# PizzaStore - version2

> Design Document By Lee Kyeong Jun

## Index

---



---

## What's New?

- PizzaStore version 2 에서는, 반복 허용되는 상황을 제외한 모든 에러케이스에 대해 `System.exit(0)` 구문을 추가해 프로그램을 강제 종료 되도록 만들었습니다.
- Abstract class Order으로 인해 영향이 생긴 모든 코드들을 Refactoring 했습니다.
- 실행 케이스와 그 예시를 추가했습니다.
- toString() 함수로 인해 출력되는 구문들의 가독성을 향상했습니다.
- 이제, 아무런 토핑이 없는 Pizza도 생성할 수 있습니다.

---

## Codes

### Order.java

```
package Assignment1_code_Leekyeongjun_2019092824;

public abstract class Order {
	protected Pizza[] pizza;


	public void addPizza(Pizza pizza) {
		Pizza newPizza = new Pizza(pizza);
		if(this.pizza == null) {
			// there was no pizzalist.
			this.pizza = new Pizza[1];
			this.pizza[0] = newPizza;
		}else {
			// there was pizzalist.
			int curCount = this.pizza.length+1;
			Pizza[] newPizzaList = new Pizza[curCount];
			for(int i = 0; i<this.pizza.length; i++) {
				newPizzaList[i] = new Pizza(this.pizza[i]);
			}newPizzaList[curCount-1] = newPizza;
			this.pizza = newPizzaList;
		}

	}

	public void removePizza(int index) {
		if(this.pizza == null) {
			System.out.println("error : There is No PizzaList.");
			System.exit(0);
		}
		if(this.pizza.length == 0) {
			System.out.println("error : There is No Pizza in PizzaList.");
			System.exit(0);
		}
		if(index > this.pizza.length) {
			System.out.println("error : Out of bound error.");
			System.exit(0);
		}
		else {
			Pizza[] removedList = new Pizza[this.pizza.length-1];
			for(int i = 0; i<removedList.length; i++) {
```

```
                        if(i < index) removedList[i] = new Pizza(this.pizza[i]);
                        else if(i >= index) {/* Remove it!*/
                                removedList[i] = new Pizza(this.pizza[i+1]);
                        }
                }
                this.pizza = removedList;
                System.out.println("removed Pizza[" + index + "].");
        }
    }

    abstract public double calculateOrderPrice();

    public String toString() {
            String ret;
            // ret = pizza.toString();
            ret = "Your total will be $" + calculateOrderPrice() + ".";
            return ret;
    }
}
```

- Java의 Array는 한번 고정된 배열 길이에 요소를 추가하거나, 제거할 수 없습니다.
  - 이에 따라, 요소의 추가 제거를 위해, (조금은 비효율적이지만) 새로운 배열을 만들어 변경 사항을 적용한 후, 이를 원해 배열에 덮어 씌우는 형식으로 구현했습니다.
  - `addPizza` 는 길이가 `length+1` 인 새로운 배열을 만들고, `this.pizza` , 즉 원래 배열의 요소를 새로운 배열에 복사한 뒤 마지막 인덱스에 추가할 Pizza를 삽입했습니다.
  - `removePizza` 는 다음 세가지 케이스에 대해 작동하면 안됩니다.
    - `this.pizza` 가 null 일 경우. 즉, Order는 만들어졌지만 안에 PizzaList는 만들어지지 않은 경우.
    - `this.pizza` 의 배열의 길이가 0일 경우, 즉 뺄 것이 없는 경우.
    - `this.pizza` 의 length보다 큰 index 번호에 대한 접근을 요청하는 경우.
    - 다음 세가지 exception을 제외하고는 아래와 같이 동작합니다.
      - 새로운 배열을 만듭니다. 배열의 길이는 original보다 1 작습니다.
      - 배열 요소에 대한 복사를 시작합니다.
        - 만약 삭제 대상 인덱스보다 작은 요소들이라면, 그대로 복사합니다.
        - 만약 삭제 대상 인덱스보다 크거나 같은 요소들이라면, 한칸씩 앞으로 당겨 복사합니다.
- `calculateOrderPrice` 는 abstract method로, 자식 클래스인 InstoreOrder와 OnlineOrder의 `calculateOrderPrice` 의 정의를 강제합니다. 이는 해당 클래스를 살펴보며 알아보겠습니다.
- `toString()` 은 deprecated 되었습니다. Order를 inherit하는 두 클래스만 존재할 수 있기 때문에, Order 자체의 `toString()` 은 두 클래스 상에서 Override 될 예정입니다.

