use Adobe Reader to complete

(Type in fields)

PA4 – Visitor,Observer,State

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: [✓ Yes] No

Name:	Joey Domino
Date:	3/3/2022

Submission Details

Final ***Changelist*** number:	135833
Verified build:	[✓ Yes] No
Required Configurations:	DEBUG only
Test Passed:	4

Discussion (What did you learn):

I learned:
- How to set up derived classes
- How to handle transition states between different tree structures

Finally! Another assignment I finished quickly and with relative ease.

Verify Builds

- Follow the Piazza procedure on submission
    - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
    - No – Generated files
        - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db, *.user
        - Anything that is generated by the compiler should not be included
    - No – Generated directories
        - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
    - *.sln, *.csproj, *.cs,
    - App.config, AssemblyInfo.cs, CleanMe.bat
    - Resources Directory:
        - *.tga, *.dll, *.wav, *.glsl, *.azul

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
    - As soon as you get something working, submit to perforce
    - Have reasonable check-in comments
        - Points will be deducted if minimum is not reached

Submission Report

- Fill out the submission Report
    - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
    - Warning level 4
    - NO Warnings or ERRORS
        - Your code should be squeaky clean.
    - Code needs to work "as-is".
        - No modifications to files or deleting files necessary to compile or run.
    - All your code must compile from perforce with no modifications.
        - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason…
    - It will not be graded and you get a 0

No Containers

- Containers (No automatic containers or arrays
- Template or generic parameters
- No arrays
  - You need to do this the old fashion way - ***YOU EARNED IT***

Leave Project Settings

- Do NOT change the project or warning level
  - Any changing of level or suppression of warnings is an integrity issue

Simple C#

- No .Net
- We are using the basics
  - Types:
    - Class, Structs, intrinsic types (int, float, bool, etc…)
    - NO arrays allowed!
  - Basics language features
    - Inheritance, methods, abstract, virtual, etc…

No Debug code or files disabled

- Make sure the program has only active code
  - If you added debug code or commented out code,
    - please return to code to active state or remove it

DO NOT Adding files to this project

- No adding of files… live inside the project

Due Dates

- See Piazza for due date and time
- Submit program perforce in your student directory assignment supplied.
- Fill out your this ***Submission Report*** and commit to perforce
  - ***ONLY*** use Adobe Reader to fill out form, all others will be rejected.
  - Fill out the form and discussion for full credit.

Goals

- Learn
    - Design Patterns
        - Visitor Pattern
        - Observer Pattern
        - State Pattern

Assignments

General:

- Look at notes / lecture for Design Patterns
- Additional useful links
    - https://www.oodesign.com/
    - https://www.dofactory.com/net/design-patterns
    - https://sourcemaking.com/design_patterns
    - https://en.wikipedia.org/wiki/Design_Patterns
    - https://en.wikipedia.org/wiki/Software_design_pattern
    - https://refactoring.guru/design-patterns
- Books
    - Head First Design Patterns:Building Extensible & Maintainable Object-Oriented Software
    - Design Patterns: Elements of Reusable Object-Oriented Software

NOTE:

- This was a very difficult set of patterns to TEST
- The way I can verify that the pattern is working correctly is with a registration class.
    - It marks what class/method/state your code is in…
    - I verify these markers in unit test.

For example:

- In the Obsever class… you have a obsever – PlaySound
- You need to add a mailbox marker

