CECS 575 - Group 7

Food Ordering System

Assignment 4

Find **two applications of Behavioral Patterns** and implement them in Java. Create Sequence and Class diagrams for it.

Contents

Behavioral Patterns **2**

1. Iterator Pattern **2**
- 1.1 Class Diagram 2
- 1.2 Sequence Diagram 3

2. Command Pattern **3**
- 2.1 Class Diagram 4
- 2.2 Sequence Diagram 5

3. Driver Code & Console Output **6**
- 3.1 Iterator Pattern 6
- 3.2 Command Pattern 7

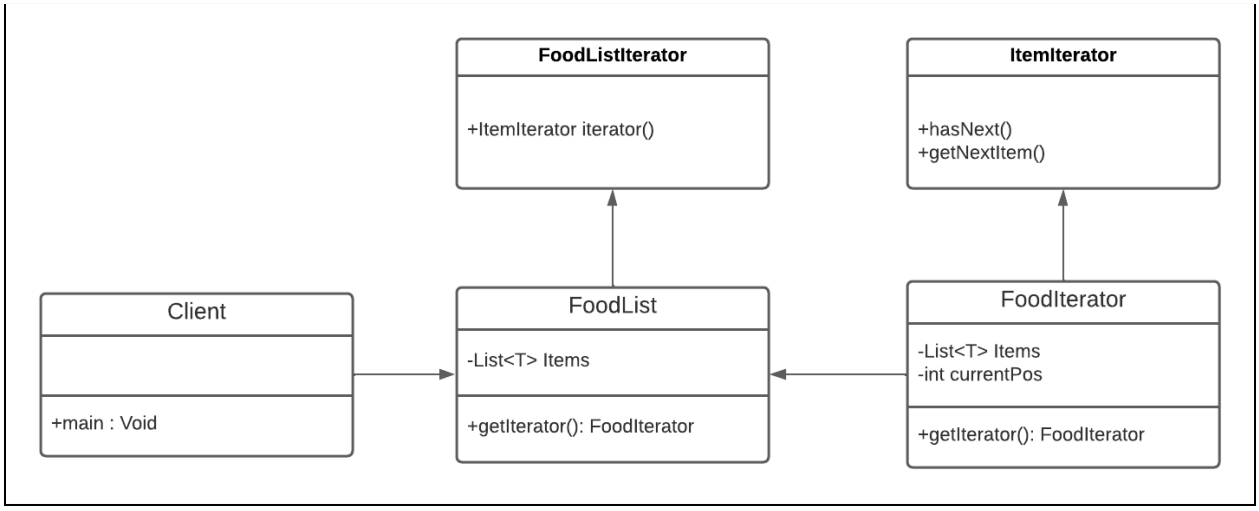
Behavioral Patterns

We have implemented 2 Behavioral Patterns:

