Kevin Wong
CS4800.01
Professor Davarpanah
March 13, 2024

Github repo link: https://github.com/kdub8/CS4800-HW4-Creational-Patterns.git

Pizza (Builder) code:

```
import java.util.*;

/**
 * Enum representing various toppings for a pizza.
 */
enum Topping {
    PEPPERONI, SAUSAGE, MUSHROOMS, BACON, ONIONS,
EXTRA_CHEESE, PEPPERS, CHICKEN, OLIVES,
    SPINACH, TOMATO_AND_BASIL, BEEF, HAM, PESTO,
SPICY_PORK, HAM_AND_PINEAPPLE
}

/**
 * Class representing a pizza.
 */
class Pizza {
    private final String chain;
    private final String size;
    private final List<Topping> toppings;

    /**
     * Constructor for Pizza class.
```

```
     *
     * @param builder The builder object used to
construct the pizza.
     */
    protected Pizza(Builder builder) {
        this.chain = builder.chain;
        this.size = builder.size;
        this.toppings = builder.toppings;
    }

    /**
     * Method to eat the pizza.
     */
    public void eat() {
        System.out.println("Eating " + chain + " "
+ size + " pizza with toppings: " + toppings);
        System.out.println();
    }

    /**
     * Builder class for constructing Pizza
objects.
     */
    public static class Builder {
        private final String chain;
        private final String size;
        private final List<Topping> toppings = new
ArrayList<>();
```

```
        /**
         * Constructor for Builder class.
         *
         * @param chain The chain of the pizza
restaurant.
         * @param size  The size of the pizza.
         */
        public Builder(String chain, String size) {
            this.chain = chain;
            this.size = size;
        }

        /**
         * Method to add a topping to the pizza.
         *
         * @param topping The topping to add.
         * @return The Builder object for method
chaining.
         */
        public Builder addTopping(Topping topping)
{
            toppings.add(topping);
            return this;
        }

        /**
         * Method to build the Pizza object.
```

```
         *
         * @return The constructed Pizza object.
         * @throws IllegalStateException If the
chain or size is null or an empty
         *                                  string.
         */
        public Pizza build() {
            if (chain == null || chain == "") {
                throw new
IllegalStateException("Pizza chain is required!");
            }
            if (size == null || size == "") {
                throw new
IllegalStateException("Pizza size is required!");
            }
            return new Pizza(this);
        }
    }
}

/**
 * Class representing a Little Caesars pizza.
 */
class LittleCaesars extends Pizza {
    public LittleCaesars(Builder builder) {
        super(builder);
    }
}
```

```java
/**
 * Class representing a Dominos pizza.
 */
class Dominos extends Pizza {
    public Dominos(Builder builder) {
        super(builder);
    }
}

/**
 * Driver program to create and eat pizzas.
 */
public class Main {
    public static void main(String[] args) {

        // ANSWERS FOR QUESTION 1
        /*
         * Create a driver program to create three pizzas one of each size with 3, 6,
         * and 9 toppings to your
         * liking and eat() all of them
         */

        System.out.println("Pizzas for question 1:");
        System.out.println();
```

```
        // 3 toppings
        Pizza hutSmall = new Pizza.Builder("Pizza
Hut", "Small")
                .addTopping(Topping.PEPPERONI)
                .addTopping(Topping.SAUSAGE)
                .addTopping(Topping.MUSHROOMS)
                .build();
        hutSmall.eat();

        // 6 toppings
        Pizza hutMedium = new Pizza.Builder("Pizza
Hut", "Medium")
                .addTopping(Topping.PEPPERONI)
                .addTopping(Topping.SAUSAGE)
                .addTopping(Topping.MUSHROOMS)
                .addTopping(Topping.ONIONS)
                .addTopping(Topping.OLIVES)
                .addTopping(Topping.PEPPERS)
                .build();
        hutMedium.eat();

        // 9 toppings
        Pizza hutLarge = new Pizza.Builder("Pizza
Hut", "Large")

.addTopping(Topping.TOMATO_AND_BASIL)
                .addTopping(Topping.BEEF)
                .addTopping(Topping.ONIONS)
```

