Github link: https://github.com/VietNguyen705/CS5800

Part 1 — Builder Design Pattern (Pizza)

File: Pizza.java

```java
package pizza;

import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

public class Pizza {
    private final String chainName;
    private final PizzaSize size;
    private final List<Topping> toppings;

    Pizza(String chainName, PizzaSize size, List<Topping> toppings) {
        this.chainName = chainName;
        this.size = size;
        this.toppings = List.copyOf(toppings);
    }

    public String getChainName() {
        return chainName;
    }

    public PizzaSize getSize() {
        return size;
    }

    public List<Topping> getToppings() {
        return Collections.unmodifiableList(toppings);
    }

    public void eat() {
        String toppingsDescription = toppings.isEmpty()
            ? "No Toppings"
            : toppings.stream()
                .map(Topping::getDisplayName)
                .collect(Collectors.joining(", "));
        System.out.printf("%s %s pizza with toppings: %s%n",
            chainName,
            size.getDisplayName(),
            toppingsDescription);
    }
}
```

File: PizzaBuilder.java

```
package pizza;

public interface PizzaBuilder {
    PizzaBuilder setSize(PizzaSize size);

    PizzaBuilder addTopping(Topping topping);

    PizzaBuilder addToppings(Topping... toppings);

    Pizza build();
}
```

File: BasePizzaBuilder.java

```
package pizza;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public abstract class BasePizzaBuilder implements PizzaBuilder {
    private final String chainName;
    private PizzaSize size;
    private final List<Topping> toppings = new ArrayList<>();

    protected BasePizzaBuilder(String chainName) {
        this.chainName = Objects.requireNonNull(chainName, "chainName");
    }

    @Override
    public PizzaBuilder setSize(PizzaSize size) {
        this.size = Objects.requireNonNull(size, "size");
        return this;
    }

    @Override
    public PizzaBuilder addTopping(Topping topping) {
        toppings.add(Objects.requireNonNull(topping, "topping"));
        return this;
    }
    @Override
    public PizzaBuilder addToppings(Topping... toppings) {
        Objects.requireNonNull(toppings, "toppings");
        for (Topping topping : toppings) {
            addTopping(topping);
        }
        return this;
    }

    @Override
    public Pizza build() {
        if (size == null) {
            throw new IllegalStateException("Pizza size must be set before building.");
        }
        return new Pizza(chainName, size, toppings);
    }
}
```

File: `PizzaHutBuilder.java`

```
package pizza;

public class PizzaHutBuilder extends BasePizzaBuilder {
    public PizzaHutBuilder() {
        super("Pizza Hut");
    }
}
```

File: `LittleCaesarsBuilder.java`

```
package pizza;

public class LittleCaesarsBuilder extends BasePizzaBuilder {
    public LittleCaesarsBuilder() {
        super("Little Caesars");
    }
}
```

File: `DominosBuilder.java`

```
package pizza;

public class DominosBuilder extends BasePizzaBuilder {
    public DominosBuilder() {
        super("Dominos");
    }
}
```

File: PizzaSize.java

```
package pizza;

public enum PizzaSize {
    SMALL("Small"),
    MEDIUM("Medium"),
    LARGE("Large");

    private final String displayName;

    PizzaSize(String displayName) {
        this.displayName = displayName;
    }

    public String getDisplayName() {
        return displayName;
    }
}
```