## InStoreOrder.java

```
package Assignment1_code_Leekyeongjun_2019092824;

public class InStoreOrder extends Order {

        private int tableNumber;

        public InStoreOrder(int tableNumber) {
                super();
                this.tableNumber = tableNumber;
        }

        public double calculateOrderPrice() {
                double sum = 0;
                for(int i = 0; i<super.pizza.length; i++) {
                        sum += super.pizza[i].getPrice();
                }
                return sum*1.15;
        }

        public String toString() {
                String ret = "InStore Order\n";

                for(int i = 0; i<super.pizza.length; i++) {
                        ret += "Pizza["+i+"] : " + super.pizza[i].toString() + '\n';
                }ret += "Total Price : " + calculateOrderPrice() + '\n';
                ret += "TableNumber : " + tableNumber + '\n';
                return ret;
        }
}
```

- Order를 extends하는 InStoreOrder입니다.
  - `InStoreOrder()` 는 생성자로, `super()` 를 통해 Order 클래스를 만들고, 해당 요소를 상속합니다.
- `calculateOrderPrice` 는 `super.pizza` 를 통해 배열에 접근하고, 이를 순회하며 마지막에 팁 15%를 추가로 적용해 최종 가격을 리턴합니다.
  - [주의!] 이 함수는 `super.pizza` 가 `addPizza` Method를 통해 할당되어있음이 전제 되어 있습니다. 만약 InStoreOrder가 생성된 후 아무런 Pizza의 추가 없이 해당 메서드를 call 하면 에러를 유발합니다.

## OnlineOrder.java

```java
package Assignment1_code_Leekyeongjun_2019092824;

public class OnlineOrder extends Order {

	private String deliveryAddress;

	public OnlineOrder(String address) {
		super();
		deliveryAddress = address;
	}

	@Override
	public double calculateOrderPrice() {
		double sum = 0;
		for(int i = 0; i<super.pizza.length; i++) {
			sum += super.pizza[i].getPrice();
		}
		return sum+3;
	}

	public String toString() {
		String ret = "Online Order\n";

		for(int i = 0; i<super.pizza.length; i++) {
			ret += "Pizza["+i+"] : " + super.pizza[i].toString() + '\n';
		}ret += "Total Price : " + calculateOrderPrice() + '\n';
		ret += "Address : " + deliveryAddress + '\n';
		return ret;
	}
}
```

- InStoreOrder와 거의 유사합니다. calculateOrderPrice에서 고정 fee 3을 더한다는 점, 그리고 Instance Variable로 String을 갖고 있다는 점이 다릅니다.

## Pizza.java

```java
package Assignment1_code_Leekyeongjun_2019092824;

public class Pizza {
	private int size;
	private boolean hasPeperoni;
	private boolean hasMushrooms;
	private boolean hasCheese;

	Pizza(int _size, boolean _hasPeperoni, boolean _hasMushrooms, boolean _hasCheese){

	size = _size;
	hasPeperoni = _hasPeperoni;
	hasMushrooms = _hasMushrooms;
	hasCheese = _hasCheese;

	if(size <= 0) {
	    System.out.println("error! : Pizza has Invalid Size : " + size);
	    System.exit(0);
	}
	}

	public Pizza(Pizza originPizza) {
		size = originPizza.size;
		hasPeperoni = originPizza.hasPeperoni;
		hasMushrooms = originPizza.hasMushrooms;
		hasCheese = originPizza.hasCheese;
	}

	public String toString() {
		String ret;

		// ret example : "[Size = 28 / Ingredients = Peperoni Mushrooms ]"

		ret = "[Size = " + size + " / Ingredients = ";
		if(hasPeperoni) ret += "Peperoni ";
		if(hasMushrooms) ret += "Mushrooms ";
		if(hasCheese) ret += "Cheese";
		ret += "]";
		return ret;
	}

	public double getPrice() {
		double offset = 0;

		if(hasPeperoni) offset ++;
		if(hasMushrooms) offset ++;
```

```
                if(hasCheese) offset ++;

                double price = size*0.25*((100-(10*offset))/100);
                return price;
        }
}
```

- 이전 프로젝트의 Pizza와 동일합니다. 그러나 하나가 달라졌습니다!
    - 독특한 취향의 손님을 위해, 이제 아무런 토핑이 없는 피자도 허용됩니다.
        - 명세의 main test에서 이런 독특한 취향의 손님이 한 분 계십니다.

