SWE2410-111 – Lab 4: Tourist Observer – Jawadul Chowdhury – 10/1/2024

Objectives

State your objectives for the lab here: what is the lab supposed to do in your own words. It should be a short paragraph.

For this lab, an additional two challenges must be implemented. The first challenge to be implemented would be the MSOE challenge, and the other challenge to be implemented would be the BUS challenge. These two challenges must be implemented using the observer pattern.

Requirements

Capture the detailed requirements here. Look at the grading rubrics, the lab assignment instructions to make sure you capture what you must do. Also, put in any requirements that you have added.

The main requirements for the lab are listed as follows:

- For the MSOE challenge, the main character must go around the map to tag cars which have license plates. The goal is to spell out the phrase "MSOE", by picking out matching characters from the license plates of the cars.

- The next challenge is the BUS challenge, where the user must first tag a bus for the challenge to appear on the user interface. Once the challenge appears, the user must tag the cars and obtain characters from their license plates to spell out the phrase "BUS". The added conditions are that if the player finds the S before activating the BUS challenge, the S needs to be obtained again from the car plates.

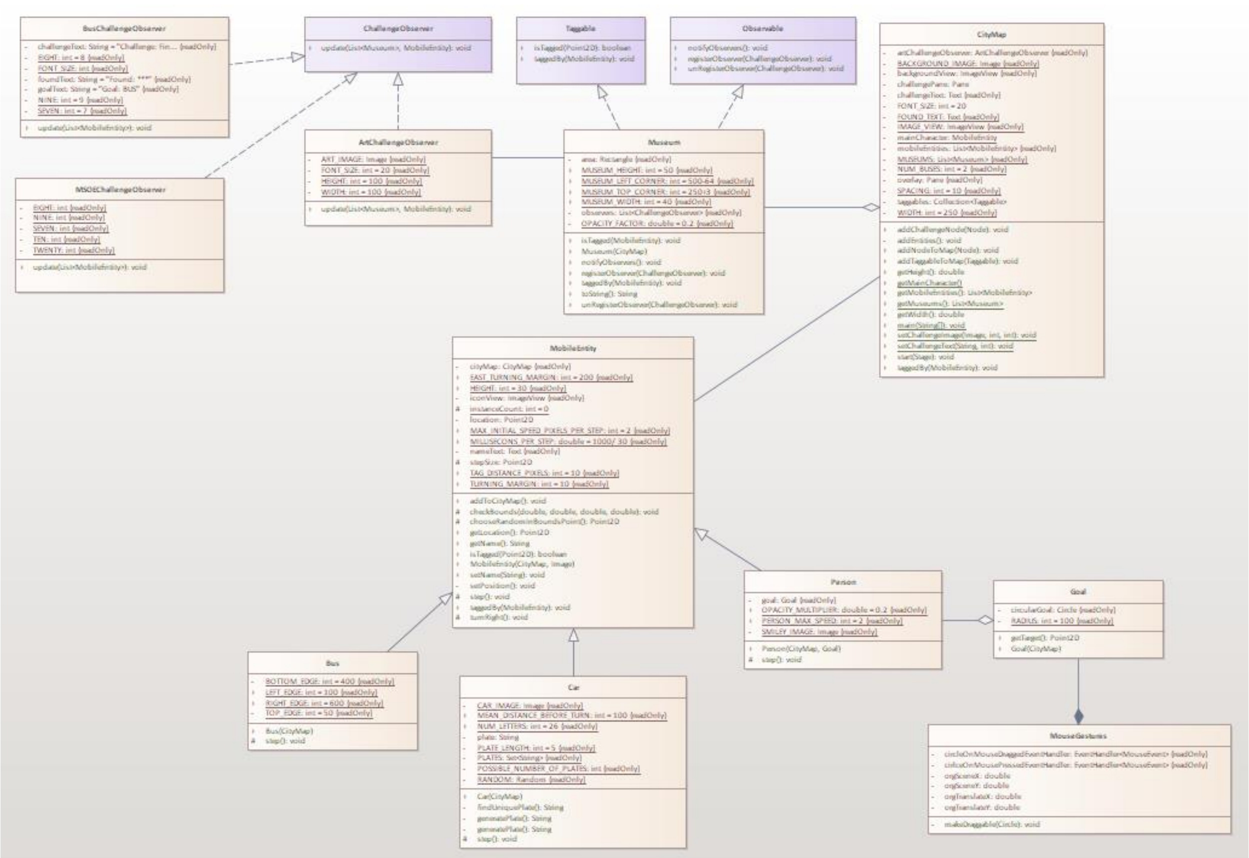
The additional requirements for the lab are listed as follows:

