Interfaces - Real World Examples

1. **Payment Processing System**
    - An interface for different types of payment methods such as Credit Card, PayPal, and Bank Transfer.
2. **Notification System**
    - An interface for sending notifications via different channels like Email, SMS, and Push Notifications.
3. **Data Storage System**
    - An interface for saving data in different storage systems like SQL Database, NoSQL Database, and File System.

Example 1: Payment Processing System

PaymentProcessor Interface:

```
interface PaymentProcessor {
    void processPayment(double amount);
    default void paymentDetails() {
        System.out.println("Processing payment...");
    }
}
```

CreditCardPayment Class:

```
class CreditCardPayment implements PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of
$" + amount);
```

```
    }
}
```

PayPalPayment Class:

```
class PayPalPayment implements PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment of $" +
amount);
    }
}
```

Main Class:

```
public class PaymentMain {
    public static void main(String[] args) {
        PaymentProcessor creditCardPayment = new CreditCardPa
yment();
        creditCardPayment.processPayment(150.0);
        creditCardPayment.paymentDetails();

        PaymentProcessor payPalPayment = new PayPalPayment();
        payPalPayment.processPayment(75.0);
        payPalPayment.paymentDetails();
    }
}
```

Example 2: Notification System

Notifier Interface:

```
interface Notifier {
    void sendNotification(String message);
}
```

EmailNotifier Class:

```
class EmailNotifier implements Notifier {
    @Override
    public void sendNotification(String message) {
        System.out.println("Sending email notification: " + m
essage);
    }
}
```

SMSNotifier Class:

```
class SMSNotifier implements Notifier {
    @Override
    public void sendNotification(String message) {
        System.out.println("Sending SMS notification: " + mes
sage);
    }
}
```

Main Class:

```
public class NotificationMain {
    public static void main(String[] args) {
        Notifier emailNotifier = new EmailNotifier();
        emailNotifier.sendNotification("Your order has been s
hipped.");

        Notifier smsNotifier = new SMSNotifier();
        smsNotifier.sendNotification("Your OTP is 123456.");
    }
}
```

Example 3: Data Storage System

DataStorage Interface:

```
interface DataStorage {
    void saveData(String data);
    void deleteData(String dataId);
}
```

SQLDatabase Class:

```
class SQLDatabase implements DataStorage {
    @Override
    public void saveData(String data) {
        System.out.println("Saving data to SQL Database: " +
data);
    }

    @Override
    public void deleteData(String dataId) {
        System.out.println("Deleting data from SQL Database w
ith ID: " + dataId);
    }
}
```

FileSystemStorage Class:

```
class FileSystemStorage implements DataStorage {
    @Override
    public void saveData(String data) {
        System.out.println("Saving data to File System: " + d
ata);
    }

    @Override
    public void deleteData(String dataId) {
        System.out.println("Deleting data from File System wi
th ID: " + dataId);
```

```
    }
}
```

Main Class:

```
public class DataStorageMain {
    public static void main(String[] args) {
        DataStorage sqlDatabase = new SQLDatabase();
        sqlDatabase.saveData("Employee data");
        sqlDatabase.deleteData("12345");

        DataStorage fileSystemStorage = new FileSystemStorage
();
        fileSystemStorage.saveData("Backup data");
        fileSystemStorage.deleteData("67890");
    }
}
```

Summary

These examples demonstrate how interfaces can be used to define a contract for various classes that implement specific functionalities, such as payment processing, sending notifications, and data storage. The use of interfaces allows for greater flexibility and scalability, enabling the implementation of different methods for each functionality while adhering to a common interface.