Welcome to the Object-Oriented Programming Master Challenge!

Congratulations, you've made it to the **object-oriented programming** master challenge.

In this challenge, we're going to build a complete application, using all the **principles of object-oriented programming**, we've covered in the last two sections of this course.

Bill's Burger Challenge

Bill runs a fast food hamburger restaurant, and sells hamburger meals.

His meal orders are composed of three items, the hamburger, the drink, and the side item.

Your application lets Bill select the type of burgers, and some of the additional items, or extras, that can be added to the burgers, as well as the actual pricing.

Bill's Burger Challenge - The objects

You need a meal order.

- This should be composed of exactly one burger, one drink, and one side item.
- The most common meal order should be created without any arguments, like a regular burger, a small coke, and fries, for example.
- You should be able to create other meal orders, by specifying different burgers, drinks, and side items.

You need a drink, and side item.

- The drink should have at least a type, size and price, and the price of the drink should change for each size.
- The side item needs at least a type and price.

Bill's Burger Challenge - The objects

You need burgers.

- Every hamburger should have a burger type, a base price, and up to a maximum of three extra toppings.
- The constructor should include only the burger type and price.
- Extra Toppings on a burger need to be added somehow, and priced according to their type.

The deluxe burger bonus.

- Create a deluxe burger meal, with a deluxe burger, that has a set price, so that any additional toppings do not change the price.
- The deluxe burger should have room for an additional two toppings.

Bill's Burger Challenge - The functionality

Your main method should have code to do the following:

- Create a default meal, that uses the no arguments constructor.
- Create a meal with a burger, and the drink and side item of your choice, with up to 3 extra toppings.
- Create a meal with a deluxe burger, where all items, drink, side item and toppings up to 5 extra toppings, are included in the burger price.

Bill's Burger Challenge - The functionality

For each meal order, you'll want to perform these functions:

- Add some additional toppings to the burger.
- Change the size of the drink.
- Print the itemized list. This should include the price of the burger, any extra toppings, the drink price based on size, and the side item price.
- Print the total due amount for the meal.

Bill's Burger Challenge - The objects

You need a meal order.

- This should be composed of exactly one burger, one drink, and one side item.
- The most common meal order should be created without any arguments, like a regular burger, a small coke, and fries, for example.
- You should be able to create other meal orders, by specifying different burgers, drinks, and side items.

You need a drink, and side item.

- The drink should have at least a type, size and price, and the price of the drink should change for each size.
- The side item needs at least a type and price.

Bill's Burger Challenge - The objects

You need burgers.

- Every hamburger should have a burger type, a base price, and up to a maximum of three extra toppings.
- The constructor should include only the burger type and price.
- Extra Toppings on a burger need to be added somehow, and priced according to their type.

The deluxe burger bonus.

- Create a deluxe burger meal, with a deluxe burger, that has a set price, so that any additional toppings do not change the price.
- The deluxe burger should have room for an additional two toppings.