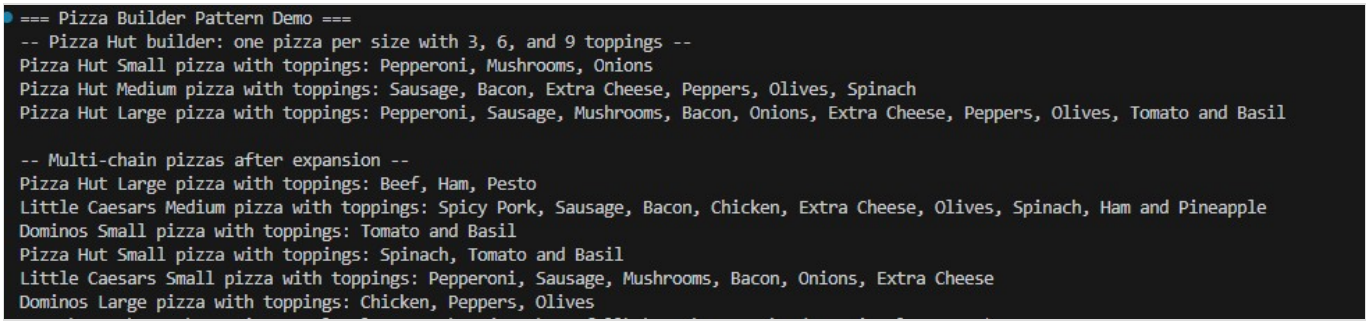
File: Topping.java

```
package pizza;

public enum Topping {
    PEPPERONI("Pepperoni"),
    SAUSAGE("Sausage"),
    MUSHROOMS("Mushrooms"),
    BACON("Bacon"),
    ONIONS("Onions"),
    EXTRA_CHEESE("Extra Cheese"),
    PEPPERS("Peppers"),
    CHICKEN("Chicken"),
    OLIVES("Olives"),
    SPINACH("Spinach"),
    TOMATO_AND_BASIL("Tomato and Basil"),
    BEEF("Beef"),
    HAM("Ham"),
    PESTO("Pesto"),
    SPICY_PORK("Spicy Pork"),
    HAM_AND_PINEAPPLE("Ham and Pineapple");

    private final String displayName;
    Topping(String displayName) {
        this.displayName = displayName;
    }

    public String getDisplayName() {
        return displayName;
    }
}
```

File: PizzaDriver.java

```
package pizza;

import java.util.List;

public class PizzaDriver {
    public static void main(String[] args) {
        System.out.println("=== Pizza Builder Pattern Demo ===");
        System.out.println("-- Pizza Hut builder: one pizza per size with 3, 6, and 9 toppings --");

        List<Pizza> pizzaHutShowcase = List.of(
            createPizza(new PizzaHutBuilder(), PizzaSize.SMALL,
                Topping.PEPPERONI, Topping.MUSHROOMS, Topping.ONIONS),
            createPizza(new PizzaHutBuilder(), PizzaSize.MEDIUM,
                Topping.SAUSAGE, Topping.BACON, Topping.EXTRA_CHEESE,
                Topping.PEPPERS, Topping.OLIVES, Topping.SPINACH),
            createPizza(new PizzaHutBuilder(), PizzaSize.LARGE,
                Topping.PEPPERONI, Topping.SAUSAGE, Topping.MUSHROOMS,
                Topping.BACON, Topping.ONIONS, Topping.EXTRA_CHEESE,
                Topping.PEPPERS, Topping.OLIVES, Topping.TOMATO_AND_BASIL)
        );
        pizzaHutShowcase.forEach(Pizza::eat);

        System.out.println();
        System.out.println("-- Multi-chain pizzas after expansion --");
        List<Pizza> chainPizzas = List.of(
            createPizza(new PizzaHutBuilder(), PizzaSize.LARGE,
                Topping.BEEF, Topping.HAM, Topping.PESTO),
            createPizza(new LittleCaesarsBuilder(), PizzaSize.MEDIUM,
                Topping.SPICY_PORK, Topping.SAUSAGE, Topping.BACON,
                Topping.CHICKEN, Topping.EXTRA_CHEESE, Topping.OLIVES,
                Topping.SPINACH, Topping.HAM_AND_PINEAPPLE),
            createPizza(new DominosBuilder(), PizzaSize.SMALL,
                Topping.TOMATO_AND_BASIL),
            createPizza(new PizzaHutBuilder(), PizzaSize.SMALL,
                Topping.SPINACH, Topping.TOMATO_AND_BASIL),
            createPizza(new LittleCaesarsBuilder(), PizzaSize.SMALL,
                Topping.PEPPERONI, Topping.SAUSAGE, Topping.MUSHROOMS,
                Topping.BACON, Topping.ONIONS, Topping.EXTRA_CHEESE),
            createPizza(new DominosBuilder(), PizzaSize.LARGE,
                Topping.CHICKEN, Topping.PEPPERS, Topping.OLIVES)
        );
        chainPizzas.forEach(Pizza::eat);
    }

    private static Pizza createPizza(PizzaBuilder builder, PizzaSize size, Topping... toppings) {
        builder.setSize(size);
        if (toppings != null && toppings.length > 0) {
            builder.addToppings(toppings);
        }
        return builder.build();
    }
}
```

