**Homework 2**

**Github:** <https://github.com/Dubs2310/CS5800-Homework-2.git>

**Builder**

* Pizza.java

import java.util.ArrayList;  
  
public class Pizza {  
 private String size;  
 private String chain;  
 private ArrayList<String> toppings;  
  
 public Pizza(String size, String chain, ArrayList<String> toppings) {  
 this.size = size;  
 this.chain = chain;  
 this.toppings = toppings;  
 }  
  
 public String getSize() {  
 return size;  
 }  
  
 public void setSize(String size) {  
 this.size = size;  
 }  
  
 public String getChain() {  
 return chain;  
 }  
  
 public void setChain(String chain) {  
 this.chain = chain;  
 }  
  
 public void setToppings(ArrayList<String> toppings) {   
 this.toppings = toppings;  
 }  
  
 public ArrayList<String> getToppings() {  
 return toppings;  
 }  
  
 public void eat() {  
 System.*out*.println(this.size + " pizza from " + this.chain + " with " + toppings.size() + " topping(s):");  
 for (String topping: this.toppings)  
 System.*out*.print(topping + ", ");  
 System.*out*.println("\nEnjoy!\n");  
 }  
}

* PizzaBuilder.java

/\* I've provided 2 ways to set size:  
 1. either pass in size to the parameterized constructor,  
 2. or use setSize when building pizza with default constructor (if size not set, build() throws an error)  
  
 Since size is a required field, having only a parameterized constructor would work as the compiler would throw  
 a missing argument error if no parameter was passed. I only included the default constructor + setSize method  
 to do additional testing  
\*/  
  
import java.util.ArrayList;  
  
public class PizzaBuilder {  
 private ArrayList<String> toppings;  
 private String size;  
 private String chain;  
  
 public PizzaBuilder() {  
 this.size = null;  
 this.chain = "Pizza Hut";  
 this.toppings = new ArrayList<>();  
 }  
  
 // default chain is Pizza Hut, to change use method orderFrom(chain)  
 public PizzaBuilder(String size) {  
 this.size = size;  
 this.chain = "Pizza Hut";  
 this.toppings = new ArrayList<>();  
 }  
  
 public PizzaBuilder setSize(String size) {  
 this.size = size;  
 return this;  
 }  
  
 public PizzaBuilder orderFrom(String chain) {  
 this.chain = chain;  
 return this;  
 }  
  
 public PizzaBuilder addPepperoni() {  
 toppings.add("Pepperoni");  
 return this;  
 }  
  
 public PizzaBuilder addSausage() {  
 toppings.add("Sausage");  
 return this;  
 }  
  
 public PizzaBuilder addMushrooms() {  
 toppings.add("Mushrooms");  
 return this;  
 }  
  
 public PizzaBuilder addBacon() {  
 toppings.add("Bacon");  
 return this;  
 }  
  
 public PizzaBuilder addOnions() {  
 toppings.add("Onions");  
 return this;  
 }  
  
 public PizzaBuilder addExtraCheese() {  
 toppings.add("Extra Cheese");  
 return this;  
 }  
  
 public PizzaBuilder addPeppers() {  
 toppings.add("Peppers");  
 return this;  
 }  
  
 public PizzaBuilder addChicken() {  
 toppings.add("Chicken");  
 return this;  
 }  
  
 public PizzaBuilder addOlives() {  
 toppings.add("Olives");  
 return this;  
 }  
  
 public PizzaBuilder addSpinach() {  
 toppings.add("Spinach");  
 return this;  
 }  
  
 public PizzaBuilder addTomatoAndBasil() {  
 toppings.add("Tomato and Basil");  
 return this;  
 }  
  
 public PizzaBuilder addBeef() {  
 toppings.add("Beef");  
 return this;  
 }  
  
 public PizzaBuilder addHam() {  
 toppings.add("Ham");  
 return this;  
 }  
  
 public PizzaBuilder addPesto() {  
 toppings.add("Pesto");  
 return this;  
 }  
  
 public PizzaBuilder addSpicyPork() {  
 toppings.add("Spicy Pork");  
 return this;  
 }  
  
 public PizzaBuilder addHamAndPineapple() {  
 toppings.add("Ham and Pineapple");  
 return this;  
 }  
  
 // if size not set, throw null exception  
 public Pizza build() throws NullPointerException {  
 if (this.size == null)  
 throw new NullPointerException("Please select size of the pizza before ordering");  
  
 return new Pizza(this.size, this.chain, this.toppings);  
 }  
}

* Main.java

public class Main {  
 public static void main(String[] args) {  
 // by default, pizza is ordered from Pizza Hut unless stated otherwise  
  
 // Large pizza from Pizza Hut with 3 toppings  
 new PizzaBuilder("Large")  
 .addMushrooms()  
 .addTomatoAndBasil()  
 .addHam()  
 .build()  
 .eat();  
  
 // Small pizza from Pizza Hut with 2 toppings  
 new PizzaBuilder("Small")  
 .addPesto()  
 .addBeef()  
 .build()  
 .eat();  
  