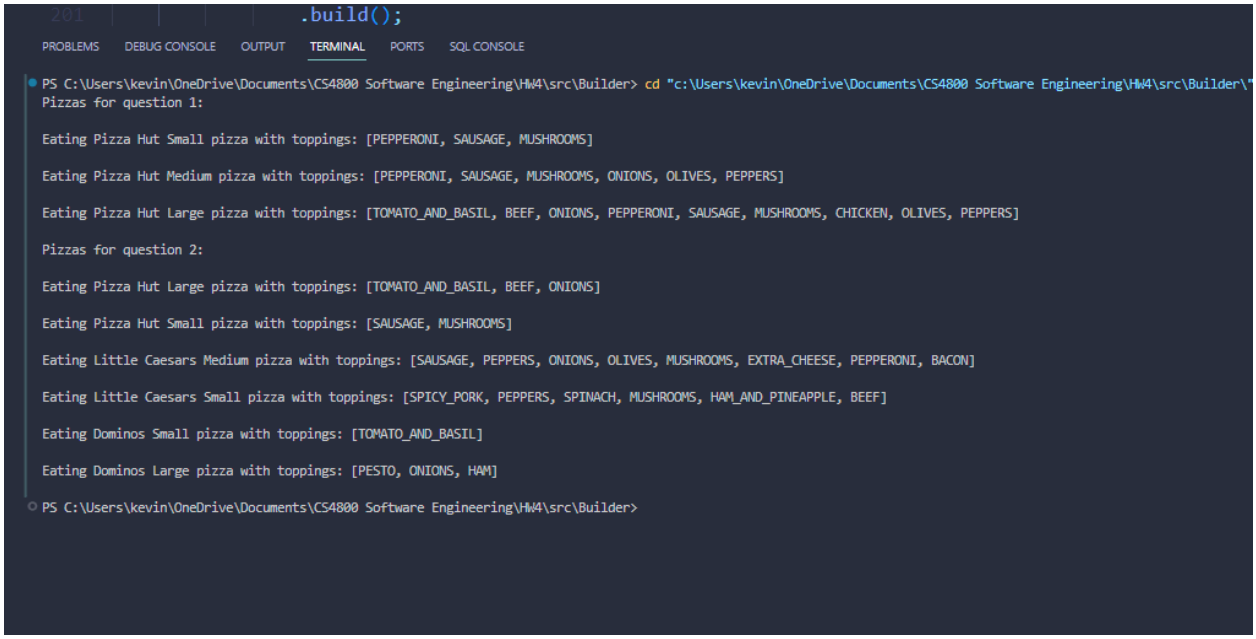
```
                .addTopping(Topping.PEPPERONI)
                .addTopping(Topping.SAUSAGE)
                .addTopping(Topping.MUSHROOMS)
                .addTopping(Topping.CHICKEN)
                .addTopping(Topping.OLIVES)
                .addTopping(Topping.PEPPERS)
                .build();
        hutLarge.eat();

        // ANSWERS FOR QUESTION 2
        /*
         * Now assume you purchased another two
pizza chains, Little Caesars, and
         * Dominos. You want to
         * add them to your program following the
rules mentioned above.
         * Create the following pizzas and eat()
all of them:
         *
         * Pizza Hut:
         * Large pizza with 3 toppings
         * Small pizza with 2 toppings
         *
         * Little Caesars:
         * Medium pizza with 8 toppings
         * Small pizza with 6 toppings
         *
         * Dominos:
```

```
         * Small pizza with 1 topping
         * Large pizza with 3 toppings
         */

        System.out.println("Pizzas for question
2:");
        System.out.println();

        // Create Pizza Hut pizzas
        Pizza hutLarge2 = new Pizza.Builder("Pizza
Hut", "Large")

.addTopping(Topping.TOMATO_AND_BASIL)
                .addTopping(Topping.BEEF)
                .addTopping(Topping.ONIONS)
                .build();
        hutLarge2.eat();

        Pizza hutSmall2 = new Pizza.Builder("Pizza
Hut", "Small")
                .addTopping(Topping.SAUSAGE)
                .addTopping(Topping.MUSHROOMS)
                .build();
        hutSmall2.eat();

        // Create Little Caesar's pizzas
        Pizza lcMedium = new Pizza.Builder("Little
Caesars", "Medium")
```

```
                .addTopping(Topping.SAUSAGE)
                .addTopping(Topping.PEPPERS)
                .addTopping(Topping.ONIONS)
                .addTopping(Topping.OLIVES)
                .addTopping(Topping.MUSHROOMS)
                .addTopping(Topping.EXTRA_CHEESE)
                .addTopping(Topping.PEPPERONI)
                .addTopping(Topping.BACON)
                .build();
        lcMedium.eat();

        Pizza lcSmall = new Pizza.Builder("Little
Caesars", "Small")
                .addTopping(Topping.SPICY_PORK)
                .addTopping(Topping.PEPPERS)
                .addTopping(Topping.SPINACH)
                .addTopping(Topping.MUSHROOMS)

.addTopping(Topping.HAM_AND_PINEAPPLE)
                .addTopping(Topping.BEEF)
                .build();
        lcSmall.eat();

        // Create Dominos pizzas
        Pizza dominosSmall = new
Pizza.Builder("Dominos", "Small")

.addTopping(Topping.TOMATO_AND_BASIL)
```

```
                .build();
        dominosSmall.eat();

        Pizza dominosLarge = new
Pizza.Builder("Dominos", "Large")
                .addTopping(Topping.PESTO)
                .addTopping(Topping.ONIONS)
                .addTopping(Topping.HAM)
                .build();
        dominosLarge.eat();
    }
}
```