Output Screenshot:



```
=== Pizza Builder Pattern Demo ===
-- Pizza Hut builder: one pizza per size with 3, 6, and 9 toppings --
Pizza Hut Small pizza with toppings: Pepperoni, Mushrooms, Onions
Pizza Hut Medium pizza with toppings: Sausage, Bacon, Extra Cheese, Peppers, Olives, Spinach
Pizza Hut Large pizza with toppings: Pepperoni, Sausage, Mushrooms, Bacon, Onions, Extra Cheese, Peppers, Olives, Tomato and Basil

-- Multi-chain pizzas after expansion --
Pizza Hut Large pizza with toppings: Beef, Ham, Pesto
Little Caesars Medium pizza with toppings: Spicy Pork, Sausage, Bacon, Chicken, Extra Cheese, Olives, Spinach, Ham and Pineapple
Dominos Small pizza with toppings: Tomato and Basil
Pizza Hut Small pizza with toppings: Spinach, Tomato and Basil
Little Caesars Small pizza with toppings: Pepperoni, Sausage, Mushrooms, Bacon, Onions, Extra Cheese
Dominos Large pizza with toppings: Chicken, Peppers, Olives
```

Part 2 — Abstract Factory, Factory and Singleton (Macronutrient Meals)

File: MacronutrientAbstractFactory.java

```
package macros;

public interface MacronutrientAbstractFactory {
    FoodFactory carbFactory();
    FoodFactory proteinFactory();
    FoodFactory fatFactory();
}
```

File: MacronutrientFactoryProvider.java

```
package macros;

public final class MacronutrientFactoryProvider implements MacronutrientAbstractFactory {
    private static final MacronutrientFactoryProvider INSTANCE = new MacronutrientFactoryProvider();

    private MacronutrientFactoryProvider() {
    }

    public static MacronutrientFactoryProvider getInstance() {
        return INSTANCE;
    }

    @Override
    public FoodFactory carbFactory() {
        return CarbFactory.getInstance();
    }

    @Override
    public FoodFactory proteinFactory() {
        return ProteinFactory.getInstance();
    }

    @Override
    public FoodFactory fatFactory() {
        return FatFactory.getInstance();
    }
}
```

File: FoodFactory.java

```
package macros;

public interface FoodFactory {
    FoodItem create(DietPlan dietPlan);
}
```

File: CarbFactory.java

```
package macros;

import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.stream.Collectors;
```

```java
public final class CarbFactory implements FoodFactory {
    private static final CarbFactory INSTANCE = new CarbFactory();
    private static final List<FoodItem> CARB_ITEMS = List.of(
        FoodItem.BREAD,
        FoodItem.PISTACHIO
    );

    private CarbFactory() {
    }

    public static CarbFactory getInstance() {
        return INSTANCE;
    }

    @Override
    public FoodItem create(DietPlan dietPlan) {
        List<FoodItem> allowedItems = CARB_ITEMS.stream()
            .filter(dietPlan::allows)
            .collect(Collectors.toList());
        if (allowedItems.isEmpty()) {
            throw new IllegalStateException("No carb options available for diet plan " + dietPlan);
        }
        int index = ThreadLocalRandom.current().nextInt(allowedItems.size());
        return allowedItems.get(index);
    }
}
```

File: ProteinFactory.java

```java
package macros;

import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.stream.Collectors;

public final class ProteinFactory implements FoodFactory {
    private static final ProteinFactory INSTANCE = new ProteinFactory();
    private static final List<FoodItem> PROTEIN_ITEMS = List.of(
        FoodItem.CHICKEN,
        FoodItem.BEEF,
        FoodItem.FISH,
        FoodItem.TUNA,
        FoodItem.TOFU,
        FoodItem.LENTILS
    );

    private ProteinFactory() {
    }

    public static ProteinFactory getInstance() {
        return INSTANCE;
    }

    @Override
    public FoodItem create(DietPlan dietPlan) {
        List<FoodItem> allowedItems = PROTEIN_ITEMS.stream()
            .filter(dietPlan::allows)
            .collect(Collectors.toList());
        if (allowedItems.isEmpty()) {
            throw new IllegalStateException("No protein options available for diet plan " + dietPlan)
        }
        int index = ThreadLocalRandom.current().nextInt(allowedItems.size());
        return allowedItems.get(index);
    }
}
```

