Exercise #1 - S.O.L.I.D Principles

In this exercise let's go through all the SOLID steps and see how we can improve what is a bad design to start with into a much better design.

Question 1:

How does this code violate SRP? (Single Responsibility Principle)

```
class PizzaShop {
   constructor(name: string, city: string, zipCode: int){ }
   getName() { }
   changeAddress(city:string, zipCode: int) { }
}
```

Question 2:

How does this code violate OCP? (Open-Closed Principle)

```
class PizzaShop {
    constructor(name: string, address: Address){ }
    getName() { }
    getAddress() { }
}
class InvoiceService {
    generateInvoice(shop: MusicShop): String{
        let invoice = "";
        if(company instanceOf A)
          invoice = "format of invoice for A";
        if(company instanceOf B)
          invoice = "format of invoice for B";
        if(company instanceOf C)
          invoice = "format of invoice for C";
        return invoice;
    }
}
```

Question 3:

How does this code violate LCP? (Liskov Substitution Principle)

```
class PizzaShop {
    homeDelivery();
    //...
}
class A extends PizzaShop {
    homeDelivery() {
        return "delivery is free for all our customers";
    }
}
class B extends PizzaShop {
    takeaway() {
        throw new Exception('We do not have home delivery
service');
    }
}
```

Question 4:

How does this code violate ISP? (Interface Segregation Principle)

```
abstract class IPizzaShop{
    //traditional pizzerias
    getOvenBakedPizza()();
    getClassicalBakedPizza();
    //new wave pizzerias
    getElectricOvenBakedPizza();
    getPizzaPocketSquareBakedPizza();
    // all pizzerias
    getDrinks();
}
class TraditionalPizzeria implements IPizzaShop {
    getOvenBakedPizza() {
        //...           }
    getClassicalBakedPizza() {
        //...             }
    getDrinks() {
        //...   }
    getElectricOvenBakedPizza() {
        throw new Exception('We don't do that');
    }
    getPizzaPocketSquareBakedPizza() {
        throw new Exception('We don't do that');
    }
}
class NewWavePizzeria implements IPizzaShop {
    getElectricOvenBakedPizza() {
        //...                   }
    getPizzaPocketSquareBakedPizza() {
        //...                       }
    getDrinks() {
        //...   }
    getOvenBakedPizza() {
        throw new Exception('We don't do that');
    }
    getClassicalBakedPizza() {
        throw new Exception('We don't do that');
    }
}
```

Question 5:

How does this code violate the Dependency Inversion Principle?

```
class PizzaShop {
    getPayment() {
    }
    deliverPizza() {
    }
}
class Customer {
    makePayment() {
    }
    receivePizza() {
    }
}
class Delivery {
    constructor(customer: Customer, pizzaShop: PizzaShop) { }
    deliver() {
        customer.makePayment
        pizzaShop.getPayment
        pizzaShop.deliverPizza
        customer.receivePizza
    }
}
```