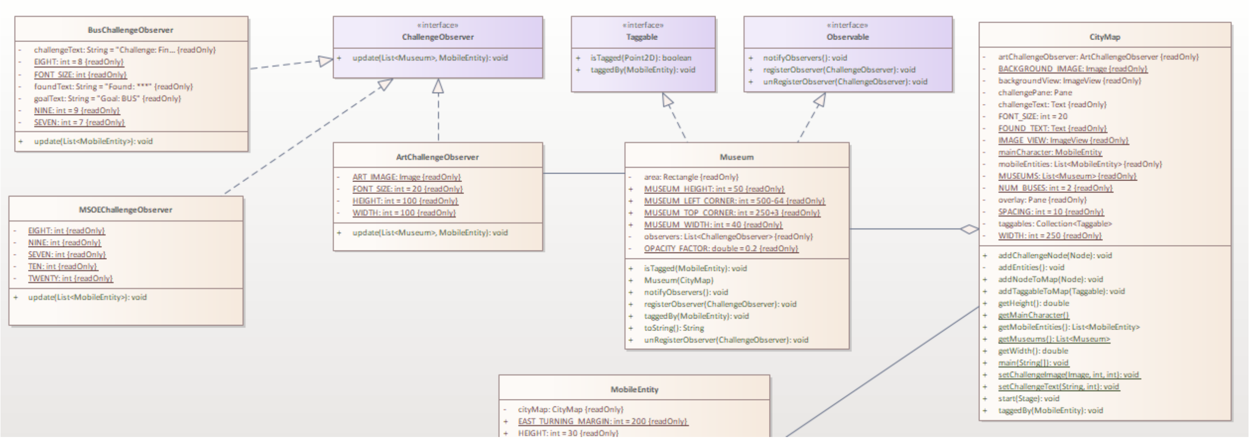
- Creating a sketch of the solution diagram, to show how the additional two challenges are implemented along with the extra challenge.
- Like the previous lab, ensure that the observer pattern is implemented correctly. This mean keeping the observer and the subject separate from each other
- The original classes within the repository should have as less edits as possible, and this is because since we're implementing the observer pattern, not many edits should be required.