File: FatFactory.java

```
package macros;

import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.stream.Collectors;

public final class FatFactory implements FoodFactory {
    private static final FatFactory INSTANCE = new FatFactory();
    private static final List<FoodItem> FAT_ITEMS = List.of(
        FoodItem.CHEESE,
        FoodItem.SOUR_CREAM,
        FoodItem.AVOCADO,
        FoodItem.PEANUTS
    );

    private FatFactory() {
    }

    public static FatFactory getInstance() {
        return INSTANCE;
    }

    @Override
    public FoodItem create(DietPlan dietPlan) {
        List<FoodItem> allowedItems = FAT_ITEMS.stream()
            .filter(dietPlan::allows)
            .collect(Collectors.toList());
        if (allowedItems.isEmpty()) {
            throw new IllegalStateException("No fat options available for diet plan " + dietPlan);
        }
        int index = ThreadLocalRandom.current().nextInt(allowedItems.size());
        return allowedItems.get(index);
    }
}
```

File: FoodItem.java

```
package macros;

public enum FoodItem {
    BREAD(FoodCategory.CARB, "Bread", false, false, false, false),
    PISTACHIO(FoodCategory.CARB, "Pistachio", false, false, true, false),
    CHICKEN(FoodCategory.PROTEIN, "Chicken", false, true, false, false),
    BEEF(FoodCategory.PROTEIN, "Beef", false, true, false, false),
    FISH(FoodCategory.PROTEIN, "Fish", false, true, false, false),
    TUNA(FoodCategory.PROTEIN, "Tuna", false, true, false, false),
    TOFU(FoodCategory.PROTEIN, "Tofu", false, false, false, true),
    LENTILS(FoodCategory.PROTEIN, "Lentils", false, false, false, false),
    CHEESE(FoodCategory.FAT, "Cheese", true, false, false, false),
    SOUR_CREAM(FoodCategory.FAT, "Sour cream", true, false, false, false),
    AVOCADO(FoodCategory.FAT, "Avocado", false, false, false, false),
    PEANUTS(FoodCategory.FAT, "Peanuts", false, false, true, false);

    private final FoodCategory category;
    private final String displayName;
    private final boolean dairy;
    private final boolean meat;
    private final boolean nut;
    private final boolean tofu;

    FoodItem(FoodCategory category, String displayName, boolean dairy, boolean meat, boolean nut, boo
        this.category = category;
        this.displayName = displayName;
        this.dairy = dairy;
        this.meat = meat;
        this.nut = nut;
        this.tofu = tofu;
    }

    public FoodCategory getCategory() {
        return category;
    }

    public String getDisplayName() {
        return displayName;
    }

    public boolean isDairy() {
        return dairy;
    }

    public boolean isMeat() {
        return meat;
    }

    public boolean isNut() {
        return nut;
    }

    public boolean isTofu() {
        return tofu;
    }
}
```

File: FoodCategory.java

```
package macros;

public enum FoodCategory {
    CARB("Carb"),
    PROTEIN("Protein"),
    FAT("Fat");

    private final String displayName;

    FoodCategory(String displayName) {
        this.displayName = displayName;
    }

    public String getDisplayName() {
        return displayName;
    }
}
```

File: DietPlan.java

```
package macros;

public enum DietPlan {
    NO_RESTRICTION("No Restriction") {
        @Override
```

```java
        public boolean allows(FoodItem item) {
            return true;
        }
    },
    PALEO("Paleo") {
        @Override
        public boolean allows(FoodItem item) {
            if (item.getCategory() == FoodCategory.CARB && item != FoodItem.PISTACHIO) {
                return false;
            }
            if (item.isTofu()) {
                return false;
            }
            return !item.isDairy();
        }
    },
    VEGAN("Vegan") {
        @Override
        public boolean allows(FoodItem item) {
            if (item.isMeat()) {
                return false;
            }
            return !item.isDairy();
        }
    },
    NUT_ALLERGY("Nut Allergy") {
        @Override
        public boolean allows(FoodItem item) {
            return !item.isNut();
        }
    };

    private final String displayName;

    DietPlan(String displayName) {
        this.displayName = displayName;
    }

    public String getDisplayName() {
        return displayName;
    }

    public abstract boolean allows(FoodItem item);
}
```

File: Customer.java

```java
package macros;
```

```
import java.util.Objects;

public class Customer {
    private final String name;
    private final DietPlan dietPlan;

    public Customer(String name, DietPlan dietPlan) {
        this.name = Objects.requireNonNull(name, "name");
        this.dietPlan = Objects.requireNonNull(dietPlan, "dietPlan");
    }

    public String getName() {
        return name;
    }

    public DietPlan getDietPlan() {
        return dietPlan;
    }
}
```