## PizzaStore.java

```
package Assignment1_code_Leekyeongjun_2019092824;
import java.util.Scanner; // to use in createOrder(String type)

public class PizzaStore {
        private double cash;
        private Order currentOrder;
        private int peperoniStock;
        private int mushroomStock;
        private int cheeseStock;

        private static int tableNumber = 1;

        PizzaStore() {
                cash = 2.5;
                peperoniStock = 3; // init value changed
                mushroomStock = 3; // init value changed
                cheeseStock = 3; // init value changed
        }

        public void addCash(double amount) {
                if(amount >= 0) cash += amount;
        }

        public void addCash() {
                if(currentOrder != null) {
                        cash += currentOrder.calculateOrderPrice();
                        currentOrder = null;
                }
        }

        public void removeCurrentOrder(int index) {
                if(currentOrder != null) {
                        currentOrder.removePizza(index);
                }
        }


        public void createOrder(String type) {
                Scanner scanner =  new Scanner(System.in);
                if(type.equals("2")) {
                        System.out.println("Chosen : Online");
                        System.out.println("What is delivery address?");
                        String address = scanner.nextLine();
                        OnlineOrder onlineorder = new OnlineOrder(address);
                        currentOrder = onlineorder;
                }
                else if(type.equals("1")) {
                        System.out.println("Chosen : In Store");
                        InStoreOrder instoreorder = new InStoreOrder(tableNumber);
                        tableNumber++;
                        currentOrder = instoreorder;
                }
                else {
                        System.out.println("Error : Invalid Order type");
                        scanner.close();
                        System.exit(0);
                }
        }

        public void AddPizzaToOrder(int pizzaSize, boolean hasPeperoni, boolean hasMushrooms, boolean hasCheese) {
                boolean availableOrder = true;
                int tmpPep = peperoniStock;
                int tmpMus = mushroomStock;
                int tmpChe = cheeseStock;

                if(hasPeperoni && availableOrder) {
                        if(peperoniStock >= 1){peperoniStock -= 1;}
                        else availableOrder = false;
```

```
                }

                if(hasMushrooms && availableOrder) {
                        if(mushroomStock >= 1) {mushroomStock -= 1;}
                        else availableOrder = false;
                }

                if(hasCheese && availableOrder) {
                        if(cheeseStock >= 1) {cheeseStock -= 1;}
                        else availableOrder = false;
                }

                if(availableOrder) {
                        Pizza newpizza = new Pizza(pizzaSize, hasPeperoni, hasMushrooms, hasCheese);
                        currentOrder.addPizza(newpizza);
                }
                else {
                        System.out.println("The order is Not Available.");
                        peperoniStock = tmpPep;
                        mushroomStock = tmpMus;
                        cheeseStock = tmpChe;
                }
        }

        public void restockPeperoni(int amount) {
                double price = amount * 1;
                if(cash >= price) {
                        cash -= price;
                        peperoniStock += amount;
                        System.out.println(amount + " of peperoni purchased. Your current Cash is " + cash);
                }
                else {
                        System.out.println("Not enough Cash. Your current cash is " + cash + " and the price is " + price );
                }
        }

        public void restockMushrooms(int amount) {
                double price = amount * 1.5;
                if(cash >= price) {
                        cash -= price;
                        mushroomStock += amount;
                        System.out.println(amount + " of mushrooms purchased. Your current Cash is " + cash);
                }
                else {
                        System.out.println("Not enough Cash. Your current cash is " + cash + " and the price is " + price );
                }
        }

        public void restockCheese(int amount) {
                double price = amount * .75;
                if(cash >= price) {
                        cash -= price;
                        cheeseStock += amount;
                        System.out.println(amount + " of peperoni purchased. Your current Cash is " + cash);
                }
                else {
                        System.out.println("Not enough Cash. Your current cash is " + cash + " and the price is " + price );
                }
        }

        public String toString() {
                if(currentOrder != null) {
                        return currentOrder.toString();
                }
                else{
                        String ret = "PizzaStore : cash: $" + cash + ", peperoni:" + peperoniStock + ", mushrooms:" + mushroomStock+ ", cheeses:" +
cheeseStock +".";
                        return ret;
                }
        }
}
```

