CSCI-C 322 Assignment 1 – Wade Tubbs – 02/02/2025

Problem 1: Setting Behavior Dynamically

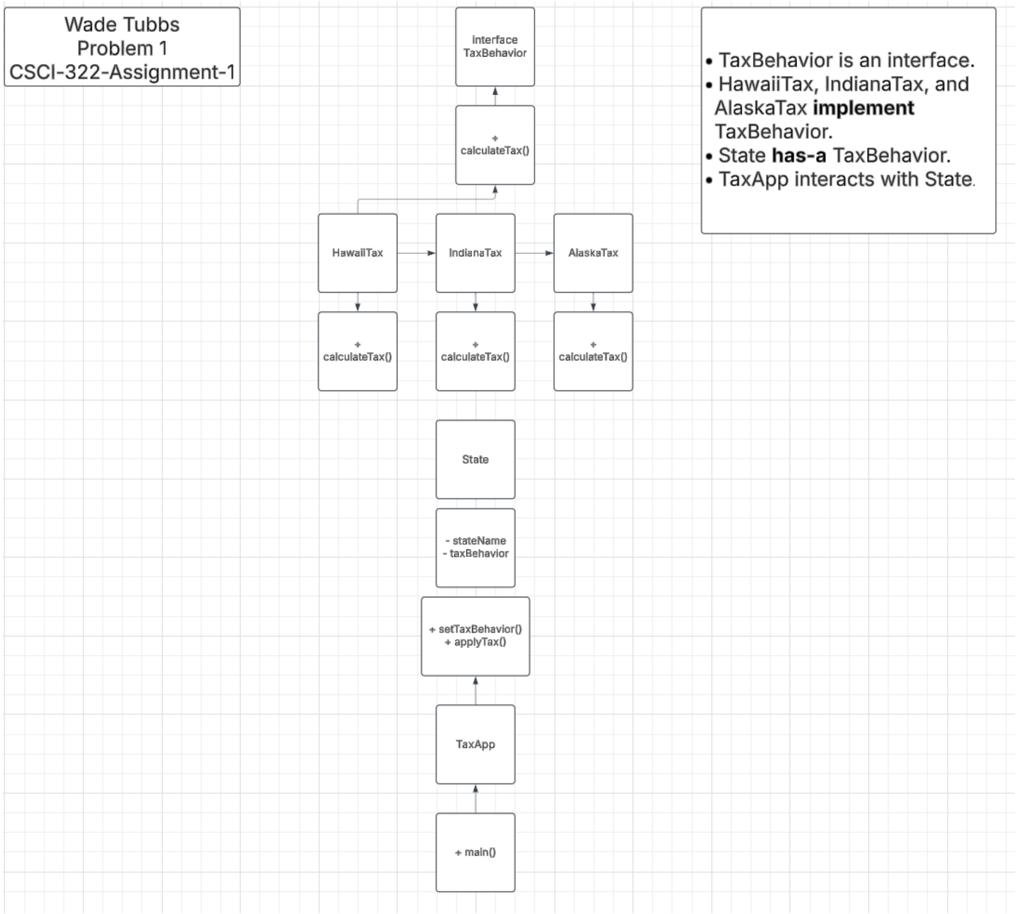
Solution Explanation

For this problem, I implemented a **Strategy Pattern** to dynamically assign sales tax behavior based on the selected state. This approach allows the program to determine the correct tax rate at runtime rather than hardcoding it.

Steps Taken:

1. **Created a TaxBehavior interface** with a calculateTax() method.
2. **Implemented HawaiiTax, IndianaTax, and AlaskaTax classes** that define tax rates for different states.
3. **Created a State class** that stores the state's name and dynamically assigns the correct TaxBehavior.
4. **Developed a TaxApp class** that takes **command-line input** for the state and sale amount, applies the correct tax, and prints the result.

UML:



Problem 2: HAS-A and IS-A Relationships

In the Duck Simulator UML diagram, I classified each relationship as either **HAS-A** or **IS-A**, as explained below:

Arrow #	**Relationship**	**Type**	**Explanation**
1	MallardDuck → Duck	**IS-A**	MallardDuck extends Duck, meaning a MallardDuck **is a** type of Duck.
2	RedheadDuck → Duck	**IS-A**	RedheadDuck extends Duck, meaning a RedheadDuck **is a** type of Duck.
3	RubberDuck → Duck	**IS-A**	RubberDuck extends Duck, meaning a RubberDuck **is a** type of Duck.
4	DecoyDuck → Duck	**IS-A**	DecoyDuck extends Duck, meaning a DecoyDuck **is a** type of Duck.
5	Duck → FlyBehavior	**HAS-A**	Duck contains a reference to a FlyBehavior, meaning a Duck **has a** FlyBehavior.
6	Duck → QuackBehavior	**HAS-A**	Duck contains a reference to a QuackBehavior, meaning a Duck **has a** QuackBehavior.
7	Quack → QuackBehavior	**IS-A**	Quack implements QuackBehavior, meaning Quack **is a** type of QuackBehavior.
8	Squeak → QuackBehavior	**IS-A**	Squeak implements QuackBehavior, meaning Squeak **is a** type of QuackBehavior.
9	MuteQuack → QuackBehavior	**IS-A**	MuteQuack implements QuackBehavior, meaning MuteQuack **is a** type of QuackBehavior.