1. Iterator Pattern

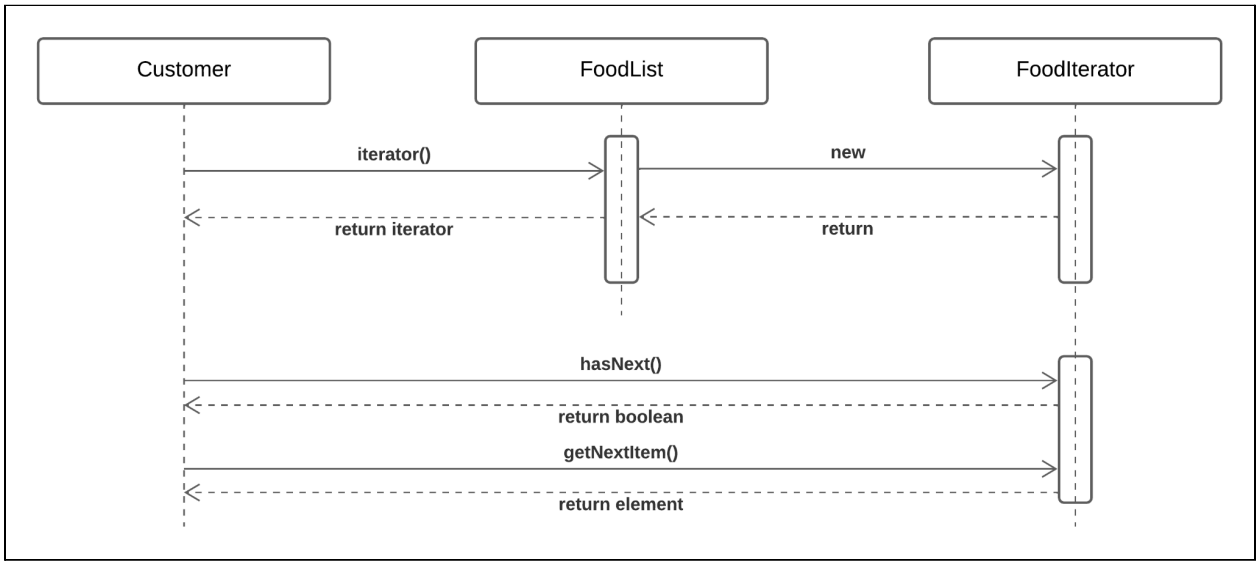
- Iterator Pattern provides a way to traverse the elements sequentially without exposing the underlying interface.
- In our system, we use an iterator pattern to iterate over **FoodCategories** in the **Menu** and **FoodItems** in the **FoodCategory**.
- We have defined an interface called **ItemIterator** which has 2 functions: hasNext**()** - which checks if a list has the next item and **getNextItem()** - which gets us the next item in the list.
- **FoodList** is a generic class that takes a list of any type **T** and returns an **Iterator**. We made a generic class as we had to iterate over multiple FoodCategories in the Menu and multiple FoodItems inside FoodCategories.
- In code, we use the while loop with condition if iterator.hasNext() items and getNextItem() to access the next iterable element.