- public 한 mutator `addCash(double amount)` 를 overload하는 `addCash()` 가 추가되었습니다.
  - 아무런 인자도 주어지지 않은 `addCash()` 는 자동적으로 현재 Store의 Current Order의 가격을 계산해 Cash에 추가합니다.
- public 한 mutator `removeCurrentOrder(int index)` 가 추가되었습니다.
  - Current Order의 `removePizza(int index)` 를 간접 호출합니다.
- `createOrder` 메서드가 생겼습니다.
  - 여기서는 currentOrder를 만들게 됩니다.
- `AddPizzaToOrder` 는 지난 버전의 `createOrder` 를 어느정도 포함하는 메서드입니다.

    - order의 availability를 확인한 뒤, 확인된 order에 한해 pizza를 만들어 order에 추가합니다.
        - 이번 버전부터, 모든 토핑을 추가하지 않는 order는 available 한 order입니다.
- [주의!] InStoreOrder의 tableNumber는 Store 별로 static 하게 존재해야 합니다.
    - 즉 Store에서 Order가 들어올때마다 tableNumber를 하나씩 증가시키고, 이를 바탕으로 다음 InStoreOrder가 들어왔을 때 instantiate 합니다.
- `toString()` 의 변화
    - PizzaStore의 `toString` 은 크게 두 역할을 합니다.
        - 현재 가게의 재정 상황에 대한 것
        - 현재 가게가 가진 주문에 대한 것
    - 이 두 가지를 동시에 커버 하기 위해, CurrentOrder가 null이 아닐 경우에 Order에 대한 정보를 제공하도록 변경했습니다.
    - CurrentOrder가 null인 경우, 주문을 받는 상황이 아니므로 현재 가게에 대한 정보를 제공합니다.

