Learning Outcomes

Upon completing this assignment, students should be able to

- Use Gradle to initialize a repository.
- Add an existing *package* to a new project.
- Implement a class *hierarchy* (using `extends`).
- Store objects in a collection that behave *polymorphicly*.

Introduction

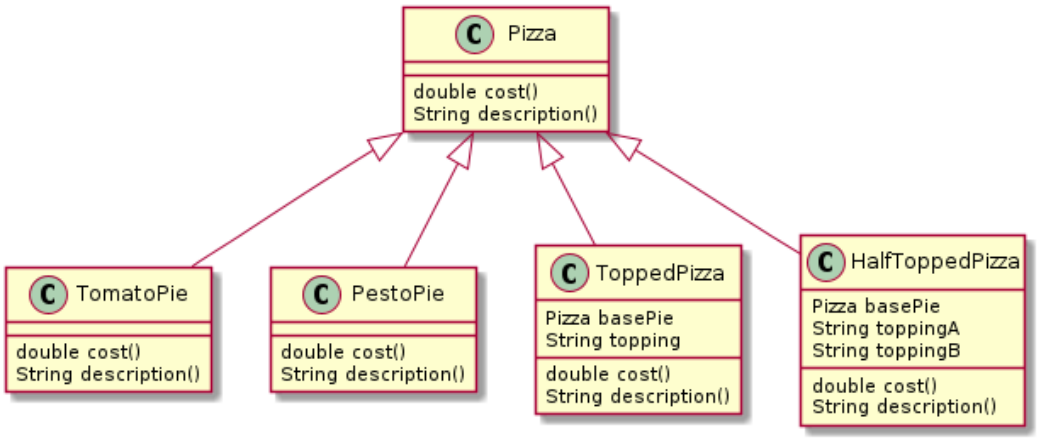
Using your `Menu` class from the previous assignment as the interface, you will implement a pizza ordering simulation. The simulation will present a main menu where a pizza can be ordered, the list of ordered pizzas printed, or the program terminated.

When ordering a pizza, the user will select a tomato or pesto base and then add toppings. The user can end this be either placing the order for the pizza or canceling it.

Method

Class Hierarchy

The `Pizza` class is the base of a hierarchy of classes that includes two base pizzas, with either tomato or pesto sauce, and two topped pizzas, one with a single topping and one with two toppings, one on each half. This relationship is shown in the following class diagram:



`Pizza` The parent of the hierarchy, the `Pizza` class will be the element type in a `List`(`ArrayList`) so that *any* type of pizza can be stored in the collection and the two functions, `cost` and `description` can be called *polymorphicly*.

`TomatoPie` A crust with tomato sauce on it, this is one of the two most basic pizzas available. It costs $7.00

`PestoPie` A crust with basil pesto sauce on it, this is the *other* basic pizza available. It costs $8.50

`ToppedPizza` A base `Pizza` (must be tracked by the objects of this class) *decorated* with a topping. The constructor for this class takes a `Pizza` to top, a `String` naming the topping, and a `double` representing the additional cost of this topping.

`HalfToppedPizza` A decorated base `Pizza` topped with *two* different toppings, each on half of the pie. The constructor takes the `Pizza` to be topped, *two* `String`s, one for each topping, and a `double` for the price of adding both halves.

The `cost` is just returned by the `TomatoPie` and `PestoPie`. Topped (half or full) pizzas cost whatever the topped pizza costs plus the cost of this topping.

The `description` of the two basic pizza are `"tomato pizza"` and `"pesto pizza"`. A topped pizza with, say, `pepperoni`, would return the description of the pizza it tops, followed by `", pepperoni"`. One really good pizza could then be described as

```
tomato pizza, cheese, onion, pepperoni
```

Getting Started

Start by creating a new directory for your solution to `F23-205-<ccid>-p002` (you **will** name the repo on Gitea with that format; call the directory whatever you want).

Initialize the directory with Gradle. Name the project `pizza` and have the source package be `shop`. Modify the created project so that the main class is called `shop.Ordering` and so that `gradle run` passes the keyboard in to the running program.

Compile and run the hello world created by Gradle so you feel like your code works.

`menu.Menu`

You will add your `Menu` class from the previous assignment to the project by copying the code into a new `menu` package in the project. For testing the setup, add a menu to `Ordering` with a hard-coded file that it opens. Then call `match`, just to see `Ordering` compile with the `Menu`.

Define `pizza.Pizza` and `pizza.TomatoPie`. Make a `List` of `pizza.Pizza` in the main program and see if you can create and insert a couple of tomato pizza into it and print their costs and descriptions.

You may want to write a second executable program, down in the `test` folder, that creates each of the different kinds of pizza with known values and then tests that `cost` and `description` work as expected.

Input/Output

This program will use multiple instances of your `Menu` class, all fields of your `Ordering` object (you are constructing an `Ordering` in `main` and then calling `run`, right?). The following are examples, not text you must use.

Main Menu

The main loop in the program is to show this menu and then process the input the user gives.

```
order - order another pizza
list - show all pizzas already ordered
quit - end the program
```

Basic Pie Menu

When the user orders another pizza, start by asking what kind of basic pie they want to start with

```
tomato - tomato pie
pesto - pesto pie
cancel - cancel this order
```

Build the Pizza

This menu will be shown after the user has selected a basic pizza. Note that the toppings shown are the kind that Oscar the Grouch, from Sesame Street, might like. You are to pick 4 or 5 toppings that you like for your simulation.

```
glass - add broken glass to pizza
sticks - add twigs and plant debris to pizza
sock - add a dirty gym sock to pizza
lint - add some under-couch lint to pizza
mold - add some moldy, not-supposed-to-be blue cheese
half - add two half toppings
cancel - cancel this order
place - place this order
```

Half Toppings

This menu should repeat the toppings in the previous menu. It makes no sense to order from this menu because user must enter two toppings before the pizza is "valid". I leave it to you to permit **cancel** the whole order and/or **remove** the half and half topping when the user is using this menu.

Since the same menu will appear twice in a row, even when the user entered something correctly, use the prompt to let them know what is going on.

Finishing the Order

When the user cancels an order, go back to main menu loop w/o changing the list of pizzas.

When the user places an order, put the pizza in the list of pizza and then go back to the main menu loop.

Design Considerations

Testing

Use a separate testing program to test whether `ToppedPizza` and `HalfToppedPizza` have correctly working `cost` and `description` methods.

Remember that you can use commandline input redirection (the < character) to feed a file to a running program as the keyboard. Maybe do this with a final list command to see that the list has the pizzas that were ordered.

Documentation

README

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted.

Deliverables

Submission medium: `git` to Gitea at `cs-devel.potsdam.edu`.