I. What is the Single Responsibility Principle? Explain in detail different source code one following the SRP and other violating the SRP.

The Single Responsibility Principle states that a class should have only one Responsibility.

Following SRP*

```
~

public class UserData {
    public void saveUser() {
        System.out.println("Saving user data");
    }
}

public class NotificationService {
    public void sendEmail() {
        System.out.println("Sending email");
    }
}

public class UserService {
    private UserData userData;
    private NotificationService notificationService;

    public UserService() {
        userData = new UserData();
        notificationService = new NotificationService();
    }

    public void registerUser() {
        userData.saveUser();
        notificationService.sendEmail();
    }
}
```

Violating SRP

```
~
public class UserService {
    public void saveUser() {
        System.out.println("Saving user data");
    }

    public void sendEmail() {
        System.out.println("Sending email");
    }

    public void registerUser() {
        saveUser();
        sendEmail();
    }
}
```

II. What is the Dependency Inversion Principle? List out its advantages. Explain how DI principle makes our code more efficient and testable.

Dependency Inversion Principle states that high level modules should depend on abstractions.
Advantages:

- Loose Coupling
- Easier To Test: If everything is dependent on abstractions, they can be replaced with a mock test.

- Flexibility: New modules can be added without modifying the high-level code.

How Dependency Injection makes our code more efficient and testable:

- Efficiency: By decoupling components, changes in one part of the system don’t require changes somewhere else, reducing refactoring and making the system easier to extend.
- Testability: Since high-level code depends on abstractions, it's easy to mock dependencies in unit tests, allowing isolated testing of components.

III. What is the Open Closed Principle? Explain how it helps to achieve Runtime Polymorphism.

Open Closed Principle basically states that a class should be open to extend but closed to modify. Polymorphism allows different objects to be treated as objects of a common parent, since OCP promotes runtime Polymorphism by encouraging the use of abstractions, it acheives runtime Polymorphism. For example:

```
~

interface PaymentMethod {
    void processPayment();
}

class CreditCardPayment implements PaymentMethod {
    public void processPayment() { System.out.println("Credit Card Payment"); }
}

class PaymentProcessor {
    void process(PaymentMethod method) { method.processPayment(); }
}

public class Main {
    public static void main(String[] args) {
        PaymentProcessor processor = new PaymentProcessor();
        processor.process(new CreditCardPayment());
    }
}
```

The correct `processPayment()` is called at runtime based on the type of the object passed which demonstrates runtime polymorphism

IV. What is Interface Segregation Principle? Explain how it helps to achieve Multiple Inheritance in Java.

Interface Segregation Principle states that a class shouldn't be forced to implement interfaces that dont need to be implemented. ISP helps acheive multiple Inheritance because ISP focuses on creating small interfaces enabling a class to implement multiple interfaces thus acheiving Multiple Inheritance

V. What is the Liskov Substitution Principle? Explain the two different source code which follows LSV and other violating the LSP.

The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

Follows

```
~

class Bird {
    void fly() { System.out.println("Flying"); }
}

class Sparrow extends Bird {
    void fly() { System.out.println("Sparrow Flying"); }
}
```

Doesnt Follow

```
~

class Bird {
    void fly() { System.out.println("Flying"); }
}

class Ostrich extends Bird {
    void fly() { throw new UnsupportedOperationException("Ostriches can't fly"); }
}
```