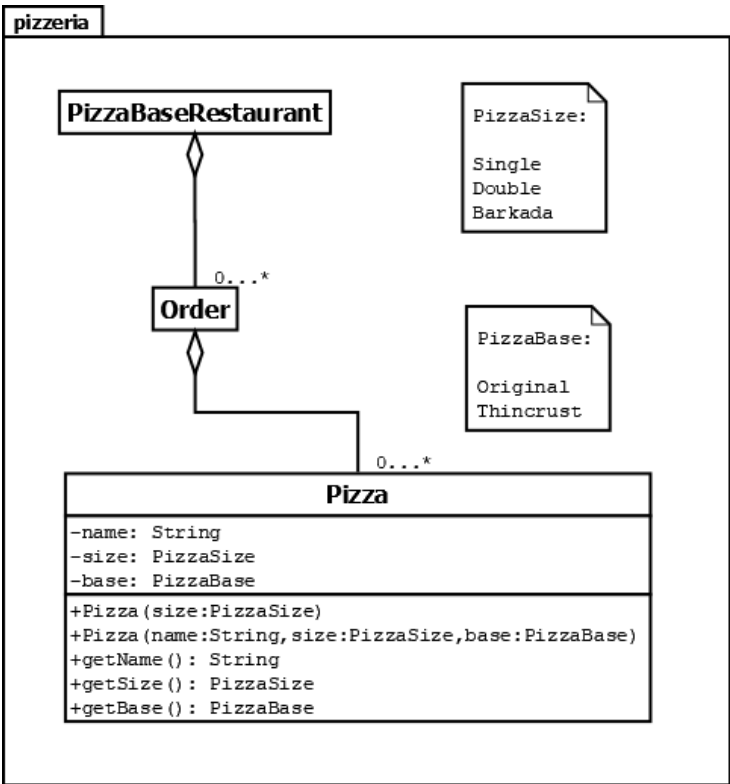


Objective

In this exercise you will explore the purpose of using *polymorphism* on classes and interfaces applied with inheritance. *Static polymorphism* will be evident on the implementation of class constructors, *dynamic polymorphism* will be applied by overriding superclass methods, and *inclusion polymorphism* will be executed during ordering of multiple pizza types referenced to a single order list.

Directions

Version 1: Non-abstract Class Pizza



1. Start by changing your working directory to `.../version1` on your computer.
2. To the `pizzeria` package, you will add the `Pizza` class as modeled by the UML diagram above. `PizzaSize` and `PizzaBase` enumeration constructs have already been implemented for you.
3. Implement the `Pizza` class as modeled in the above UML diagram.
4. It must include a `name` attribute with type `String`, a `size` attribute with type `PizzaSize`, and a `base` attribute with type `PizzaBase`.
5. It must include a public constructor that takes one parameter: `size`. In this constructor, the default value of the `name` attribute is `null` and the default value of the `base` attribute is `Original`. Assign the value of the `size` attribute with the parameter value.
6. It must also include another public constructor that takes three parameters: `name`, `size`, and `base`. Initialize all the corresponding attributes of the class.
7. Implement all *accessor* methods as modeled in the above UML diagram.

Test the Code

In the main directory (`.../version1`), compile and execute the `TestPizzaBaseRestaurant` program. The output should be:

```
Creating the order for customer Lelouch.  
He orders a Single Hawaiian Thincrust pizza and 2 Barkada Size Ham and Cheese  
Original base pizzas
```

```
Creating the order for customer Raito.  
He orders a Double pizza and a Single Thincrust Pepperoni pizza.
```