## MainClass.java

```
package Assignment1_code_Leekyeongjun_2019092824;
import java.util.Scanner;

public class MainClass {

	public void showErrorMsg(int errcode) {
		String msg = "";
		if(errcode == 0) {
			// Number error.(repeatable)
			// if size has unavailable size, than this message appears.
			msg = "Unavailable Size, Please Retry.";
		}
		else if(errcode == 1) {
			// Command error.(repeatable)
			// if user has typed unavailable command, than this message appears.
			msg = "Unavailable Command, Please Retry.";
		}
		else if(errcode == 2){
			// Ingredient error.(repeatable)
			// if there is no Ingredients in Pizza, than this message appears.
			msg = "Pizza has no Ingredients. Please Retry.";
		}
		else if(errcode == 3) {
			System.out.println("[Error] : Invalid Command");
			System.exit(0);
		}
		else {
			// Unknown error.
			// This should not be appeared.
			msg = "ErrorCode Index error.";
		}
		System.out.println("[Error] : " + msg);
	}

	public void selectAction(PizzaStore store, Scanner input) {

		int cmd = 0;
		System.out.println(store.toString());
		System.out.println("What would you like to do:");
		System.out.println("1: Place an order, 2: buy ingredients");

		cmd = input.nextInt();
		input.nextLine();

		while(!(cmd == 1 || cmd == 2)) {
			showErrorMsg(1);
			cmd = input.nextInt();
			input.nextLine();
		}

		if(cmd == 1) {
			placeOrder(store, input);
		}
		else if(cmd == 2) {
			buyIngredients(store, input);
		}

	}

	public void placeOrder(PizzaStore store, Scanner input) {
		int finished = 0;
		String type;
		System.out.println("What type of order?");
		System.out.println("1: In store, 2: Online, 3: back");
		type = input.nextLine();
		if(type.equals("3")) {
			finished = -1;
```

```
		}

		else {
			store.createOrder(type);
		}

		while(finished == 0) {
		    boolean hasPep= false, hasMus = false, hasChe = false;
		    System.out.println("What size pizza do you want?");
		    int size;
		    size = input.nextInt();
		    input.nextLine();

		    while(size < 0) {
		        showErrorMsg(0);
		        size = input.nextInt();
		        input.nextLine(); // flush nextLine after nextInt();
		    }

		    hasPep = selectIngredient(store, input, "peperoni");
		    hasMus = selectIngredient(store, input, "mushrooms");
		    hasChe = selectIngredient(store, input, "cheese");


		    store.AddPizzaToOrder(size, hasPep, hasMus, hasChe);

		    String cmd;
		    System.out.println("Do you want to order another Pizza? (Y/N)");
		    cmd = input.nextLine();
		    if(cmd.equals("Y")) {
		        finished = 0;
		    }else if(cmd.equals("N")) {
		        finished = 1;
		    }
		}
		// normal case
		if(finished == 1) {
			System.out.println("Your final order is :");
			System.out.println(store.toString());
			System.out.println("Do you want to change your order? (Y/N)");
			String cmd2 = input.nextLine();
			if(cmd2.equals("Y")) {
				changeOrder(store, input);
			}
			else if(cmd2.equals("N")){
				store.addCash();
			}
			else {
				showErrorMsg(3);
			}
		}
	}
	public void changeOrder(PizzaStore store, Scanner input) {
		boolean finished = false;
		while(finished == false) {
			String cmd;
			System.out.println("What do you want to do?");
			System.out.println("1: Add a pizza, 2: Remove a pizza, 3: Nothing");
			cmd = input.nextLine();

			if(cmd.equals("1")) {
			    int size;
			    boolean hasPep= false, hasMus = false, hasChe = false;

			    System.out.println("What size pizza do you want?");

			    size = input.nextInt();

			    input.nextLine(); // flush nextLine after nextInt();
			    while(size < 0) {
			        showErrorMsg(0);
			        size = input.nextInt();
			        input.nextLine(); // flush nextLine after nextInt();
			    }

			    hasPep = selectIngredient(store, input, "peperoni");
			    hasMus = selectIngredient(store, input, "mushrooms");
			    hasChe = selectIngredient(store, input, "cheese");

			    store.AddPizzaToOrder(size, hasPep, hasMus, hasChe);
			}
			else if(cmd.equals("2")) {
				System.out.println("Which pizza do you want to remove?");
```

```
                                int index;
                                index = input.nextInt();
                                input.nextLine();
                                store.removeCurrentOrder(index);
                        }
                        else if(cmd.equals("3")) {
                                finished = true;
                        }
                        else {
                                showErrorMsg(3);
                        }
                }

        }
        public boolean selectIngredient(PizzaStore store, Scanner input, String label) {
            String cmd;
            System.out.println("Do you want "+ label +" on your pizza? Y/N");
            cmd = input.nextLine();
            while(!(cmd.toUpperCase().equals("Y") || cmd.toUpperCase().equals("N"))) {
                showErrorMsg(1);
                cmd = input.nextLine();
            }

            if(cmd.toUpperCase().equals("Y")) {
                return true;
            }else return false;

        }

        public void buyIngredients(PizzaStore store, Scanner input) {
                int cmd;

                System.out.println("What ingredients do you want to buy?");
                System.out.println("1: peperoni, 2: mushrooms, 3: cheese, 4: none.");

                cmd = input.nextInt();
                while(cmd < 1 || cmd >= 5) {
                        showErrorMsg(1);
                        cmd = input.nextInt();
                        input.nextLine();
                }


                if(cmd < 4) {
                        int amount = 0;
                        System.out.println("What amount do you want to buy?");
                        amount = input.nextInt();
                        input.nextLine();
                        if(cmd == 1) {
                                store.restockPeperoni(amount);
                        }
                        if(cmd == 2) {
                                store.restockMushrooms(amount);
                        }
                        if(cmd == 3) {
                                store.restockCheese(amount);
                        }
                }

                if(cmd == 4) {
                        store.addCash(1);
                }

        }
        public static void main(String[] args) {
                Scanner scanner =  new Scanner(System.in);
                PizzaStore myStore = new PizzaStore();
                MainClass mainClassInstance = new MainClass();

                while(true) {
                        mainClassInstance.selectAction(myStore, scanner);
                }

        }
}
```

- `placeOrder(PizzaStore store, Scanner input)` 의 형태가 변경되었습니다.
  - 이제 Order를 먼저 만들고(line 71) → Size를 정한 뒤 (line 76)→ 재료를 선택하고 (line 87~89) → 피자를 추가한뒤 (line 92) → 다음 피자를 주문할 것인지 물어봅니다. (line 95)
    - 만약 Order 만들기가 완료되었다면, 한번 현재까지의 order를 출력해주고, 변경할 것인지 물어봅니다. (line 107)