1.1 Class Diagram



1.1: Class diagram for Iterator Pattern implementation

1.2 Sequence Diagram

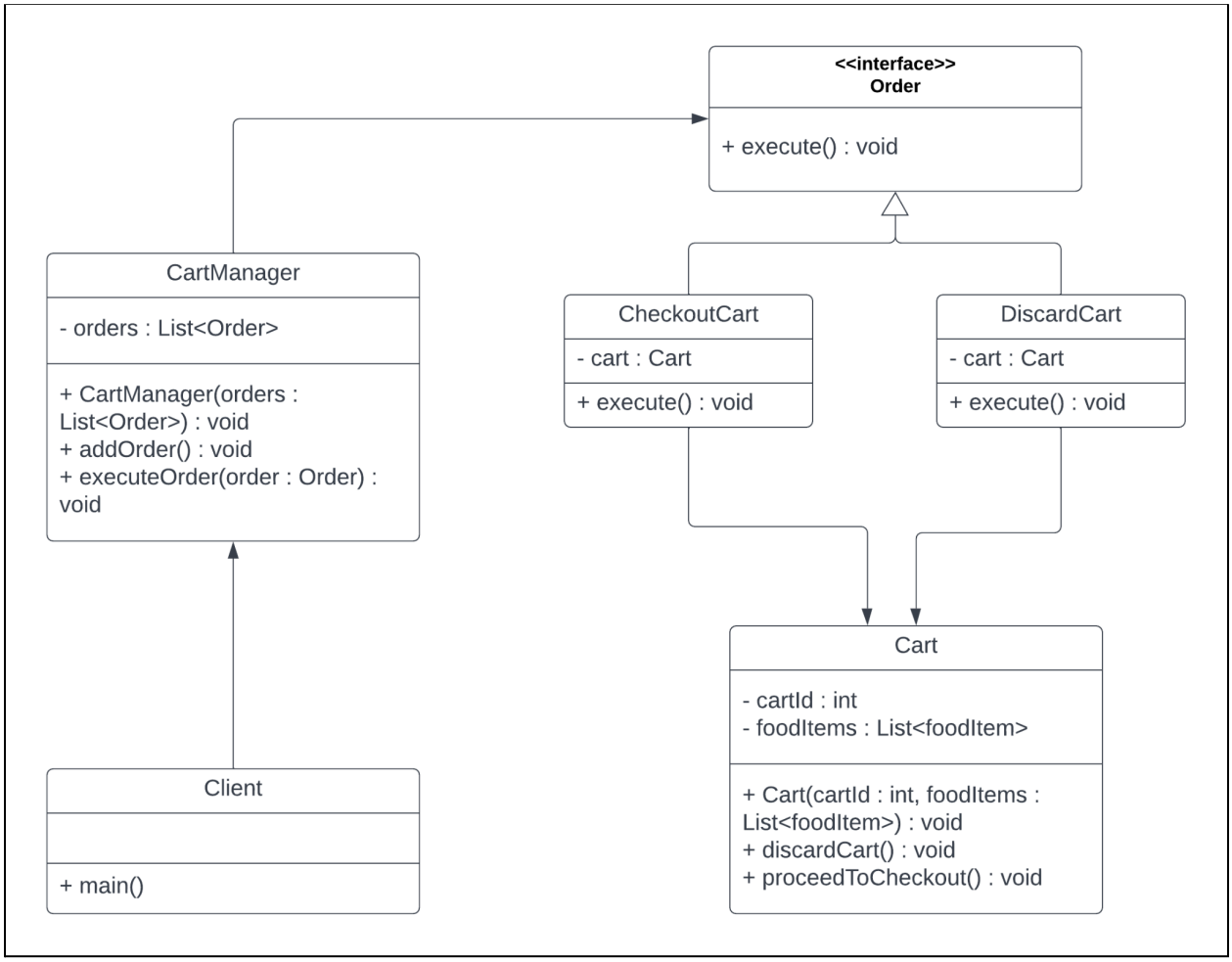


1.2: Sequence diagram for Iterator Pattern implementation

2. Command Pattern

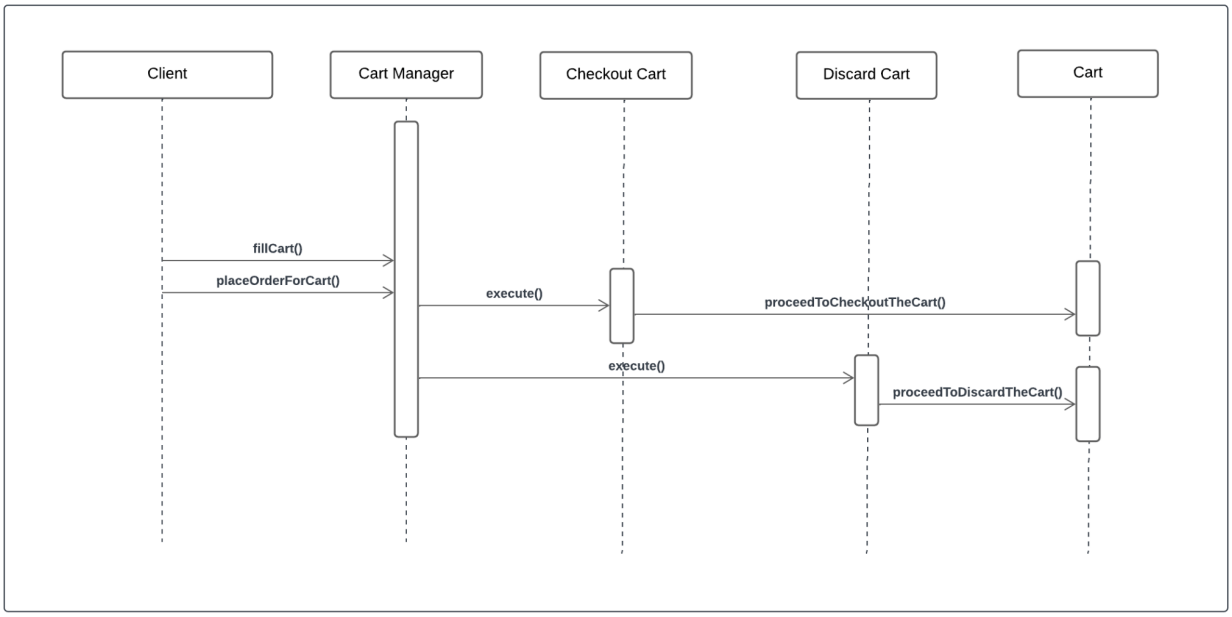
- The command pattern is a content design pattern that falls under the category of behavioral pattern.
- A request is wrapped in an object and sent as a command to the invoker object. The Invoker object looks for an object that can handle this command and passes it on to the matching object, which performs it.
- In our system, we built the command pattern for the case of the food cart.
- There is an interface **Order** that has been constructed and is implemented by the concrete command classes **DiscardCart** and **CheckoutCart**, which are built to process/display discarded and checked-out items, respectively.
- The command pattern is used by the **CartManager** to determine which object will execute which command. It has the ability to checkout and discard the cart.
- The cart manager will be used by the client class to demonstrate the command pattern.