Output:



Factory, Abstract Factory, and Singleton Macronutrients

Main.java

```
import java.util.ArrayList;

/**
 * Main class that demonstrates creating and
displaying customer meals.
 */

public class Main {
    /**
     * Main method that creates and displays meals
for different customers.
     */
    public static void main(String[] args) {
        makeAndDisplayCustomerMeal(new
Customer("Deby Lee", DietPlan.NONE));
        makeAndDisplayCustomerMeal(new
Customer("Jalen Tom", DietPlan.PALEO));
        makeAndDisplayCustomerMeal(new
Customer("Kevin Wong", DietPlan.NUT_ALLERGY));
        makeAndDisplayCustomerMeal(new
Customer("Danzel Yap", DietPlan.VEGAN));
        makeAndDisplayCustomerMeal(new
Customer("Joshua Casuga", DietPlan.NONE));
        makeAndDisplayCustomerMeal(new
Customer("Koki Yamaguchi", DietPlan.NUT_ALLERGY));
```

```
    }

    /**
     * Creates and displays a meal for a given
customer.
     *
     * @param customer The customer for whom the
meal is created.
     */
    public static void
makeAndDisplayCustomerMeal(Customer customer) {
        MacronutrientFactory[]
macronutrientFactories = {
CarbFactory.getFactory(),
ProteinFactory.getFactory(),
                FatFactory.getFactory() };
        Macronutrient[] meal = createMeal(customer,
macronutrientFactories);

System.out.println(customer.getDescription());
        displayMeal(meal);
        System.out.println();

    }

    /**
     * Creates a meal for a given customer using
the specified macronutrient
```

```
     * factories.
     *
     * @param customer                The customer
for whom the meal is created.
     * @param macronutrientFactories The array of
macronutrient factories to use.
     * @return An array of macronutrients
representing the meal.
     */
    public static Macronutrient[]
createMeal(Customer customer,
MacronutrientFactory[] macronutrientFactories) {
        ArrayList<Macronutrient> macronutrients =
new ArrayList<>();
        for (MacronutrientFactory
macronutrientFactory : macronutrientFactories) {
            Macronutrient macronutrient =
macronutrientFactory.getMacronutrient(customer.getD
ietPlan());
            macronutrients.add(macronutrient);
        }

        return macronutrients.toArray(new
Macronutrient[macronutrients.size()]);
    }

    /**
```

```
     * Displays the meal plan, listing each
macronutrient in the meal.
     *
     * @param macronutrients An array of
macronutrients representing the meal.
     */
    public static void displayMeal(Macronutrient[]
macronutrients) {
        System.out.println("Meal Plan");
        System.out.println("|--|--|--|--|");
        for (Macronutrient macronutrient :
macronutrients) {

System.out.println(macronutrient.getDescription());
        }
    }
}
```

Carb.java

```
/**
 * Represents a carbohydrate macronutrient.
 */
public class Carb extends Macronutrient {

    /**
     * Constructs a new Carb instance with the
given food item.
     *
```

```
     * @param foodItem the food item representing
the carbohydrate
     */
    public Carb(String foodItem) {
        super(foodItem);
    }

    /**
     * Returns the description of the carbohydrate.
     *
     * @return the description of the carbohydrate
     */
    @Override
    public String getDescription() {
        return "Carb: " + foodItem;
    }
}
```