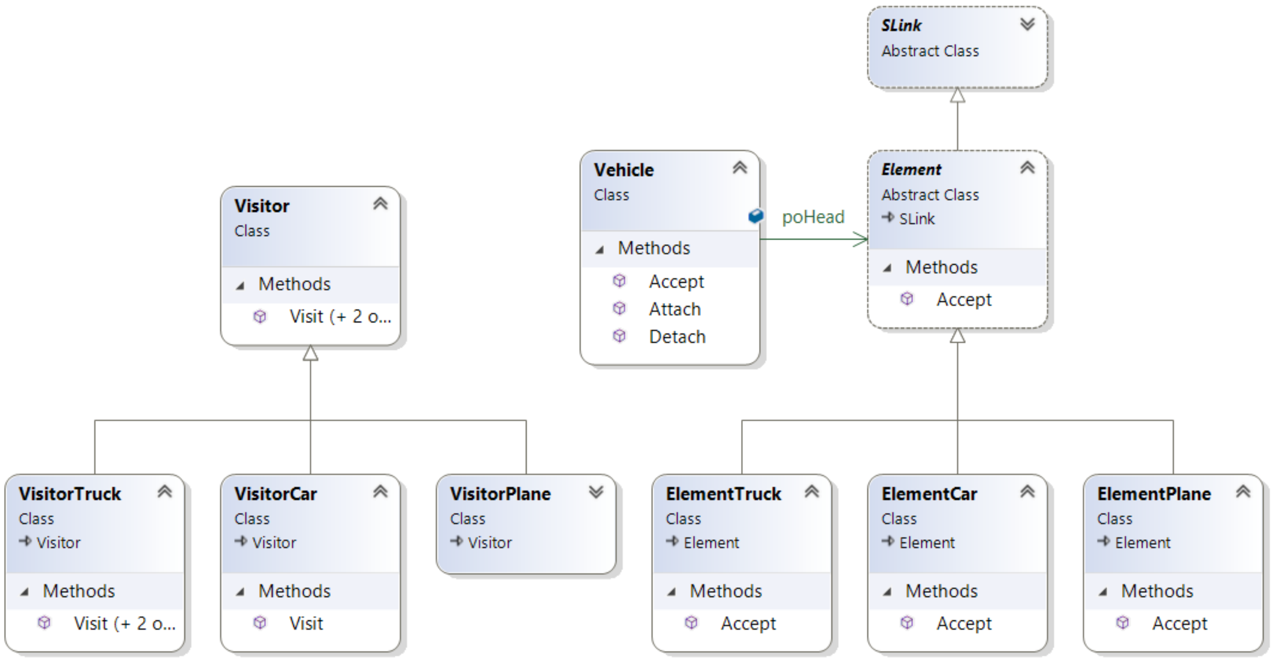
```
public override void Notify()
{
    MailBox_Observer.Register(MailBox_Observer.Status.PLAY_SOUND_OBSERVER);
}
```

Follow the instructions:

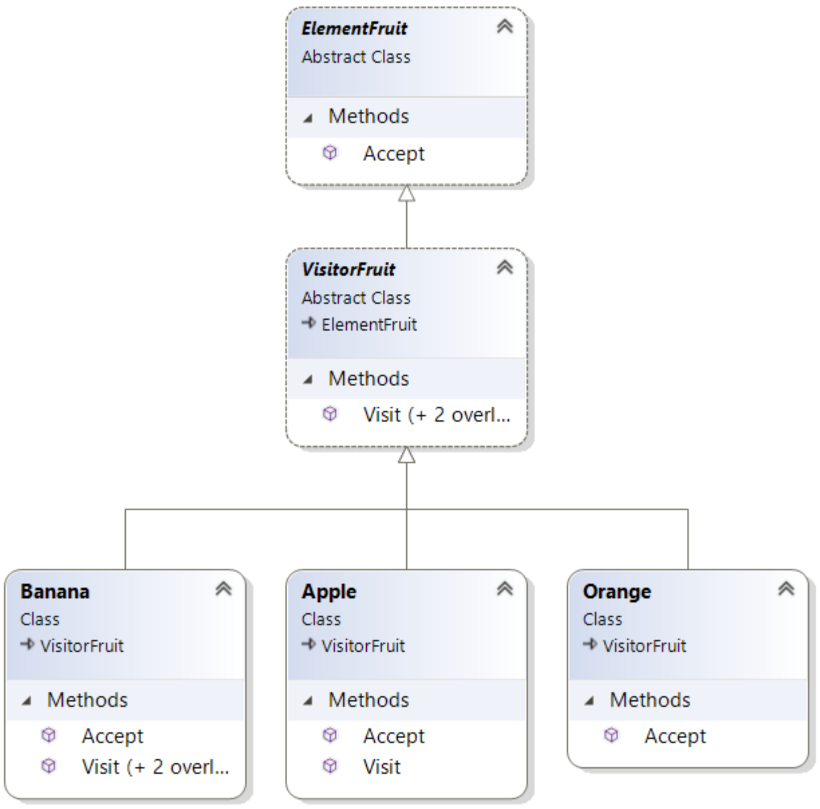
- Add the appropriate mailbox markers in your methods.

