Design Patterns - Hallmarks of Good Architecture

SOLID Principles of **Object-Oriented Programming**

Practice Session

Interface Segregation Principle (ISP)

```dart
abstract class SmartDevice {
  void makeCall();
  void sendEmail();
  void browseInternet();
  void takePicture();
}

class Smartphone implements SmartDevice {
  @override
  void makeCall() {
    print('Making a call...');
  }

  @override
  void sendEmail() {
    print('Sending an email...');
  }

  @override
  void browseInternet() {
    print('Browsing the Internet...');
  }

  @override
  void takePicture() {
    print('Taking a picture...');
  }
}
```

```dart
class SmartWatch implements SmartDevice {
  @override
  void makeCall() {
    print('Making a call...');
  }

  @override
  void sendEmail() {
    throw UnimplementedError('This device cannot
send emails');
  }

  @override
  void browseInternet() {
    throw UnimplementedError('This device cannot
browse the Internet');
  }

  @override
  void takePicture() {
    throw UnimplementedError('This device cannot
take pictures');
  }
}
```

Design Patterns - Hallmarks of Good Architecture

Interface Segregation Principle (ISP)

Hints:

1. Identify methods in the interface that are not relevant to all classes implementing the interface.
2. Split the interface into smaller, more specific interfaces.
3. Have each class implement only the interfaces it needs.

Design Patterns - Hallmarks of Good Architecture

```
abstract class Phone {
  void makeCall();
}

abstract class EmailDevice {
  void sendEmail();
}

abstract class WebBrowser {
  void browseInternet();
}

abstract class Camera {
  void takePicture();
}

class SmartWatch implements Phone {
  @override
  void makeCall() {
    print('Making a call...');
  }
}
```

```
class Smartphone implements Phone, EmailDevice,
WebBrowser, Camera {
  @override
  void makeCall() {
    print('Making a call...');
  }

  @override
  void sendEmail() {
    print('Sending an email...');
  }

  @override
  void browseInternet() {
    print('Browsing the Internet...');
  }

  @override
  void takePicture() {
    print('Taking a picture...');
  }
}
```

Design Patterns - Hallmarks of Good Architecture

Interface Segregation Principle (ISP)

1. In the refactored solution, the `SmartDevice` interface is segregated into four interfaces: `Phone`, `EmailDevice`, `WebBrowser`, and `Camera`.
2. The `Smartphone` class implements all four interfaces, while the `SmartWatch` class implements only the `Phone` interface.
3. This way, the `SmartWatch` class is not forced to implement the `sendEmail`, `browseInternet`, and `takePicture` methods, which it doesn't need.
3. The original (bad) code violated the **Interface Segregation Principle** because it forced the `SmartWatch` class to depend on methods that it didn't use.
4. This made the `SmartWatch` class implement methods throwing an `UnimplementedError`, which could lead to runtime errors.