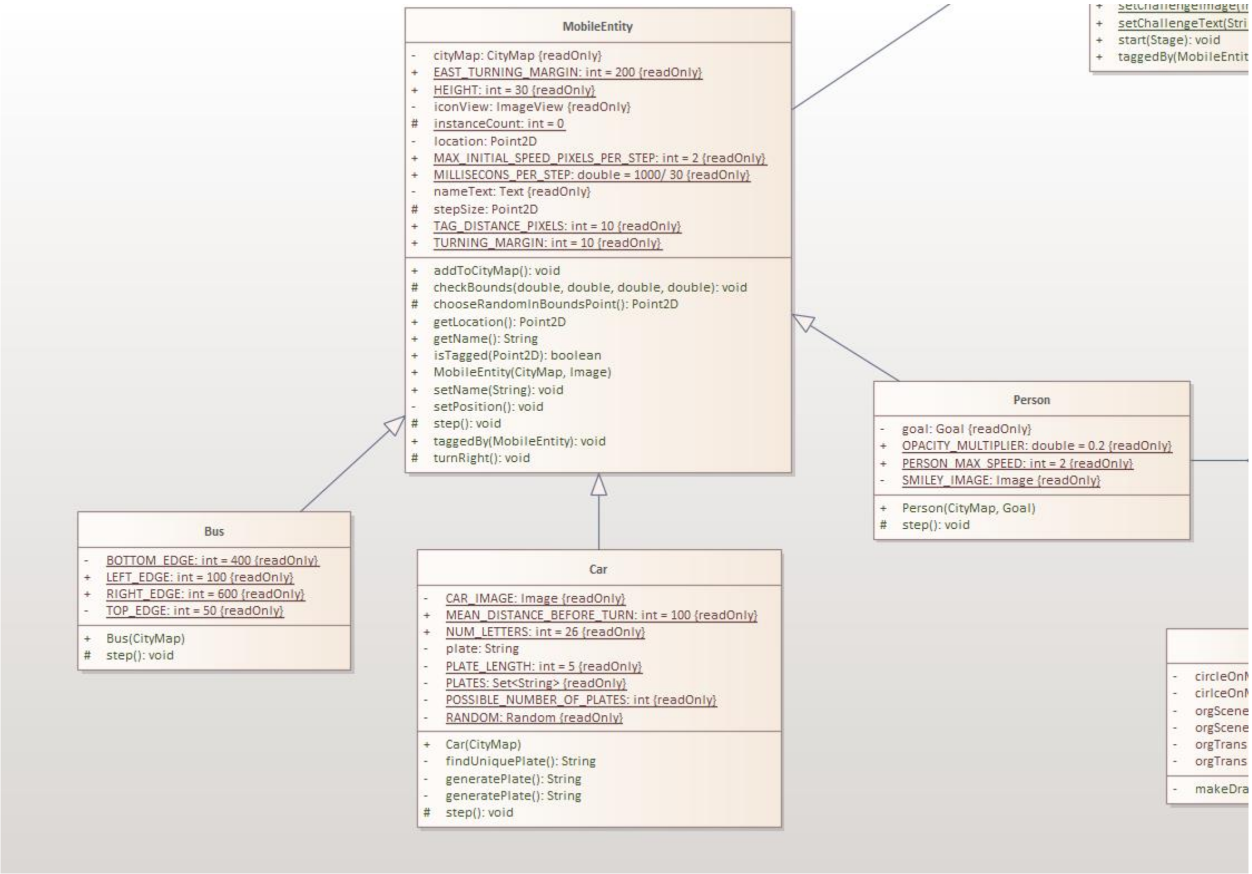
Design

Include appropriate design elements, such as UML diagrams which form the mental model of how it is intended to achieve the objectives. Sequence diagrams are also accepted.

This is a UML diagram of how the observer pattern was implemented:


BusChallengeObserver
- challengeText: String = "Challenge: Fin... {readOnly}
- EIGHT: int = 8 {readOnly}
- FONT_SIZE: int {readOnly}
- foundText: String = "Found: ***" {readOnly}
- goalText: String = "Goal: BUS" {readOnly}
- NINE: int = 9 {readOnly}
- SEVEN: int = 7 {readOnly}
+ update(List<MobileEntity>): void
ChallengeObserver
+ update(List<Museum>, MobileEntity): void
Taggable
+ isTagged(Point2D): boolean
+ taggedBy(MobileEntity): void
Observable
+ notifyObservers(): void
+ registerObserver(ChallengeObserver): void
+ unRegisterObserver(ChallengeObserver): void
CityMap
- artChallengeObserver: ArtChallengeObserver {readOnly}
- BACKGROUND_IMAGE: Image {readOnly}
- backgroundView: ImageView {readOnly}
- challengePane: Pane
- challengeText: Text {readOnly}
- FONT_SIZE: int = 20
- FOUND_TEXT: Text {readOnly}
- IMAGE_VIEW: ImageView {readOnly}
- mainCharacter: MobileEntity
- mobileEntities: List<MobileEntity> {readOnly}
- MUSEUMS: List<Museum> {readOnly}
- NUM_BUSES: int = 2 {readOnly}
- overlay: Pane {readOnly}
- SPACING: int = 10 {readOnly}
- taggables: Collection<Taggable>
- WIDTH: int = 250 {readOnly}
+ addChallengeNode(Node): void
- addEntities(): void
+ addNodeToMap(Node): void
+ addTaggableToMap(Taggable): void
+ getHeight(): double
+ getMainCharacter()
+ getMobileEntities(): List<MobileEntity>
+ getMuseums(): List<Museum>
+ getWidth(): double
+ main(String[]): void
+ setChallengeImage(Image, int, int): void
+ setChallengeText(String, int): void
+ start(Stage): void
+ taggedBy(MobileEntity): void
ArtChallengeObserver
- ART_IMAGE: Image {readOnly}
- FONT_SIZE: int = 20 {readOnly}
- HEIGHT: int = 100 {readOnly}
- WIDTH: int = 100 {readOnly}
+ update(List<Museum>, MobileEntity): void
Museum
- area: Rectangle {readOnly}
+ MUSEUM_HEIGHT: int = 50 {readOnly}
+ MUSEUM_LEFT_CORNER: int = 500-64 {readOnly}
+ MUSEUM_TOP_CORNER: int = 250+3 {readOnly}
+ MUSEUM_WIDTH: int = 40 {readOnly}
- observers: List<ChallengeObserver> {readOnly}
- OPACITY_FACTOR: double = 0.2 {readOnly}
+ isTagged(MobileEntity): void
+ Museum(CityMap)
+ notifyObservers(): void
+ registerObserver(ChallengeObserver): void
+ taggedBy(MobileEntity): void
+ toString(): String
+ unRegisterObserver(ChallengeObserver): void
MSOEChallengeObserver
- EIGHT: int {readOnly}
- NINE: int {readOnly}
- SEVEN: int {readOnly}
- TEN: int {readOnly}
- TWENTY: int {readOnly}
+ update(List<MobileEntity>): void
MobileEntity
- cityMap: CityMap {readOnly}
+ EAST_TURNING_MARGIN: int = 200 {readOnly}
+ HEIGHT: int = 30 {readOnly}
- iconView: ImageView {readOnly}
# instanceCount: int = 0
- location: Point2D
+ MAX_INITIAL_SPEED_PIXELS_PER_STEP: int = 2 {readOnly}
+ MILLISECONS_PER_STEP: double = 1000/ 30 {readOnly}
- nameText: Text {readOnly}
# stepSize: Point2D
+ TAG_DISTANCE_PIXELS: int = 10 {readOnly}
+ TURNING_MARGIN: int = 10 {readOnly}
+ addToCityMap(): void
# checkBounds(double, double, double, double): void
# chooseRandomInBoundsPoint(): Point2D
+ getLocation(): Point2D
+ getName(): String
+ isTagged(Point2D): boolean
+ MobileEntity(CityMap, Image)
+ setName(String): void
+ setPosition(): void
# step(): void
+ taggedBy(MobileEntity): void
# turnRight(): void
Person
- goal: Goal {readOnly}
+ OPACITY_MULTIPLIER: double = 0.2 {readOnly}
+ PERSON_MAX_SPEED: int = 2 {readOnly}
- SMILEY_IMAGE: Image {readOnly}
+ Person(CityMap, Goal)
# step(): void
Goal
- circularGoal: Circle {readOnly}
+ RADIUS: int = 100 {readOnly}
+ getTarget(): Point2D
+ Goal(CityMap)
Bus
- BOTTOM_EDGE: int = 400 {readOnly}
+ LEFT_EDGE: int = 100 {readOnly}
+ RIGHT_EDGE: int = 600 {readOnly}
- TOP_EDGE: int = 50 {readOnly}
+ Bus(CityMap)
# step(): void
Car
- CAR_IMAGE: Image {readOnly}
+ MEAN_DISTANCE_BEFORE_TURN: int = 100 {readOnly}
+ NUM_LETTERS: int = 26 {readOnly}
- plate: String
+ PLATE_LENGTH: int = 5 {readOnly}
- PLATES: Set<String> {readOnly}
- POSSIBLE_NUMBER_OF_PLATES: int {readOnly}
- RANDOM: Random {readOnly}
+ Car(CityMap)
- findUniquePlate(): String
- generatePlate(): String
- generatePlate(): String
# step(): void
MouseGestures
- circleOnMouseDraggedEventHandler: EventHandler<MouseEvent> {readOnly}
- circleOnMousePressedEventHandler: EventHandler<MouseEvent> {readOnly}
- orgSceneX: double
- orgSceneY: double
- orgTranslateX: double
- orgTranslateY: double
- makeDraggable(Circle): void


