilize an Object-Relational Mapping (ORM) framework like Hibernate to map Java objects to database tables and perform database operations.
5. **Unit Testing:** Write JUnit or TestNG tests to verify the correctness of your classes and methods.

**Expert Level:**

1. **Concurrency:** Create multithreaded applications with proper synchronization and concurrency control mechanisms.
2. **Design and Architectural Patterns:** Apply architectural patterns like Model-View-Controller (MVC), Microservices, or Domain-Driven Design (DDD) to design large-scale Java applications.
3. **Refactoring:** Take existing code and refactor it to improve its design, maintainability, and readability.
4. **Performance Optimization:** Identify and optimize performance bottlenecks in your Java application using profiling tools and techniques.
5. **Enterprise-Level Application:** Develop a comprehensive enterprise-level application that incorporates various OOP concepts, design patterns, and best practices.