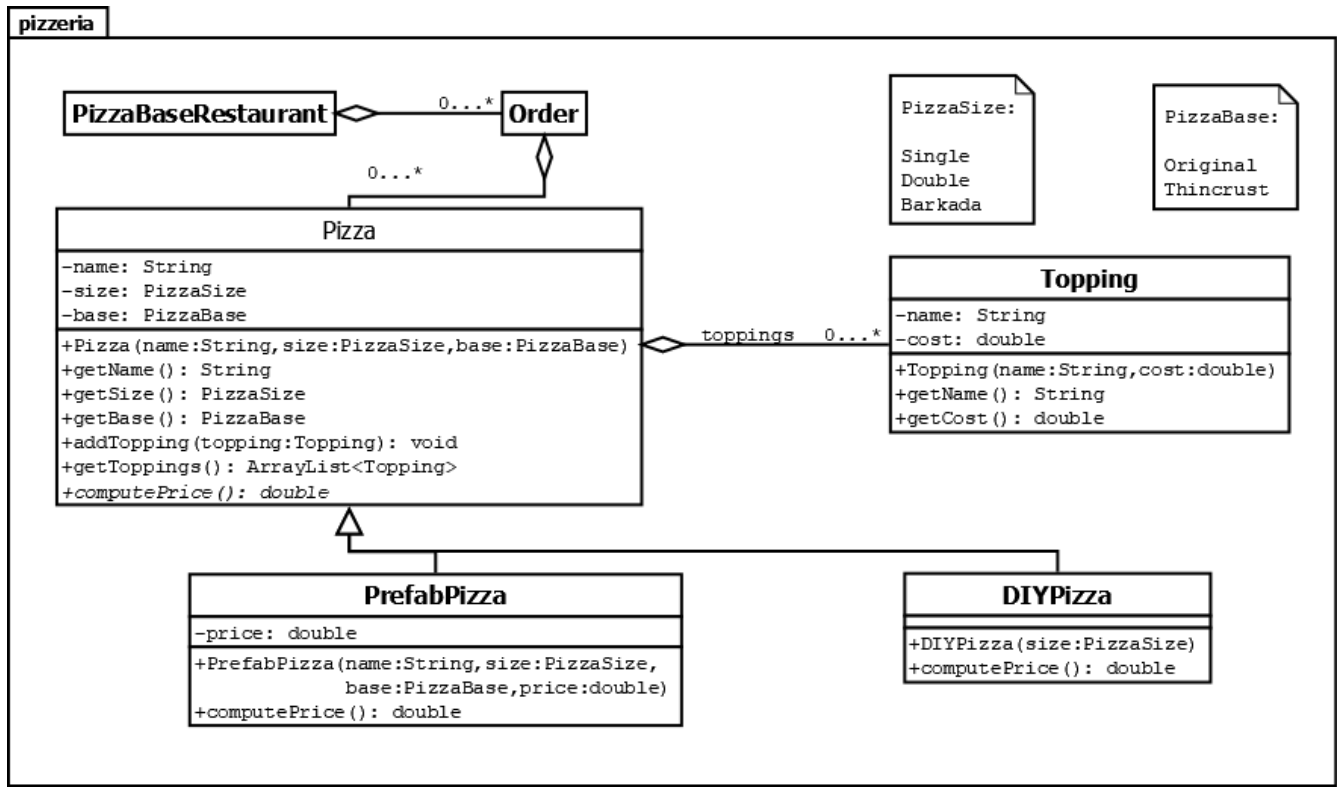
Creating the order for customer Hinamori.  
She orders a Single Blueberry pizza, a Single Thincrust Strawberry pizza, and a Double Size Diamond Dust pizza.

Retrieving the customer Lelouch with his/her orders.  
He/She ordered:  
Name            Size      Base  
Hawaiian        Single   Thincrust  
Ham and Cheese   Barkada   Original  
Ham and Cheese   Barkada   Original

Retrieving the customer Raito with his/her orders.  
He/She ordered:  
Name            Size      Base  
--No name--     Double   null  
Pepperoni       Single   Thincrust

Retrieving the customer Hinamori with his/her orders.  
He/She ordered:  
Name            Size      Base  
Blueberry        Single   Original  
Strawberry       Single   Thincrust  
Diamond Dust     Double   Original

Version 2: Prefab and DIY Pizzas



Start by changing your working directory to `.../version2` on your computer. Copy the Version 1 pizzeria project files (from `.../version1`) in this package directory.

Creating a Topping Class

- 1. To the `pizzeria` package, you will add the `Topping` class as modeled by the UML diagram above.
- 2. Implement the `Topping` class as modeled in the above UML diagram having a `name` attribute of type `String` and `cost` attribute of type `double`.
- 3. It must include a public constructor that takes two parameters: `name` and `cost`. Initialize all the corresponding attributes of the class.
- 4. It must also include accessors of all attributes of the class.