BusChallengeObserver
- challengeText: String = "Challenge: Fin... {readOnly}
- EIGHT: int = 8 {readOnly}
- FONT_SIZE: int {readOnly}
- foundText: String = "Found: ***" {readOnly}
- goalText: String = "Goal: BUS" {readOnly}
- NINE: int = 9 {readOnly}
- SEVEN: int = 7 {readOnly}
+ update(List<MobileEntity>): void
«interface»
ChallengeObserver
+ update(List<Museum>, MobileEntity): void
«interface»
Taggable
+ isTagged(Point2D): boolean
+ taggedBy(MobileEntity): void
«interface»
Observable
+ notifyObservers(): void
+ registerObserver(ChallengeObserver): void
+ unRegisterObserver(ChallengeObserver): void
CityMap
- artChallengeObserver: ArtChallengeObserver {readOnly}
- BACKGROUND_IMAGE: Image {readOnly}
- backgroundView: ImageView {readOnly}
- challengePane: Pane
- challengeText: Text {readOnly}
- FONT_SIZE: int = 20
- FOUND_TEXT: Text {readOnly}
- IMAGE_VIEW: ImageView {readOnly}
- mainCharacter: MobileEntity
- mobileEntities: List<MobileEntity> {readOnly}
- MUSEUMS: List<Museum> {readOnly}
- NUM_BUSES: int = 2 {readOnly}
- overlay: Pane {readOnly}
- SPACING: int = 10 {readOnly}
- taggables: Collection<Taggable>
- WIDTH: int = 250 {readOnly}
+ addChallengeNode(Node): void
- addEntities(): void
+ addNodeToMap(Node): void
+ addTaggableToMap(Taggable): void
+ getHeight(): double
+ getMainCharacter()
+ getMobileEntities(): List<MobileEntity>
+ getMuseums(): List<Museum>
+ getWidth(): double
+ main(String[]): void
+ setChallengeImage(Image, int, int): void
+ setChallengeText(String, int): void
+ start(Stage): void
+ taggedBy(MobileEntity): void
ArtChallengeObserver
- ART_IMAGE: Image {readOnly}
- FONT_SIZE: int = 20 {readOnly}
- HEIGHT: int = 100 {readOnly}
- WIDTH: int = 100 {readOnly}
+ update(List<Museum>, MobileEntity): void
Museum
- area: Rectangle {readOnly}
+ MUSEUM_HEIGHT: int = 50 {readOnly}
+ MUSEUM_LEFT_CORNER: int = 500-64 {readOnly}
+ MUSEUM_TOP_CORNER: int = 250+3 {readOnly}
+ MUSEUM_WIDTH: int = 40 {readOnly}
- observers: List<ChallengeObserver> {readOnly}
- OPACITY_FACTOR: double = 0.2 {readOnly}
+ isTagged(MobileEntity): void
+ Museum(CityMap)
+ notifyObservers(): void
+ registerObserver(ChallengeObserver): void
+ taggedBy(MobileEntity): void
+ toString(): String
+ unRegisterObserver(ChallengeObserver): void
MSOEChallengeObserver
- EIGHT: int {readOnly}
- NINE: int {readOnly}
- SEVEN: int {readOnly}
- TEN: int {readOnly}
- TWENTY: int {readOnly}
+ update(List<MobileEntity>): void
MobileEntity
- cityMap: CityMap {readOnly}
+ EAST_TURNING_MARGIN: int = 200 {readOnly}
+ HEIGHT: int = 30 {readOnly}



```
+   setChallengeText(Stri
+   start(Stage): void
+   taggedBy(MobileEntit
```

