LSP:

1. The example that I came up with involves a DebitCard and GiftCard class. A GiftCard is a type of debit card so it is a child class of DebitCard. However, while you can deposit money into a DebitCard account you can not deposit into a GiftCard account (side note, I know you can with some gift cards but in this example you can't). Now given these two classes, if you had another class that used the DebitCard class and uses its deposit method, then the GiftCard class would fail to follow the LSP principle. This is because there is an expectation that when you deposit money into a card's account then the balance should update accordingly, however as a GiftCard can only have money withdrawn it would fail to follow said expectation. Below is the code:
    a. DebitCard:

```java
import java.util.ArrayList;
public class DebitCard {
        private double balance;
        public DebitCard(double startingBalance) {
                this.balance = startingBalance;
        }
        public void deposit(double deposit) {
                this.balance += deposit;
        }
        public double viewBalance() {
                return this.balance;
        }
        private static boolean testLSP(DebitCard card) {
                double originalBalance = card.viewBalance();
                card.deposit(10);
                return originalBalance + 10 == card.viewBalance();
        }
        public static void main(String[] args) {
                DebitCard bankCard = new DebitCard(100);
                DebitCard giftCard = new GiftCard(100);
                ArrayList<DebitCard> cards = new ArrayList<>();
                cards.add(bankCard);
                cards.add(giftCard);
                for (DebitCard card : cards) {
                        if (DebitCard.testLSP(card)) {
                                System.out.println("LSP test passed");
                        } else {
                                System.out.println("LSP test failed");
                        }
                }
        }
}
```

b. GiftCard:

```java
public class GiftCard extends DebitCard{
      public GiftCard(double startingBalance) {
            super(startingBalance);
      }

      @Override
      public void deposit(double deposit) {
            System.out.println("Deposit not available for giftCards");
      }
}
```

DIP:

1. Changes made:
    a. Added the interface Switchable with abstract methods turnOff() and turnOn()
    b. Updated Lightbulb to implement Switchable
    c. Changed ElectricPowerSwitch to have an arraylist of Switchable objects instead of a reference to lightbulb
    d. Changed ElectricPowerSwitch's press() method to loop through the Switchable array call either turnOff() or turnOn() on each object accordingly.
    e. Updated Control to match and test changes
    f. Added new class fan that implements Switchable, no changes made to ElectricPowerSwitch
    g. Updated Control to include fan and tested, no changes made to ElectricPowerSwitch
2. Code:
    a. Control:

```java
import java.util.ArrayList;
public class Control {
      public static void main(String[] args) {
            LightBulb lightBulb = new LightBulb();
            Fan fan = new Fan();
            ArrayList<Switchable> appliances = new ArrayList<>();
            appliances.add(lightBulb);
            appliances.add(fan);
            ElectricPowerSwitch bulbSwitch = new
ElectricPowerSwitch(appliances);
            bulbSwitch.press();
            bulbSwitch.press();
```

```
        }
}
```

b. ElectricPowerSwitch:

```
import java.util.ArrayList;
public class ElectricPowerSwitch {
        public ArrayList<Switchable> appliances;
        public boolean on;
        public ElectricPowerSwitch(ArrayList<Switchable> appliances) {
                this.appliances = appliances;
                this.on = false;
        }
        public boolean isOn() {
                return this.on;
        }
        public void press() {
                boolean checkOn = isOn();
                if (checkOn) {
                        for (Switchable appliance : appliances) {
                                appliance.turnOff();
                        }
                        this.on = false;
                } else {
                        for (Switchable appliance : appliances) {
                                appliance.turnOn();
                        }
                        this.on = true;
                }
        }
}
```

c. Switchable:

```
public class LightBulb implements Switchable {
        @Override
        public void turnOn() {
                System.out.println("LightBulb: Bulb turned on...");
        }
        @Override
        public void turnOff() {
                System.out.println("LightBulb: Bulb turned off...");
        }
}
```

d. LightBulb:

```
public interface Switchable {
        public abstract void turnOn();
        public abstract void turnOff();
}
```

e. Fan:

```
public class Fan implements Switchable {
	@Override
	public void turnOn() {
		System.out.println("Fan: Fan turned on...");
	}
	@Override
	public void turnOff() {
		System.out.println("Fan: Fan turned off...");
	}
}
```