Problems:

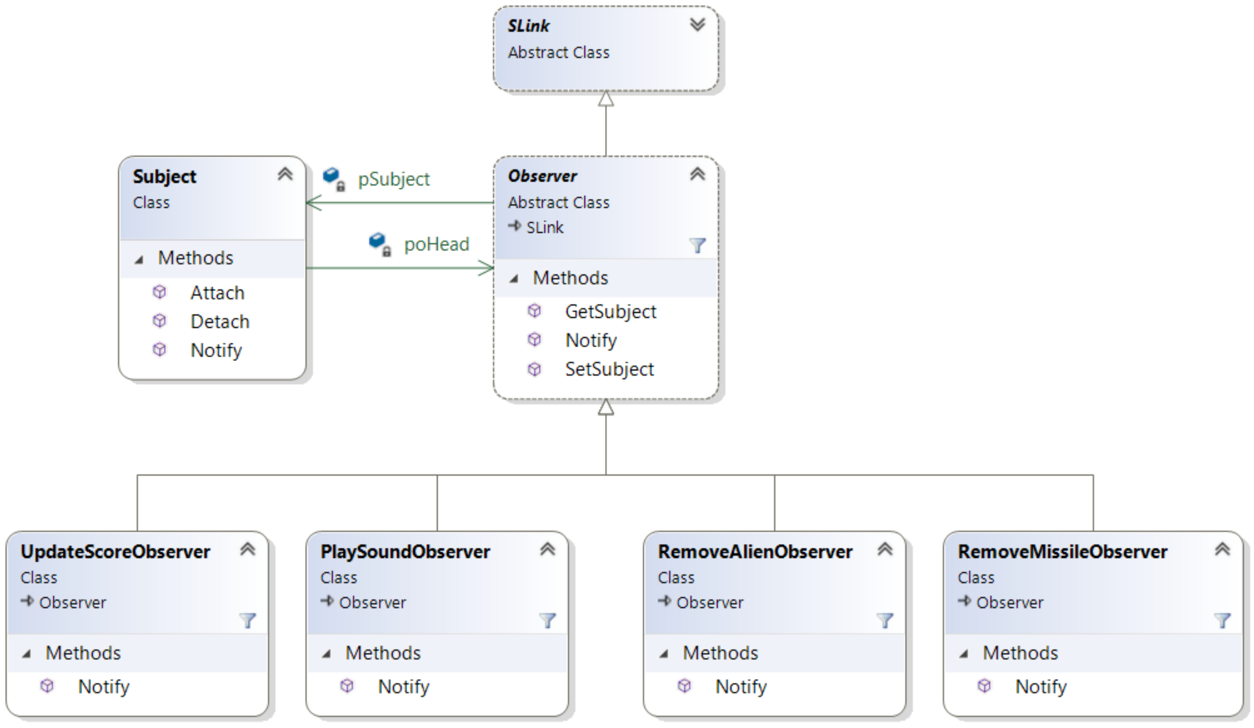
- **Visitor Pattern** - Standard
    - Attach visitors to the vehicle using Single Linked List
        - Add to front of list on attachment
    - Add ***MailBox_StandardVisitor*** markers to the visitor methods



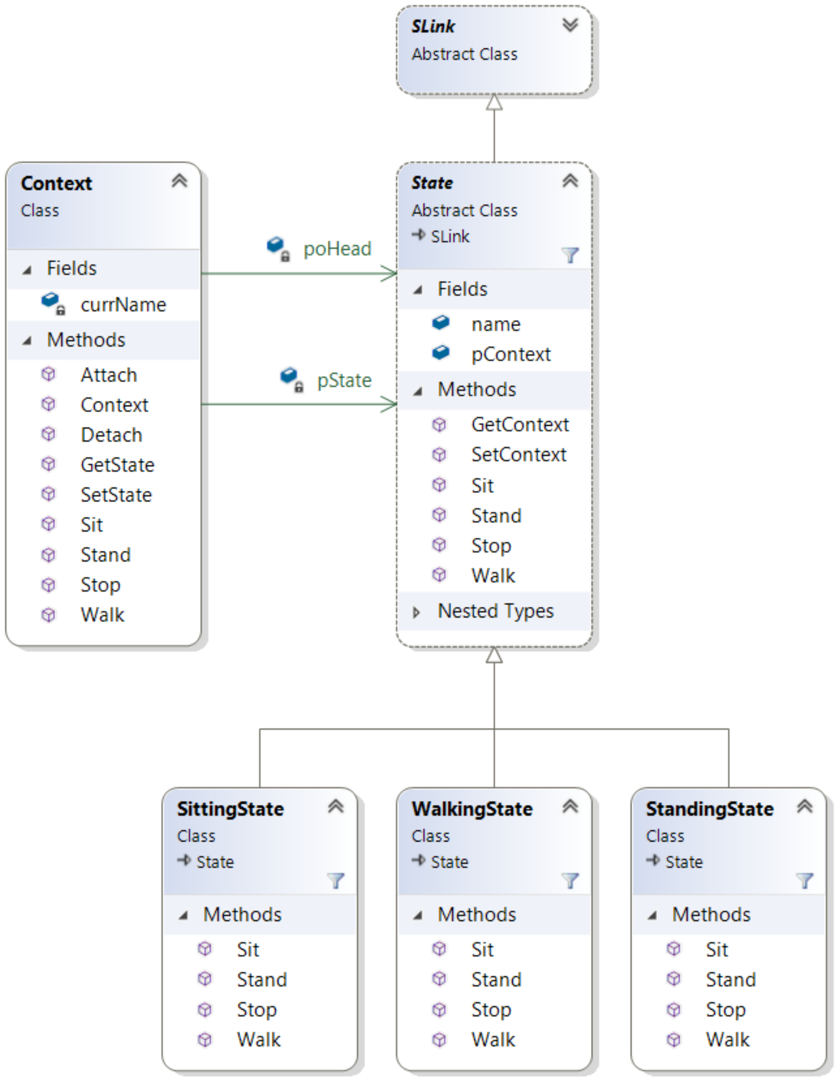
- **Visitor Pattern – Collision** (condensed)
  - Given that we don't need to dynamically add/remove visitors…
    - Our collision system defines the visitor / elements at compile time
    - This pattern is the same as the Standard Visitor pattern, but it doesn't have attach/detach functionality.
    - With this simplified approach…
      - both the element and visitor can be in the same class.
    - Look at sample code from class
  - Add ***MailBox_CollisionVisitor*** markers to the visitor methods