### Modifying Pizza Class

1. Remove the public constructor that takes one parameter, `size`.
2. Change the class `Pizza` into an abstract class.
3. Add a private association attribute called `toppings` that will hold an `ArrayList` of `Topping` objects for a specific pizza. Initialize the `toppings` attribute in the `Pizza` constructor using `ArrayList`'s default constructor.
4. Include a *mutator* method, `addTopping`, that will add a `Topping` object into the `toppings` attribute.
5. Include an *accessor* method, `getToppings`, that will return a copy of the object held by the `toppings` attribute.
6. Declare an abstract method `computePrice` as shown in the UML diagram.

### Creating PrefabPizza Class

1. To the `pizzeria` package, you will add the `PrefabPizza` class inheriting the `Pizza` abstract class as modeled by the UML diagram above.
2. Implement the `PrefabPizza` class as modeled in the above UML diagram having a `price` attribute of type `double`.
3. It must include a public constructor that takes four parameters: `name`, `size`, `base`, and `price`. You can use the superclass constructor to initialize the first three parameters then set the corresponding attribute for the fourth parameter.
4. Override the abstract method `computePrice` from the `Pizza` abstract class and just return the value of the `price` attribute.

### Creating DIYPizza Class

1. To the `pizzeria` package, you will add the `DIYPizza` class inheriting the `Pizza` abstract class as modeled by the UML diagram above.
2. Implement the `DIYPizza` class as modeled in the above UML diagram.
3. It must include a public constructor that takes one parameter: `size`. The default name of a `DIYPizza` object is "DIY Pizza" and all DIY pizzas have an original base.
4. Override the abstract method `computePrice` from the `Pizza` abstract class where the price of the pizza will be the total cost of all the toppings added into the `DIYPizza` object plus a default base cost of 100 multiplied by 1 for `Single` size, 2 for `Double` size and 5 for `Barkada` size.

### Test the Code

In the main directory (`.../version2`), compile and execute the `TestPizzaBaseRestaurant` program. The output should be:

Creating the order for customer Lelouch.

He orders a Single Hawaiian Thincrust pizza and 2 Barkada Size Ham and Cheese  
Original base pizzas

Creating the order for customer Raito.

He orders a Double size DIYPizza with Ham, Pork meat, and Onions, and a Single  
DIYPizza with Pepperoni, Onions, Pork meat, and Yellow peppers.

Creating the order for customer Hinamori.

She orders a Single DIYPizza with Bluberry, Onions, and Olives, a Single DIYPizza  
with Strawberry, Ham, and Olives, and a Double DIYPizza with Pineapple, Onions,  
Yellow Peppers, and Cheese.

Retrieving the customer Lelouch with his/her orders.

He/She ordered:

Single Thincrust Hawaiaan - 120.00

Barkada Original Ham and Cheese - 444.44

Barkada Original Ham and Cheese - 444.44

\*\*\*Total cost: 1008.88

Retrieving the customer Raito with his/her orders.

He/She ordered:

Double DIY Pizza - 281.14

- Onions

- Ham

- Pork Meat

Single DIY Pizza - 163.82

- Onions

- Pork Meat

- Yellow Peppers

- Pepperoni

\*\*\*Total cost: 444.96

Retrieving the customer Hinamori with his/her orders.

He/She ordered:

Single DIY Pizza - 151.21

- Onions

- Olives

- Blueberry

Single DIY Pizza - 150.63

- Ham

- Olives

- Strawberry

Double DIY Pizza - 319.18

- Pineapple

- Onions

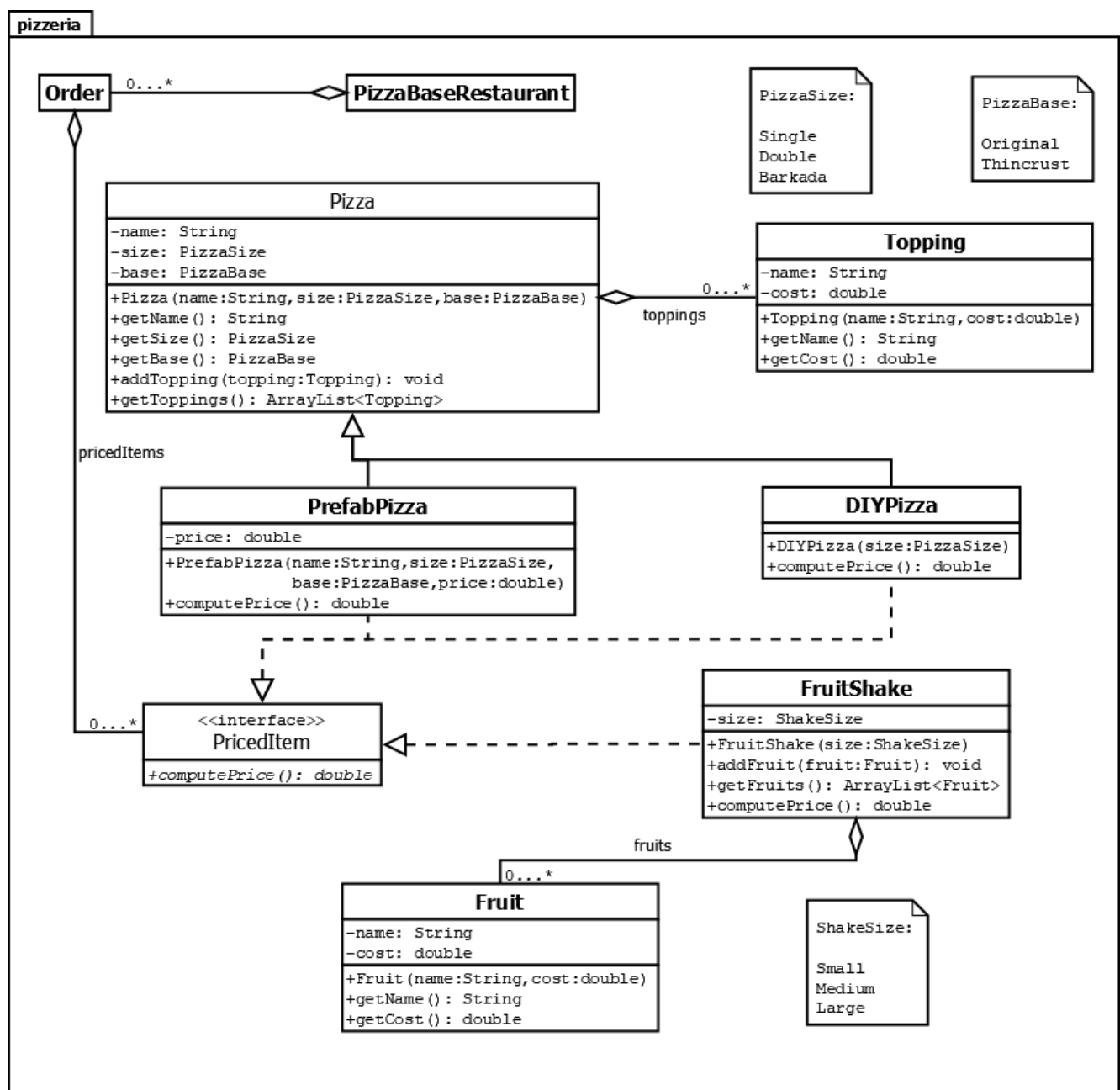
- Cheese

- Yellow Peppers

\*\*\*Total cost: 621.02

By inheriting the abstract method `computePrice` to two different classes, the `computePrice` method executed differently for each type of pizza while being in the same collection of `Pizza` which demonstrates inclusion polymorphism or subtyping *using class inheritance*.

Version 3: Calculation of Cost via PricedItem



Start by changing your working directory to `.../version3` on your computer. Copy the Version 2 pizzeria project files (from `.../version2`) in this package directory.

Modifying Pizza Class

Since we are going to create an interface `PricedItem` that will contain the abstract method `computePrice`, remove the abstract method in the class `Pizza`.