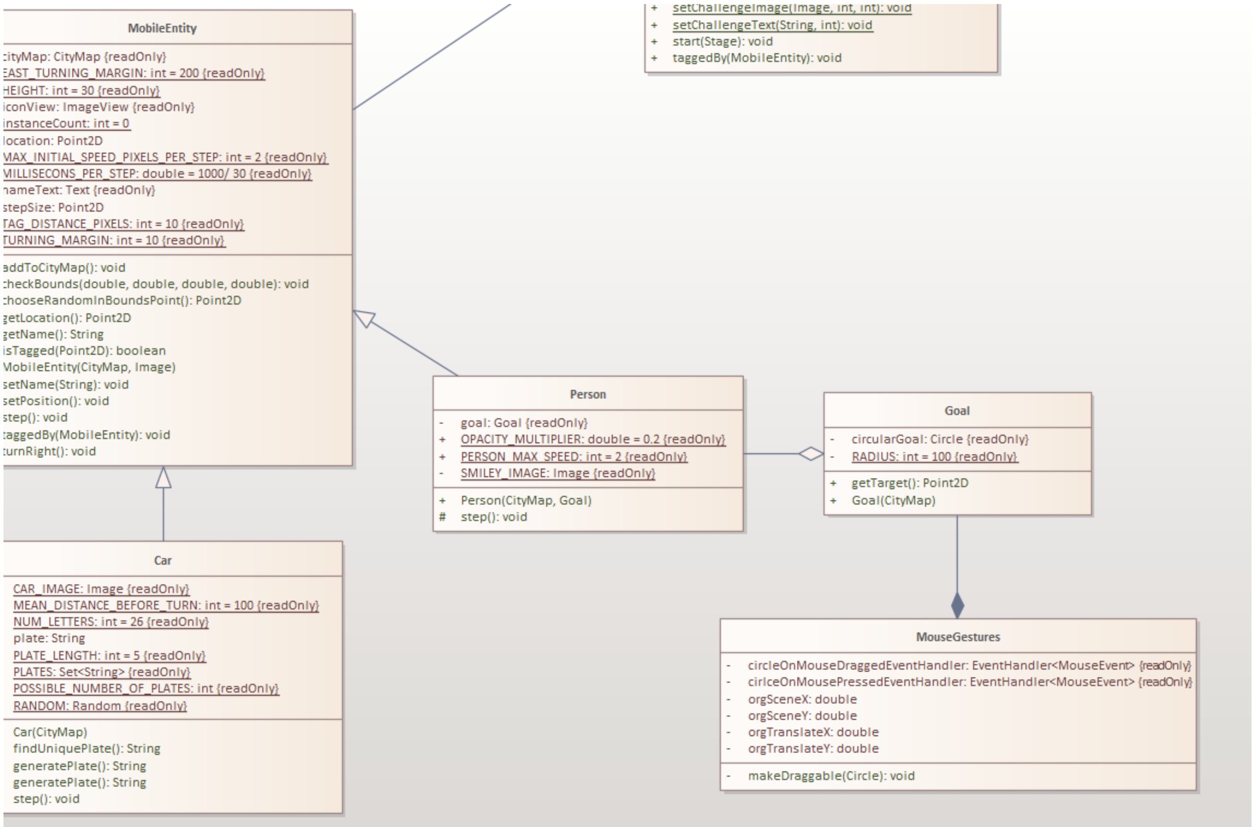
```
MobileEntity

-   cityMap: CityMap {readOnly}
+   EAST_TURNING_MARGIN: int = 200 {readOnly}
+   HEIGHT: int = 30 {readOnly}
-   iconView: ImageView {readOnly}
#   instanceCount: int = 0
-   location: Point2D
+   MAX_INITIAL_SPEED_PIXELS_PER_STEP: int = 2 {readOnly}
+   MILLISECONS_PER_STEP: double = 1000/ 30 {readOnly}
-   nameText: Text {readOnly}
#   stepSize: Point2D
+   TAG_DISTANCE_PIXELS: int = 10 {readOnly}
+   TURNING_MARGIN: int = 10 {readOnly}

+   addToCityMap(): void
#   checkBounds(double, double, double, double): void
#   chooseRandomInBoundsPoint(): Point2D
+   getLocation(): Point2D
+   getName(): String
+   isTagged(Point2D): boolean
+   MobileEntity(CityMap, Image)
+   setName(String): void
-   setPosition(): void
#   step(): void
+   taggedBy(MobileEntity): void
#   turnRight(): void
```

```
Person

-   goal: Goal {readOnly}
+   OPACITY_MULTIPLIER: double = 0.2 {readOnly}
+   PERSON_MAX_SPEED: int = 2 {readOnly}
-   SMILEY_IMAGE: Image {readOnly}

+   Person(CityMap, Goal)
#   step(): void
```

```
Bus

-   BOTTOM_EDGE: int = 400 {readOnly}
+   LEFT_EDGE: int = 100 {readOnly}
+   RIGHT_EDGE: int = 600 {readOnly}
-   TOP_EDGE: int = 50 {readOnly}

+   Bus(CityMap)
#   step(): void
```

```
Car

-   CAR_IMAGE: Image {readOnly}
+   MEAN_DISTANCE_BEFORE_TURN: int = 100 {readOnly}
+   NUM_LETTERS: int = 26 {readOnly}
-   plate: String
-   PLATE_LENGTH: int = 5 {readOnly}
-   PLATES: Set<String> {readOnly}
-   POSSIBLE_NUMBER_OF_PLATES: int {readOnly}
-   RANDOM: Random {readOnly}

+   Car(CityMap)
-   findUniquePlate(): String
-   generatePlate(): String
-   generatePlate(): String
#   step(): void
```

```
-   circleOnM
-   cirlceOnM
-   orgScene
-   orgScene
-   orgTrans
-   orgTrans

-   makeDra
```



```
+   setChallengeText(String, int): void
+   start(Stage): void
+   taggedBy(MobileEntity): void
```

```
MobileEntity

:ityMap: CityMap {readOnly}
EAST_TURNING_MARGIN: int = 200 {readOnly}
HEIGHT: int = 30 {readOnly}
iconView: ImageView {readOnly}
instanceCount: int = 0
location: Point2D
MAX_INITIAL_SPEED_PIXELS_PER_STEP: int = 2 {readOnly}
MILLISECONS_PER_STEP: double = 1000/ 30 {readOnly}
nameText: Text {readOnly}
stepSize: Point2D
TAG_DISTANCE_PIXELS: int = 10 {readOnly}
TURNING_MARGIN: int = 10 {readOnly}

addToCityMap(): void
:heckBounds(double, double, double, double): void
:hooseRandomInBoundsPoint(): Point2D
getLocation(): Point2D
getName(): String
isTagged(Point2D): boolean
MobileEntity(CityMap, Image)
setName(String): void
setPosition(): void
step(): void
:aggedBy(MobileEntity): void
:urnRight(): void
```