CarbFactory.java

```
/**
 * A factory class for creating instances of the
{@link Carb} class, which
 * represents carbohydrates.
 */
public class CarbFactory extends
MacronutrientFactory {
    private static CarbFactory factory = null;
```

```
    private CarbFactory() {
    }

    /**
     * Returns the singleton instance of the {@code
CarbFactory}.
     *
     * @return the singleton instance of the {@code
CarbFactory}
     */

    public static CarbFactory getFactory() {
        if (factory == null) {
            factory = new CarbFactory();
        }

        return factory;
    }

    /**
     * Creates and returns a new {@link Carb}
instance based on the given
     * {@link DietPlan}.
     * <p>
     * If the diet plan is {@link DietPlan#PALEO},
a {@link Carb} instance with the
     * food item "Pistachio" is returned.
```

```
     * If the diet plan is {@link DietPlan#VEGAN},
a {@link Carb} instance with a
     * random food item from "Bread", "Lentils", or
"Pistachio" is returned.
     * If the diet plan is {@link
DietPlan#NUT_ALLERGY}, a {@link Carb} instance
     * with a random food item from "Cheese",
"Bread", or "Lentils" is returned.
     * Otherwise, a {@link Carb} instance with a
random food item from "Cheese",
     * "Bread", "Lentils", or "Pistachio" is
returned.
     *
     * @param dietPlan the diet plan for which to
create the {@link Carb} instance
     * @return a new {@link Carb} instance based on
the given diet plan
     */

    @Override
    public Macronutrient getMacronutrient(DietPlan
dietPlan) {
        if (dietPlan == DietPlan.PALEO) {
            return new Carb("Pistachio");
        } else if (dietPlan == DietPlan.VEGAN) {
            return new
Carb(super.getRandomFoodItem(new String[] {
"Bread", "Lentils", "Pistachio" }));
```

```
        } else if (dietPlan ==
DietPlan.NUT_ALLERGY) {
            return new
Carb(super.getRandomFoodItem(new String[] {
"Cheese", "Bread", "Lentils" }));
        } else {
            return new
Carb(super.getRandomFoodItem(new String[] {
"Cheese", "Bread", "Lentils", "Pistachio" }));
        }
    }
}
```

Customer.java

```
/**
 * Represents a customer with a name and a diet
plan.
 */
public class Customer {

    private final String name;
    private final DietPlan dietPlan;

    /**
     * Constructs a new customer with the specified
name and diet plan.
     *
     * @param name     the name of the customer
```

```
     * @param dietPlan the diet plan of the
customer
     */
    public Customer(String name, DietPlan dietPlan)
{
        this.name = name;
        this.dietPlan = dietPlan;
    }

    /**
     * Gets the diet plan of the customer.
     *
     * @return the diet plan of the customer
     */
    public DietPlan getDietPlan() {
        return dietPlan;
    }

    /**
     * Gets a description of the customer including
their name and diet plan.
     *
     * @return a description of the customer
     */
    public String getDescription() {
        return name + " (Diet Plan: " + dietPlan +
")";
    }
```

```
}
```

DietPlan.java

```
/**
 * Enum representing different diet plans.
 */
public enum DietPlan {
    NONE,
    PALEO,
    NUT_ALLERGY,
    VEGAN
}
```

Fat.java

```
/**
 * Represents a type of macronutrient: Fat.
 */
public class Fat extends Macronutrient {
    /**
     * Constructs a new Fat object with the
specified food item.
     *
     * @param foodItem the food item representing
the fat
     */
    public Fat(String foodItem) {
```

```
        super(foodItem);
    }

    /**
     * Returns a description of the fat.
     *
     * @return a description of the fat
     */
    @Override
    public String getDescription() {
        return "Fat: " + foodItem;
    }
}
```