 // Medium pizza from Little Caesars with 8 toppings  
 new PizzaBuilder("Medium")  
 .orderFrom("Little Caesars")  
 .addMushrooms()  
 .addExtraCheese()  
 .addOlives()  
 .addChicken()  
 .addHamAndPineapple()  
 .addPeppers()  
 .addPepperoni()  
 .addSausage()  
 .build()  
 .eat();  
  
 // Small pizza from Little Caesars with 6 toppings  
 new PizzaBuilder("Small")  
 .orderFrom("Little Caesars")  
 .addMushrooms()  
 .addExtraCheese()  
 .addHamAndPineapple()  
 .addPeppers()  
 .addSpicyPork()  
 .addSpinach()  
 .build()  
 .eat();  
  
 // Small pizza from Dominos with 1 topping  
 new PizzaBuilder("Small")  
 .orderFrom("Dominos")  
 .addExtraCheese()  
 .build()  
 .eat();  
  
 // Large pizza from Dominos with 3 toppings  
 new PizzaBuilder("Large")  
 .orderFrom("Dominos")  
 .addExtraCheese()  
 .addBacon()  
 .addOnions()  
 .build()  
 .eat();  
  
 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EXTRA CASES, TESTING REQUIRED SIZE FIELD WHEN USING DEFAULT CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //  
  
 // size field not provided  
 try {  
 new PizzaBuilder()  
 .addExtraCheese()  
 .build()  
 .eat();  
 } catch (NullPointerException e) {  
 System.*out*.println(e.getMessage() + '\n');  
 }  
  
 // size field not provided in default constructor but set later using setSize(size)  
 new PizzaBuilder()  
 .setSize("Small")  
 .addExtraCheese()  
 .build()  
 .eat();  
 }  
}

* Output**Text

  Description automatically generated**

**Abstract Factory, Factory, Singleton**

* Buildable.java

public interface Buildable {  
 void build();  
}

* Repairable.java

public interface Repairable {  
 void repair();  
}

* Restorable.java

public interface Restorable {  
 void restore();  
}

* CarFactory.java

public interface CarFactory extends Buildable, Repairable, Restorable {}

* HondaFactory.java

public class HondaFactory implements CarFactory {  
 private static HondaFactory *instance* = null;  
 private HondaFactory() {}  
  
 public static HondaFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new HondaFactory();  
 else System.*out*.println("HondaFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Honda built a car");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Honda repaired a car");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Honda restored a car");  
 }  
}

* PorscheFactory.java

public class PorscheFactory implements CarFactory {  
 private static PorscheFactory *instance* = null;  
 private PorscheFactory() {}  
  
 public static PorscheFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new PorscheFactory();  
 else System.*out*.println("PorscheFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Porsche built a car");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Porsche repaired a car");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Porsche restored a car");  
 }  
}

* TeslaFactory.java

public class TeslaFactory implements CarFactory {  
 private static TeslaFactory *instance* = null;  
 private TeslaFactory() {}  
  
 public static TeslaFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new TeslaFactory();  
 else System.*out*.println("TeslaFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Tesla built a car");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Tesla repaired a car");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Tesla restored a car");  
 }  
}

* AirplaneFactory.java

public interface AirplaneFactory extends Buildable, Repairable, Restorable {}

* AirbusFactory.java

public class AirbusFactory implements AirplaneFactory {  
 private static AirbusFactory *instance* = null;  
 private AirbusFactory() {}  
  
 public static AirbusFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new AirbusFactory();  
 else System.*out*.println("AirbusFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Airbus built an airplane");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Airbus repaired an airplane");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Airbus restored an airplane");  
 }  
}

* BoeingFactory.java

public class BoeingFactory implements AirplaneFactory {  
 private static BoeingFactory *instance* = null;  
 private BoeingFactory() {}  
  
 public static BoeingFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new BoeingFactory();  
 else System.*out*.println("BoeingFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Boeing built an airplane");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Boeing repaired an airplane");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Boeing restored an airplane");  
 }  
}

* EmbraerFactory.java

public class EmbraerFactory implements AirplaneFactory {  
 private static EmbraerFactory *instance* = null;  
 private EmbraerFactory() {}  
  
 public static EmbraerFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new EmbraerFactory();  
 else System.*out*.println("EmbraerFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Embraer built an airplane");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Embraer repaired an airplane");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Embraer restored an airplane");  
 }  
}

* BoatFactory.java

public interface BoatFactory extends Buildable, Repairable, Restorable {}

* SeaRayFactory.java

public class SeaRayFactory implements BoatFactory {  
 private static SeaRayFactory *instance* = null;  
 private SeaRayFactory() {}  
  
 public static SeaRayFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new SeaRayFactory();  
 else System.*out*.println("SeaRayFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("SeaRay built a boat");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("SeaRay repaired a boat");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("SeaRay restored a boat");  
 }  
}