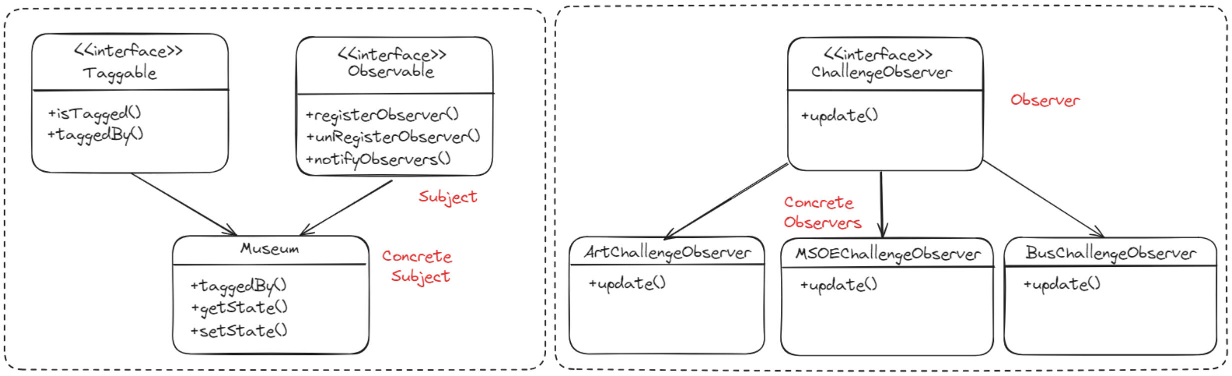
```
Person

-   goal: Goal {readOnly}
+   OPACITY_MULTIPLIER: double = 0.2 {readOnly}
+   PERSON_MAX_SPEED: int = 2 {readOnly}
-   SMILEY_IMAGE: Image {readOnly}

+   Person(CityMap, Goal)
#   step(): void
```

```
Goal

-   circularGoal: Circle {readOnly}
-   RADIUS: int = 100 {readOnly}

+   getTarget(): Point2D
+   Goal(CityMap)
```

```
Car

CAR_IMAGE: Image {readOnly}
MEAN_DISTANCE_BEFORE_TURN: int = 100 {readOnly}
NUM_LETTERS: int = 26 {readOnly}
plate: String
PLATE_LENGTH: int = 5 {readOnly}
PLATES: Set<String> {readOnly}
POSSIBLE_NUMBER_OF_PLATES: int {readOnly}
RANDOM: Random {readOnly}

Car(CityMap)
findUniquePlate(): String
generatePlate(): String
generatePlate(): String
step(): void
```

```
MouseGestures

-   circleOnMouseDraggedEventHandler: EventHandler<MouseEvent> {readOnly}
-   cirlceOnMousePressedEventHandler: EventHandler<MouseEvent> {readOnly}
-   orgSceneX: double
-   orgSceneY: double
-   orgTranslateX: double
-   orgTranslateY: double

-   makeDraggable(Circle): void
```