This classification follows standard **object-oriented programming principles**:

- **IS-A relationships represent inheritance (extends)**
- **HAS-A relationships represent composition (object references)**

Problem 3: Observer Design Pattern (Auction System)

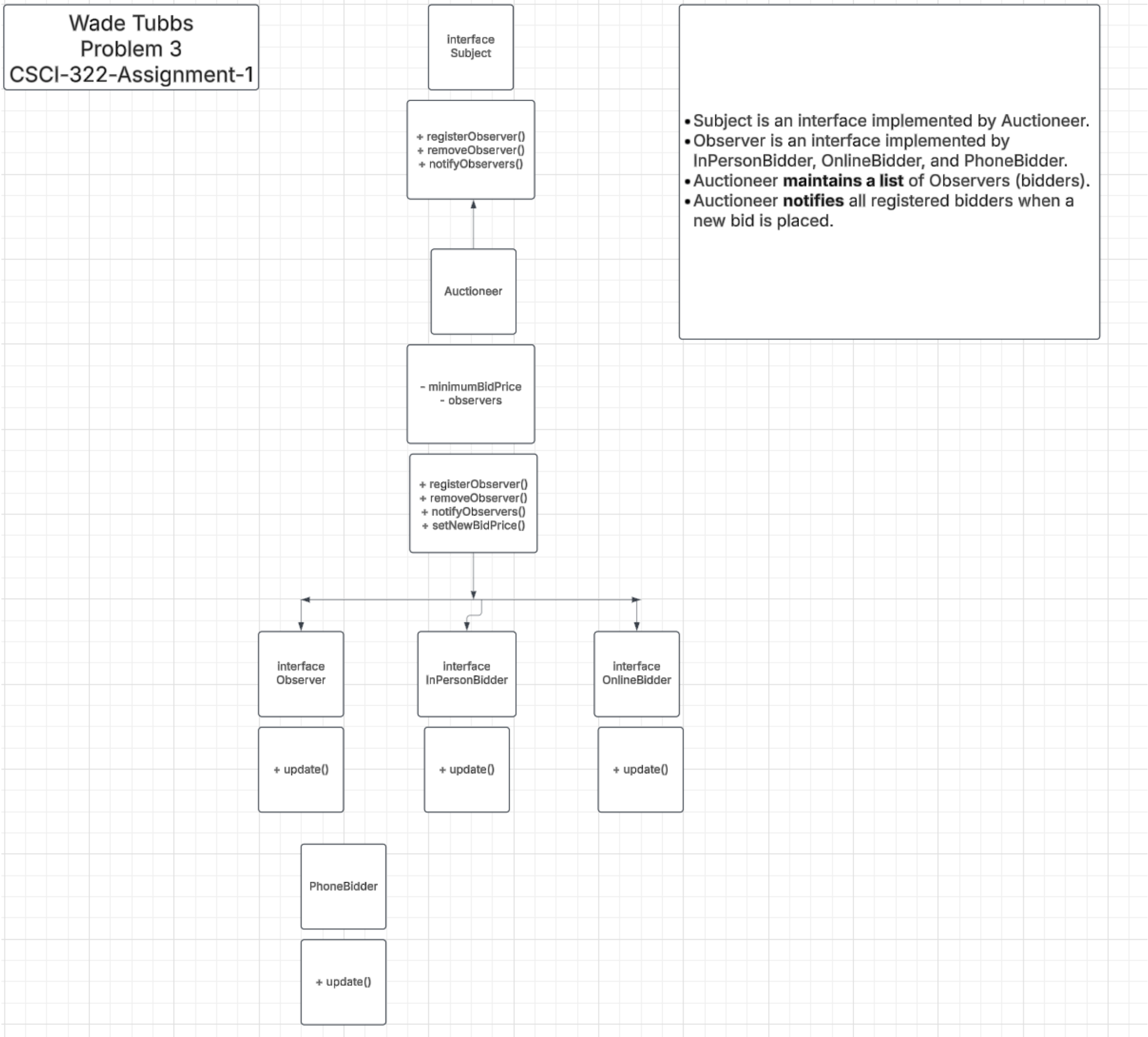
Solution Explanation

This problem requires implementing the **Observer Design Pattern**, which is useful when multiple objects (bidders) need to be notified of state changes (new bid prices).

Steps Taken:

1. **Created an Observer interface** with an update(newBidPrice) method.
2. **Created a Subject interface** for registering, removing, and notifying observers.
3. **Developed an Auctioneer class** (concrete subject) that manages the list of bidders.
4. **Implemented InPersonBidder, OnlineBidder, and PhoneBidder classes**, which receive notifications when a new bid is placed.

UML:



- Subject is an interface implemented by Auctioneer.
- Observer is an interface implemented by InPersonBidder, OnlineBidder, and PhoneBidder.
- Auctioneer **maintains a list** of Observers (bidders).
- Auctioneer **notifies** all registered bidders when a new bid is placed.