- **Observer Pattern**
  - Attach observers to the subject using Single Linked List
    - Add to front of list on attachment
  - Add ***MailBox_Observer*** markers to the observer methods



- **State Pattern**
    - We need a way to store inactive states…
        - We are using attach/detach to store inactive states
    - Attach all reference states to the context using Single Linked List
        - Add to front of list on attachment
    - Implement the state pattern for the following states:
        - Sitting
        - Standing
        - Walking
    - Each state has the following methods
        - Sit()
            - Transition:
                - If in Standing state, Sit() will switch to Sitting state
        - Stand()
            - Transition:
                - If in Sitting state, Stand() will switch to Standing state
        - Walk()
            - Transition:
                - If in Standing state, Walk() will switch to Walking state
        - Stop()
            - Transition:
                - If in Walking state, Stop() will switch to Standing state
    - Add two mailbox registrations to each state method (sit,stand,walk,stop)
        - Add ***MailBox_StateMethod*** markers to the state methods
        - Add ***MailBox_StateTransition*** markers to the state methods



General guidelines:

- Idea is to get you comfortable with these patterns
  - You will include these concepts into the Space Invaders project
- Create UML diagrams to help
  - Post on Piazza questions and clarifications
- No need to add any files… the unit tests are fully stubbed out

Make sure you delete these using directives (we are not using them)

- `using System.Collections.Generic;`
- `using System.Linq;`
- `using System.Text;`
- `using System.Threading.Tasks;`

Development

- Store project in student directory in perforce
- Do your work in the supplied PA4 project

Submission

- Submit your PA4 directory into perforce:
    - /student/<yourname>/PA4/...
        - You need to submit a complete C# project
    - Solution, project and C# files (whatever it takes to build the project)
        - Do not submit anything that is auto generated
    - Run the supplied CleanMe.bat before submission
        - Should cleanup files
- Fill out the Submission report and submit that pdf to your student directory

Validation

Simple checklist to make sure that everything is submitted correctly

- Is the project compiling and running without any errors or warnings?
- Does the project runs ***ALL*** without crashing?
- Is the submission report filled in and submitted to perforce?
- Follow the verification process for perforce
    - Is all the code there and compiles "as-is"?
    - No extra files

Hints

Most assignments will have hints in a section like this.

- Do one design pattern at a time
    - Look up the pattern
    - See some reference code
        - I like *oodesign* and *dofactory* reference
- You code might be very small…
    - You might think "that's it".
        - Understand what the pattern is doing… why its doing x behavior
    - I created semi-real examples… so there is a lot of code to give the environment
        - But in some cases, you just fill in one or two methods

Troubleshooting

- Print, print, print
    - Draw diagrams to help you understand
- Have fun… this shouldn't be stressful
    - Slow and steady discovery and development will get you there.
    - Its not hard… just different way of solving problems
        - Embrace the pattern concept