        - 만약 Order를 변경할 것이라면, `changeOrder(store, input)` 을 실행합니다.
        - 아니라면, 현재까지의 Order를 계산해줍니다.
- `changeOrder(PizzaStore store, Scanner input)` 함수가 새로이 만들어졌습니다.
    - 이 함수는 이미 *완료된* Order에 수정을 할 수 있습니다.
    - [주의!] 음식은 환불이 안됩니다!
        - 한번 Order에 넣어 조리 완료된 피자를 다시 Order에서 제거한다고 해서 사용한 재료가 돌아오지는 않습니다! 아까워라..
- `buyIngredients(PizzaStore store, Scanner input)` 의 형태가 변경되었습니다.
    - 이제, 재료를 한 개씩이 아니라 한꺼번에 여러 개 살 수 있습니다.(line 194~)

---

# Test

- 명세에서 주어진 상황은 다음과 같습니다.
    - Initialize the PizzaStore
    - Create an online order with 2 pizzas, one with all ingredients added and one with no ingredients added
    - Create an in store order with 3 pizzas, all pizzas have 1 ingredient each (nr 1: peperoni, nr 2: mushroom, nr 3: cheese). After adding all pizzas to the order, remove the first one (the peperoni pizza)
- 위 상황을 현재 버전의 PizzaStoreManager로 실행한 결과입니다.

```
PizzaStore : cash: $2.5, peperoni:3, mushrooms:3, cheeses:3.
What would you like to do:
1: Place an order, 2: buy ingredients
1
What type of order?
1: In store, 2: Online, 3: back
2
Chosen : Online
What is delivery address?
ITBT 202
What size pizza do you want?
32
Do you want peperoni on your pizza? Y/N
Y
Do you want mushrooms on your pizza? Y/N
Y
Do you want cheese on your pizza? Y/N
Y
Do you want to order another Pizza? (Y/N)
Y
What size pizza do you want?
32
Do you want peperoni on your pizza? Y/N
N
Do you want mushrooms on your pizza? Y/N
N
Do you want cheese on your pizza? Y/N
N
Do you want to order another Pizza? (Y/N)
N
Your final order is :
Online Order
Pizza[0] : [Size = 32 / Ingredients = Peperoni Mushrooms Cheese]
Pizza[1] : [Size = 32 / Ingredients = ]
Total Price : 16.6
Address : ITBT 202

Do you want to change your order? (Y/N)
N
PizzaStore : cash: $19.1, peperoni:2, mushrooms:2, cheeses:2.
What would you like to do:
1: Place an order, 2: buy ingredients
1
What type of order?
1: In store, 2: Online, 3: back
1
Chosen : In Store
What size pizza do you want?
32
Do you want peperoni on your pizza? Y/N
Y
Do you want mushrooms on your pizza? Y/N
N
Do you want cheese on your pizza? Y/N
N
Do you want to order another Pizza? (Y/N)
Y
What size pizza do you want?
```

```
32
Do you want peperoni on your pizza? Y/N
N
Do you want mushrooms on your pizza? Y/N
Y
Do you want cheese on your pizza? Y/N
N
Do you want to order another Pizza? (Y/N)
Y
What size pizza do you want?
32
Do you want peperoni on your pizza? Y/N
N
Do you want mushrooms on your pizza? Y/N
N
Do you want cheese on your pizza? Y/N
Y
Do you want to order another Pizza? (Y/N)
N
Your final order is :
InStore Order
Pizza[0] : [Size = 32 / Ingredients = Peperoni ]
Pizza[1] : [Size = 32 / Ingredients = Mushrooms ]
Pizza[2] : [Size = 32 / Ingredients = Cheese]
Total Price : 24.84
TableNumber : 1

Do you want to change your order? (Y/N)
Y
What do you want to do?
1: Add a pizza, 2: Remove a pizza, 3: Nothing
2
Which pizza do you want to remove?
0
removed Pizza[0].
What do you want to do?
1: Add a pizza, 2: Remove a pizza, 3: Nothing
3
InStore Order
Pizza[0] : [Size = 32 / Ingredients = Mushrooms ]
Pizza[1] : [Size = 32 / Ingredients = Cheese]
Total Price : 16.56
TableNumber : 1

What would you like to do:
1: Place an order, 2: buy ingredients
```

> It Works!