Implementing PricedItem Interface in PrefabPizza, DIYPizza, and FruitShake

1. To the `pizzeria` package, you will add the `PricedItem` interface as modeled by the UML diagram above.
2. Implement the interface `PricedItem` in `PrefabPizza`, `DIYPizza`, and `FruitShake` classes.
3. Since the `computePrice` method is already overridden in `PrefabPizza` and `DIYPizza`, override the method in `FruitShake`. The price of a fruit shake is computed by the total cost of all the `fruits` in the `FruitShake` object added by a default price of 40. The price is then multiplied by 1 if the size of the fruit shake is `Small`, 1.45 if the size is `Medium` and 1.85 if the size is `Large`.

Modifying the Order Class

1. In class `Order`, add an additional private association attribute called that will hold a `ArrayList` of `PricedItem` objects. Initialize the `pricedItems` attribute in the `Order` constructor using `ArrayList`'s default constructor.

2. Remove the existing attribute `pizzas` and replace all its implementation to correspond with the new attribute `pricedItems`.
3. Add a public method in class `Order` that will return a double value named `computeTotalPayment` that will accept no parameters. It must return the total cost of all `pricedItems` using the `computePrice` method.

### Test the Code

In the main directory (.../version3), compile and execute the `TestPizzaBaseRestaurant` program. The output should be:

Creating the order for customer Lelouch.

He orders a Single Hawaiian Thincrust pizza, 2 Barkada Size Ham and Cheese Original base pizzas, and 5 large Mango fruit shakes

Creating the order for customer Raito.

He orders a Double size DIYPizza with Ham, Pork meat, and Onions, a Single DIYPizza with Pepperoni, Onions, Pork meat, and Yellow peppers, a small Sweetcorn with Coconut fruit shake, and a medium Mango Pineapple fruit shake.

Creating the order for customer Hinamori.

She orders a Single DIYPizza with Blueberry, Onions, and Olives, a Single DIYPizza with Strawberry, Ham, and Olives, a Double DIYPizza with Pineapple, Onions, Yellow Peppers, and Cheese, and 5 medium strawberry shakes.

Retrieving the customer Lelouch with his/her orders.

He/She ordered:

Single Thincrust Hawaiaan - 120.00

Barkada Original Ham and Cheese - 444.44

Barkada Original Ham and Cheese - 444.44

Large Fruit Shake - 106.28

-Mango

Large Fruit Shake - 106.28

-Mango

Large Fruit Shake - 106.28

-Mango

Large Fruit Shake - 106.28

-Mango

Large Fruit Shake - 106.28

-Mango

\*\*\*Total cost: 1540.29

Retrieving the customer Raito with his/her orders.

He/She ordered:

Double DIY Pizza - 281.14

-Onions

-Ham

-Pork Meat

Single DIY Pizza - 163.82

-Onions

-Pork Meat

-Yellow Peppers

-Pepperoni

Small Fruit Shake - 52.30

-Sweetcorn

-Coconut

Medium Fruit Shake - 104.50

-Mango

-Pineapple

\*\*\*Total cost: 601.76

Retrieving the customer Hinamori with his/her orders.

He/She ordered:

Single DIY Pizza - 151.21

-Onions

-Olives

-Blueberry

```
Single DIY Pizza - 150.63
    -Ham
    -Olives
    -Strawberry
Double DIY Pizza - 319.18
    -Pineapple
    -Onions
    -Cheese
    -Yellow Peppers
Medium Fruit Shake - 86.99
    -Strawberry
Medium Fruit Shake - 86.99
    -Strawberry
Medium Fruit Shake - 86.99
    -Strawberry
Medium Fruit Shake - 86.99
    -Strawberry
Medium Fruit Shake - 86.99
    -Strawberry
```

```
***Total cost: 1055.95
```

Since the `Order` holds any object that is a `a.PricedItem`, the price can be computed collectively while being stored in only one collection. It does not matter whether it is a `Pizza` object or a `FruitShake` object. This exhibits the benefit of *implementing interfaces to be used as subtypes (inclusion polymorphism)*.