2.1 Class Diagram



2.1: Class diagram for Command Pattern implementation

2.2 Sequence Diagram

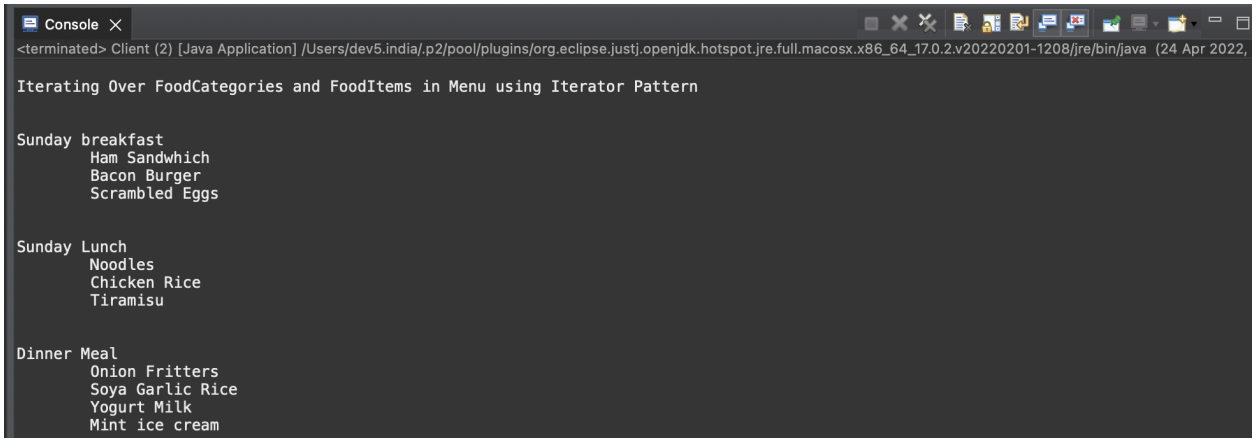


2.2: Sequence diagram for Command Pattern implementation

3. Driver Code & Console Output

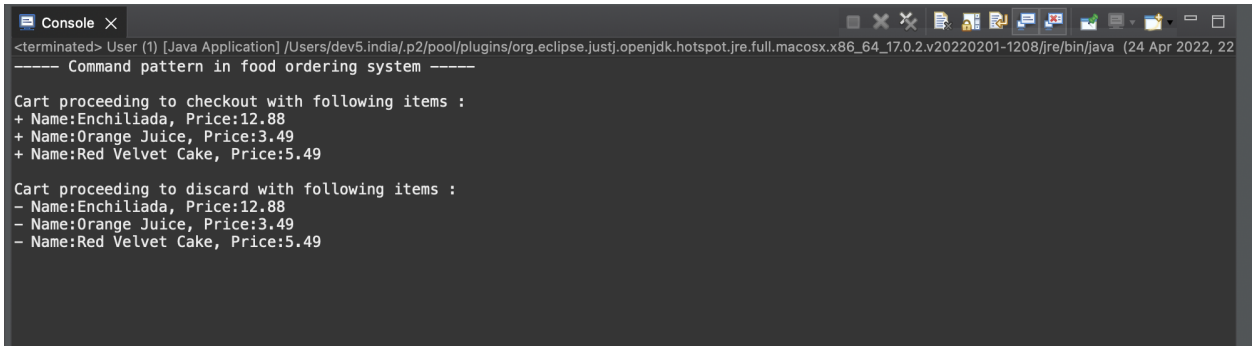
3.1 Iterator Pattern

```
public class Client {

	public static void main(String args[]) {

		List<FoodCategory> categories = new ArrayList<FoodCategory>();

		MealFactory mf = new MealFactory();
		categories.add(MealFactory.getMealType(MealType.breakfast));
		categories.add(MealFactory.getMealType(MealType.lunch));
		categories.add(MealFactory.getMealType(MealType.dinner));

		System.out.println("\nIterating Over FoodCategories and FoodItems in Menu using Iterator
Pattern");


		FoodList cat = new FoodList<FoodCategory>(categories);
		FoodIterator fi = cat.iterator();

		while(fi.hasNext()) {

			FoodCategory currentFoodCategory = (FoodCategory)fi.getNextItem();

			System.out.println("\n");
			System.out.println(currentFoodCategory.getName());

			FoodList foodItem = new FoodList(currentFoodCategory.getFoodItems());
			FoodIterator fItemIterator = foodItem.iterator();



			while(fItemIterator.hasNext()) {

				FoodItem currentFoodItem = (FoodItem)fItemIterator.getNextItem();



				System.out.println("\t" + currentFoodItem.getName());
			}

		}

	}
}
```



3.2 Command Pattern

```
public class User {
   public static void main(String arg[]){

       System.out.println("----- Command pattern in food ordering system -----");
       CreatorPattern.FoodItem cf1 = new CreatorPattern.FoodItem(Integer.valueOf("46"),"Enchiliada","Main
Dish",Double.valueOf("12.88"),"Lunch");
       CreatorPattern.FoodItem cf2 = new CreatorPattern.FoodItem(Integer.valueOf("23"),"Orange
Juice","Drinks",Double.valueOf("3.49"),"Lunch");
       CreatorPattern.FoodItem cf3 = new CreatorPattern.FoodItem(Integer.valueOf("18"),"Red Velvet
Cake","Dessert",Double.valueOf("5.49"),"Lunch");
       List<CreatorPattern.FoodItem> list  = new ArrayList<CreatorPattern.FoodItem>();
       list.add(cf1);
       list.add(cf2);
       list.add(cf3);
       Cart newCart = new Cart(list);
       CheckoutCart checkoutCart = new CheckoutCart(newCart);
       DiscardCart discardCart = new DiscardCart(newCart);
       CartManager cm = new CartManager();
       cm.fillCart(checkoutCart);
       cm.fillCart(discardCart);

       cm.placeOrderForCart();

   }
}
```


Console
<terminated> User (1) [Java Application] /Users/dev5.india/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.2.v20220201-1208/jre/bin/java (24 Apr 2022, 22
----- Command pattern in food ordering system -----
Cart proceeding to checkout with following items :
+ Name:Enchiliada, Price:12.88
+ Name:Orange Juice, Price:3.49
+ Name:Red Velvet Cake, Price:5.49
Cart proceeding to discard with following items :
- Name:Enchiliada, Price:12.88
- Name:Orange Juice, Price:3.49
- Name:Red Velvet Cake, Price:5.49