* MastercraftFactory.java

public class MastercraftFactory implements BoatFactory {  
 private static MastercraftFactory *instance* = null;  
 private MastercraftFactory() {}  
  
 public static MastercraftFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new MastercraftFactory();  
 else System.*out*.println("MastercraftFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Mastercraft built a boat");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Mastercraft repaired a boat");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Mastercraft restored a boat");  
 }  
}

* BertramFactory.java

public class BertramFactory implements BoatFactory {  
 private static BertramFactory *instance* = null;  
 private BertramFactory() {}  
  
 public static BertramFactory getInstance() {  
 if (*instance* == null)  
 *instance* = new BertramFactory();  
 else System.*out*.println("BertramFactory instance already exists");  
 return *instance*;  
 }  
  
 @Override  
 public void build() {  
 System.*out*.println("Bertram built a boat");  
 }  
  
 @Override  
 public void repair() {  
 System.*out*.println("Bertram repaired a boat");  
 }  
  
 @Override  
 public void restore() {  
 System.*out*.println("Bertram restored a boat");  
 }  
}

* FactoryCreator.java

public class FactoryCreator {  
  
 public static CarFactory createCarFactory(String carType) {  
 if (carType.equalsIgnoreCase("Honda"))  
 return HondaFactory.*getInstance*();  
 else if (carType.equalsIgnoreCase("Porsche"))  
 return PorscheFactory.*getInstance*();  
 else if (carType.equalsIgnoreCase("Tesla"))  
 return TeslaFactory.*getInstance*();  
 else {  
 System.*out*.println("Please specify a valid car type");  
 return null;  
 }  
 }  
  
 public static AirplaneFactory createAirplaneFactory(String airplaneType) {  
 if (airplaneType.equalsIgnoreCase("Airbus"))  
 return AirbusFactory.*getInstance*();  
 else if (airplaneType.equalsIgnoreCase("Boeing"))  
 return BoeingFactory.*getInstance*();  
 else if (airplaneType.equalsIgnoreCase("Embraer"))  
 return EmbraerFactory.*getInstance*();  
 else {  
 System.*out*.println("Please specify a valid airplane type");  
 return null;  
 }  
 }  
  
 public static BoatFactory createBoatFactory(String boatType) {  
 if (boatType.equalsIgnoreCase("SeaRay"))  
 return SeaRayFactory.*getInstance*();  
 else if (boatType.equalsIgnoreCase("Mastercraft"))  
 return MastercraftFactory.*getInstance*();  
 else if (boatType.equalsIgnoreCase("Bertram"))  
 return BertramFactory.*getInstance*();  
 else {  
 System.*out*.println("Please specify a valid boat type");  
 return null;  
 }  
 }  
   
}

* Main.java

public class Main {  
 public static void main(String[] args) {  
 // car factory instances  
 HondaFactory honda = (HondaFactory) FactoryCreator.*createCarFactory*("Honda");  
 honda.build();  
 honda.repair();  
 honda.restore();  
 System.*out*.println();  
  
 PorscheFactory porsche = (PorscheFactory) FactoryCreator.*createCarFactory*("Porsche");  
 porsche.build();  
 porsche.repair();  
 porsche.restore();  
 System.*out*.println();  
  
 TeslaFactory tesla = (TeslaFactory) FactoryCreator.*createCarFactory*("Tesla");  
 tesla.build();  
 tesla.repair();  
 tesla.restore();  
 System.*out*.println();  
  
 // airplane factory instances  
 AirbusFactory airbus = (AirbusFactory) FactoryCreator.*createAirplaneFactory*("Airbus");  
 airbus.build();  
 airbus.repair();  
 airbus.restore();  
 System.*out*.println();  
  
 BoeingFactory boeing = (BoeingFactory) FactoryCreator.*createAirplaneFactory*("Boeing");  
 boeing.build();  
 boeing.repair();  
 boeing.restore();  
 System.*out*.println();  
  
 EmbraerFactory embraer = (EmbraerFactory) FactoryCreator.*createAirplaneFactory*("Embraer");  
 embraer.build();  
 embraer.repair();  
 embraer.restore();  
 System.*out*.println();  
  
 // attempting to recreate tesla and airbus instances  
 TeslaFactory tesla2 = (TeslaFactory) FactoryCreator.*createCarFactory*("Tesla");  
 AirbusFactory airbus2 = (AirbusFactory) FactoryCreator.*createAirplaneFactory*("Airbus");  
 System.*out*.println();  
  
 // boat factory instances  
 SeaRayFactory seaRay = (SeaRayFactory) FactoryCreator.*createBoatFactory*("SeaRay");  
 seaRay.build();  
 seaRay.repair();  
 seaRay.restore();  
 System.*out*.println();  
  
 MastercraftFactory mastercraft = (MastercraftFactory) FactoryCreator.*createBoatFactory*("Mastercraft");  
 mastercraft.build();  
 mastercraft.repair();  
 mastercraft.restore();  
 System.*out*.println();  
  
 BertramFactory bertram = (BertramFactory) FactoryCreator.*createBoatFactory*("Bertram");  
 bertram.build();  
 bertram.repair();  
 bertram.restore();  
 }  
}

* Output

Text

Description automatically generatedText

Description automatically generated