File: Meal.java

```
package macros;

import java.util.Objects;

public class Meal {
    private final FoodItem carb;
    private final FoodItem protein;
    private final FoodItem fat;

    public Meal(FoodItem carb, FoodItem protein, FoodItem fat) {
        this.carb = Objects.requireNonNull(carb, "carb");
        this.protein = Objects.requireNonNull(protein, "protein");
        this.fat = Objects.requireNonNull(fat, "fat");
    }

    public FoodItem getCarb() {
        return carb;
    }

    public FoodItem getProtein() {
        return protein;
    }

    public FoodItem getFat() {
        return fat;
```

```java
    }

    public String describe() {
        return String.format("Carb: %s, Protein: %s, Fat: %s",
            carb.getDisplayName(),
            protein.getDisplayName(),
            fat.getDisplayName());
    }
}
```

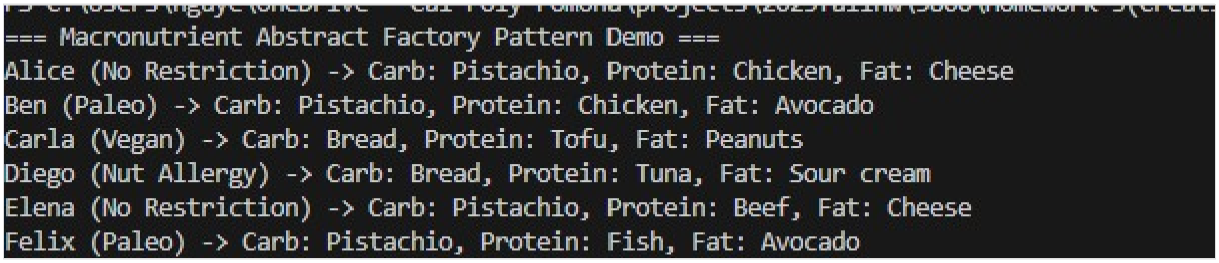
File: MealService.java

```java
package macros;

import java.util.Objects;

public class MealService {
    private final MacronutrientAbstractFactory factory;

    public MealService(MacronutrientAbstractFactory factory) {
        this.factory = Objects.requireNonNull(factory, "factory");
    }

    public Meal createMeal(Customer customer) {
        Objects.requireNonNull(customer, "customer");
        DietPlan dietPlan = customer.getDietPlan();
        FoodItem carb = factory.carbFactory().create(dietPlan);
        FoodItem protein = factory.proteinFactory().create(dietPlan);
        FoodItem fat = factory.fatFactory().create(dietPlan);
        return new Meal(carb, protein, fat);
    }
}
```

File: MacronutrientDriver.java

```java
package macros;

import java.util.List;

public class MacronutrientDriver {
    public static void main(String[] args) {
        System.out.println("=== Macronutrient Abstract Factory Pattern Demo ===");

        MacronutrientAbstractFactory factory = MacronutrientFactoryProvider.getInstance();
        MealService mealService = new MealService(factory);

        List<Customer> customers = List.of(
            new Customer("Alice", DietPlan.NO_RESTRICTION),
            new Customer("Ben", DietPlan.PALEO),
            new Customer("Carla", DietPlan.VEGAN),
            new Customer("Diego", DietPlan.NUT_ALLERGY),
            new Customer("Elena", DietPlan.NO_RESTRICTION),
            new Customer("Felix", DietPlan.PALEO)
        );

        for (Customer customer : customers) {
            Meal meal = mealService.createMeal(customer);
            System.out.printf("%s (%s) -> %s%n",
                customer.getName(),
                customer.getDietPlan().getDisplayName(),
                meal.describe());
        }
    }
}
```

Output Screenshot:



```
=== Macronutrient Abstract Factory Pattern Demo ===
Alice (No Restriction) -> Carb: Pistachio, Protein: Chicken, Fat: Cheese
Ben (Paleo) -> Carb: Pistachio, Protein: Chicken, Fat: Avocado
Carla (Vegan) -> Carb: Bread, Protein: Tofu, Fat: Peanuts
Diego (Nut Allergy) -> Carb: Bread, Protein: Tuna, Fat: Sour cream
Elena (No Restriction) -> Carb: Pistachio, Protein: Beef, Fat: Cheese
Felix (Paleo) -> Carb: Pistachio, Protein: Fish, Fat: Avocado
```