FatFactory.java

```
/**
 * A factory class for creating Fat objects.
 */
public class FatFactory extends
MacronutrientFactory {
    private static FatFactory factory = null;

    /**
     * Private constructor to prevent instantiation
from outside the class.
     */
    private FatFactory() {
    }
```

```
    /**
     * Returns the singleton instance of the
FatFactory.
     *
     * @return the FatFactory instance
     */
    public static FatFactory getFactory() {
        if (factory == null) {
            factory = new FatFactory();
        }

        return factory;
    }

    /**
     * Creates and returns a Fat object based on
the given diet plan.
     *
     * @param dietPlan the diet plan for which to
create the Fat object
     * @return a Fat object based on the given diet
plan
     */
    @Override
    public Macronutrient getMacronutrient(DietPlan
dietPlan) {
        if (dietPlan == DietPlan.VEGAN) {
```

```
            return new
Fat(super.getRandomFoodItem(new String[] {
"Avocado", "Peanuts" }));
        } else if (dietPlan == DietPlan.PALEO) {
            return new
Fat(super.getRandomFoodItem(new String[] {
"Avocado", "Tuna", "Peanuts" }));
        } else if (dietPlan ==
DietPlan.NUT_ALLERGY) {
            return new
Fat(super.getRandomFoodItem(new String[] {
"Avocado", "Tuna", "Sour Cream" }));
        } else {
            return new
Fat(super.getRandomFoodItem(new String[] {
"Avocado", "Tuna", "Sour Cream", "Peanuts" }));
        }
    }
}
```

Macronutrient.java

```
/**
 * Abstract class representing a macronutrient.
 */
public abstract class Macronutrient {
    /**
     * The food item associated with the
macronutrient.
```

```
     */
    protected String foodItem;

    /**
     * Constructs a new Macronutrient with the
given food item.
     *
     * @param foodItem the food item associated
with the macronutrient
     */
    public Macronutrient(String foodItem) {
        this.foodItem = foodItem;
    }

    /**
     * Abstract method to be implemented by
subclasses to return a description of
     * the macronutrient.
     *
     * @return a description of the macronutrient
     */
    abstract public String getDescription();
}
```

MacronutrientFactory.java

```
/**
 * Abstract class representing a factory for
creating macronutrients.
```

```java
 */
public abstract class MacronutrientFactory {
    /**
     * Gets a macronutrient based on the specified diet plan.
     *
     * @param dietPlan The diet plan for which to create the macronutrient.
     * @return A macronutrient object.
     */
    public abstract Macronutrient getMacronutrient(DietPlan dietPlan);

    /**
     * Gets a random food item from the specified array of food items.
     *
     * @param foodItems An array of food items.
     * @return A random food item from the array.
     */
    protected final String getRandomFoodItem(String[] foodItems) {
        int randomIndex = (int) (Math.random() * foodItems.length);

        return foodItems[randomIndex];
    }
}
```

Protein.java

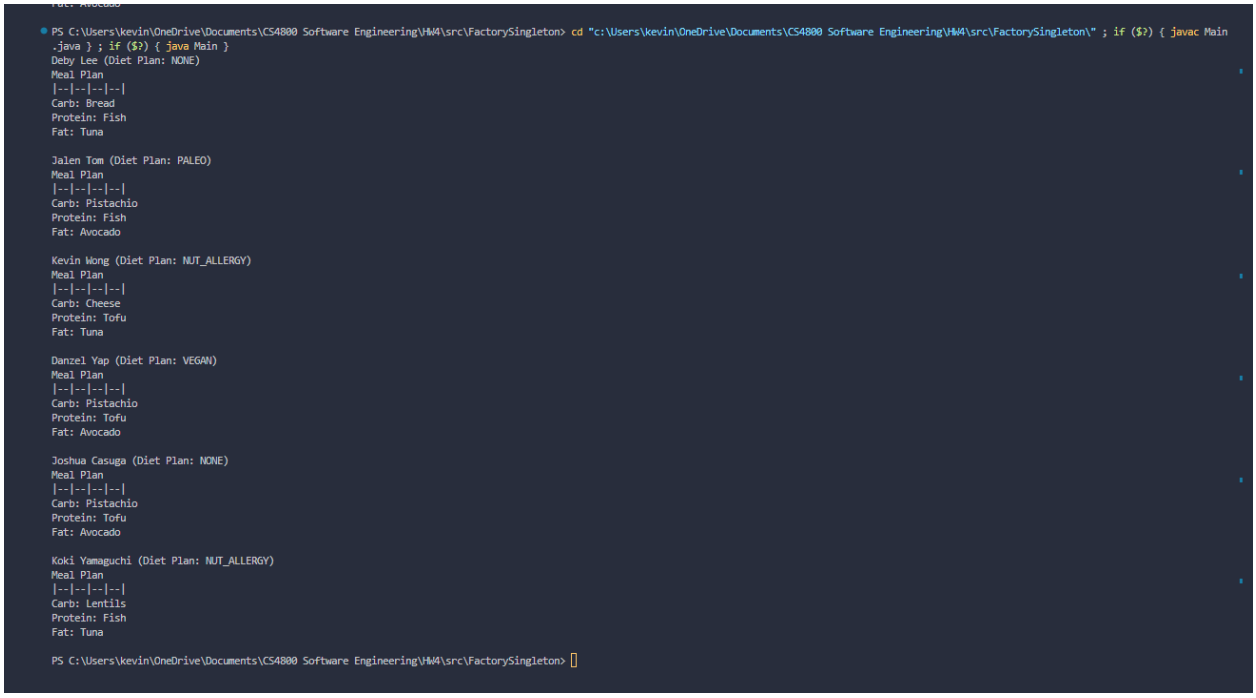
```java
/**
 * Represents a protein macronutrient.
 */
public class Protein extends Macronutrient {
    /**
     * Creates a new Protein instance with the given food item.
     *
     * @param foodItem the food item representing the protein
     */
    public Protein(String foodItem) {
        super(foodItem);
    }

    /**
     * Returns the description of the protein macronutrient.
     *
     * @return the description of the protein
     */
    @Override
    public String getDescription() {
        return "Protein: " + foodItem;
    }
}
```