Initial Design Considerations

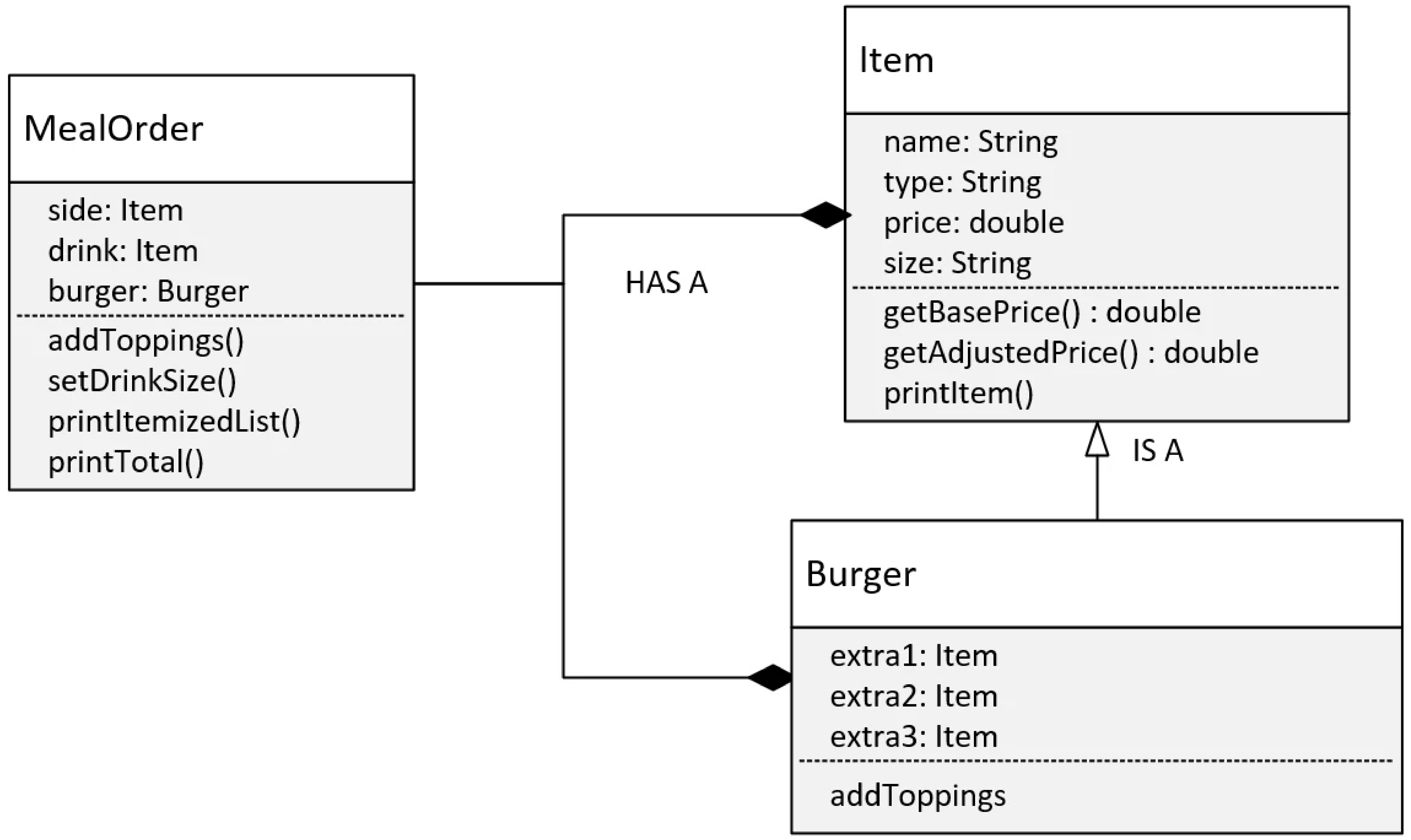
So here is the diagram of my design.

This diagram doesn't include the DeluxeBurger class. We'll look at that a bit later.

The MealOrder class uses **composition** in this design. It's composed of a burger, as well as drink and side, which will just be Items.

I've used **inheritance** for the Item and Burger relationships, which means Burger is an Item.

Every Item has a name, type, price or base price, and a size.