ProteinFactory.java

```java
/**
 * A factory class for creating Protein
macronutrients based on the customer's
 * diet plan.
 */
public class ProteinFactory extends
MacronutrientFactory {
    private static ProteinFactory factory = null;

    private ProteinFactory() {
    }

    /**
     * Returns the singleton instance of the
ProteinFactory.
     *
     * @return the singleton instance of the
ProteinFactory
     */
    public static ProteinFactory getFactory() {
        if (factory == null) {
            factory = new ProteinFactory();
        }

        return factory;
    }
```

```
    /**
     * Creates a Protein macronutrient based on the
customer's diet plan.
     *
     * @param dietPlan the customer's diet plan
     * @return a Protein macronutrient
     */
    @Override
    public Macronutrient getMacronutrient(DietPlan
dietPlan) {
        if (dietPlan == DietPlan.VEGAN) {
            return new Protein("Tofu");
        } else if (dietPlan == DietPlan.PALEO) {
            return new
Protein(super.getRandomFoodItem(new String[] {
"Fish", "Chicken", "Beef" }));
        } else {
            return new
Protein(super.getRandomFoodItem(new String[] {
"Fish", "Chicken", "Beef", "Tofu" }));
        }
    }
}
```

Output:


PS C:\Users\kevin\OneDrive\Documents\CS4800 Software Engineering\HW4\src\FactorySingleton> cd "c:\Users\kevin\OneDrive\Documents\CS4800 Software Engineering\HW4\src\FactorySingleton\" ; if ($?) { javac Main .java } ; if ($?) { java Main }
Deby Lee (Diet Plan: NONE)
Meal Plan
|--|--|--|--|
Carb: Bread
Protein: Fish
Fat: Tuna
Jalen Tom (Diet Plan: PALEO)
Meal Plan
|--|--|--|--|
Carb: Pistachio
Protein: Fish
Fat: Avocado
Kevin Wong (Diet Plan: NUT_ALLERGY)
Meal Plan
|--|--|--|--|
Carb: Cheese
Protein: Tofu
Fat: Tuna
Danzel Yap (Diet Plan: VEGAN)
Meal Plan
|--|--|--|--|
Carb: Pistachio
Protein: Tofu
Fat: Avocado
Joshua Casuga (Diet Plan: NONE)
Meal Plan
|--|--|--|--|
Carb: Pistachio
Protein: Tofu
Fat: Avocado
Koki Yamaguchi (Diet Plan: NUT_ALLERGY)
Meal Plan
|--|--|--|--|
Carb: Lentils
Protein: Fish
Fat: Tuna
PS C:\Users\kevin\OneDrive\Documents\CS4800 Software Engineering\HW4\src\FactorySingleton>