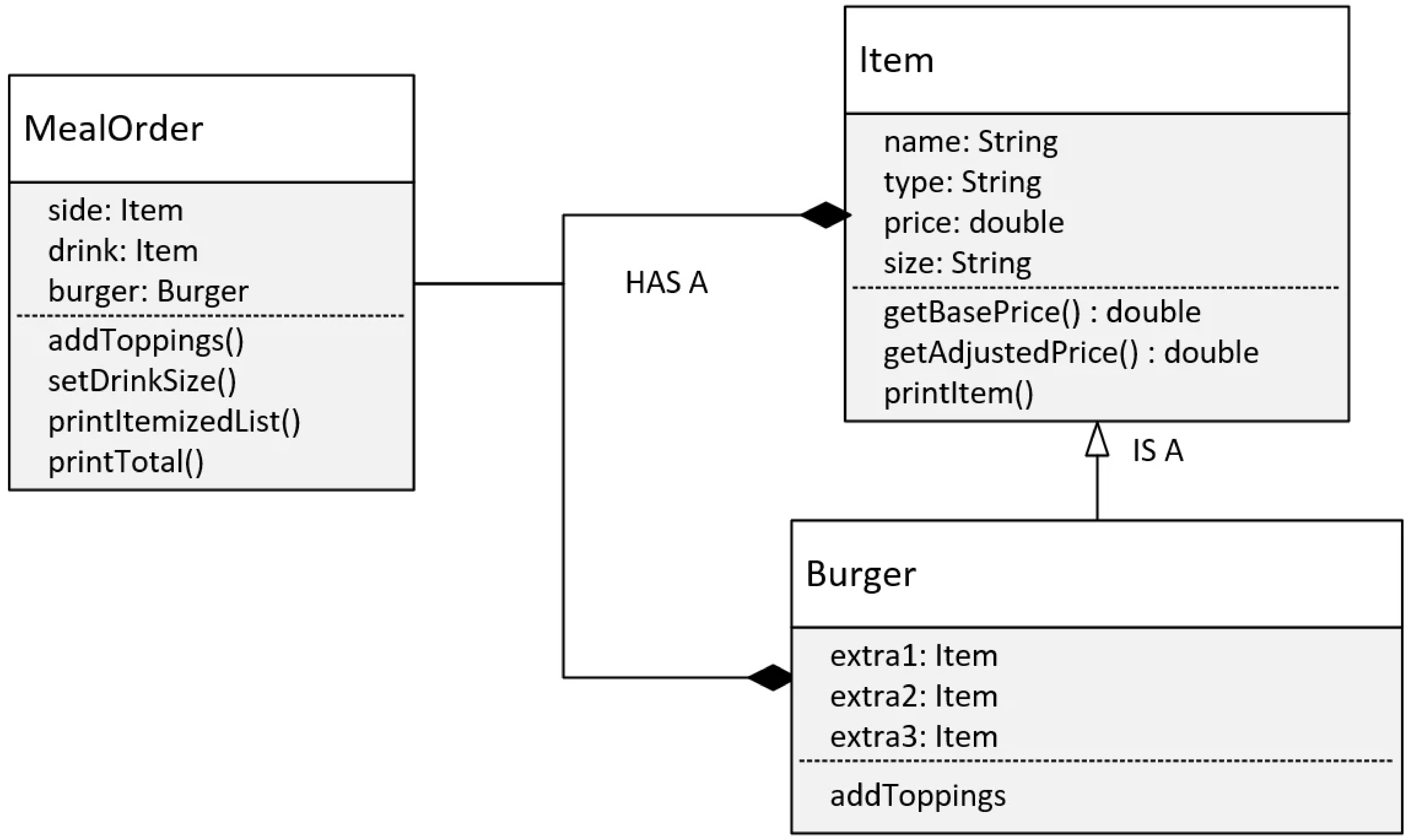


Initial Design Considerations

Item has the method getBasePrice, which is really just a getter method for the price, but the name is more descriptive.

Item also has getAdjustedPrice, and the printItem method.

These methods will exhibit different behavior, based on the runtime type, and we know that's **polymorphism**.

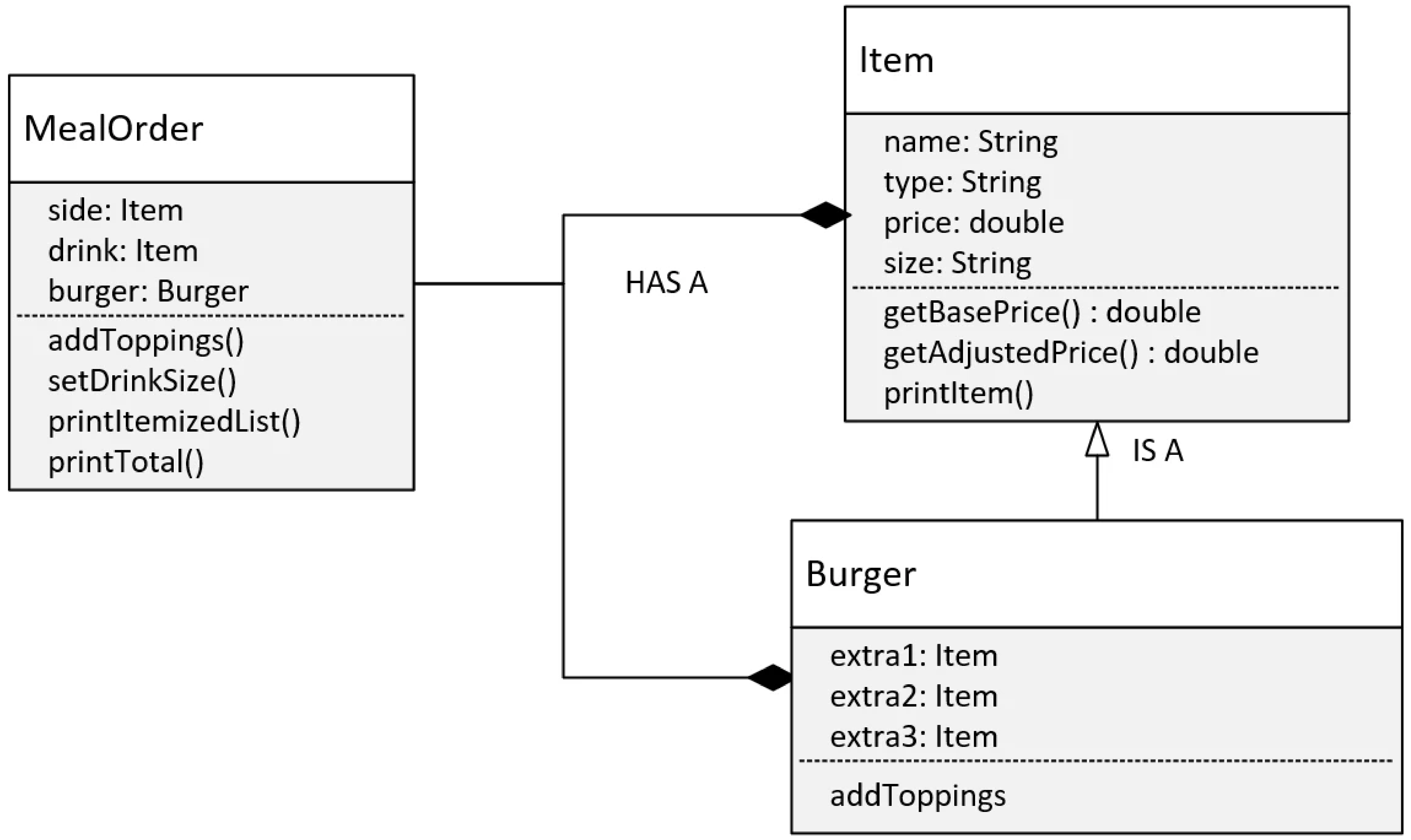


Initial Design Considerations

For the burger, the toppings or extras are individual attributes, and also have the type Item.

We're going to use the MealOrder class, to hide some of the implementation details from the calling code.

This means we're going to use **encapsulation** techniques on MealOrder and Item.


MealOrder
side: Item
drink: Item
burger: Burger
addToppings()
setDrinkSize()
printItemizedList()
printTotal()
HAS A
Item
name: String
type: String
price: double
size: String
getBasePrice() : double
getAdjustedPrice() : double
printItem()
IS A
Burger
extra1: Item
extra2: Item
extra3: Item
addToppings