This is sketch of the Observer Pattern, and how it is going to be implemented:



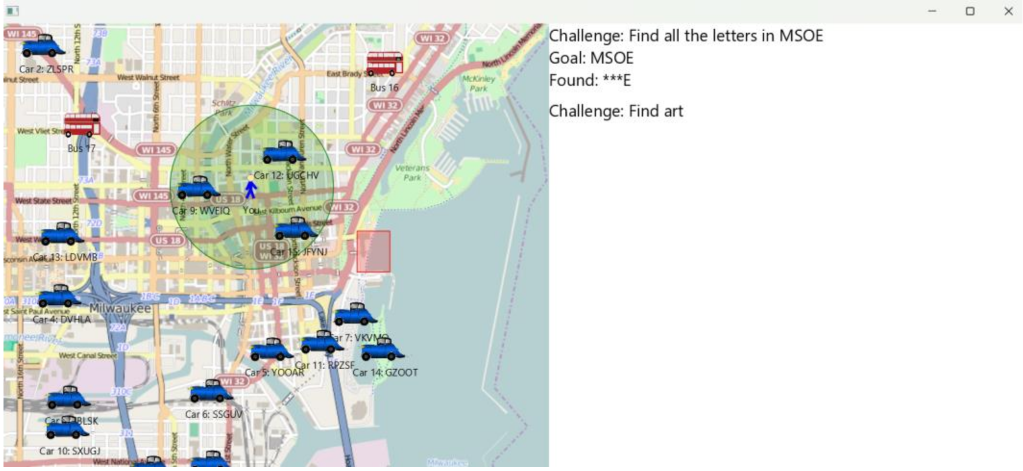
Test Run & Results

What tests did you run to support the requirements of the lab? What was your first pass rate. Demonstrate the outcomes of the tests by providing appropriate screenshots and a short explanation of what the test proves.

For the testing phase, the testing strategy was like the previous lab, although there was a change in approach as the code development went on. Here is the following testing that was done:

- **Print Statement Debugging:** when developing the MSOE challenge, print statements were used to ensure that the plate numbers were being extracted correctly from the cars. Using print statements also helped ensure that string manipulation was being performed correctly to allow the text of the challenge to change from **** to MSOE. The same is applied to Bus challenge, where *** would be changed to BUS once all the letters were found.

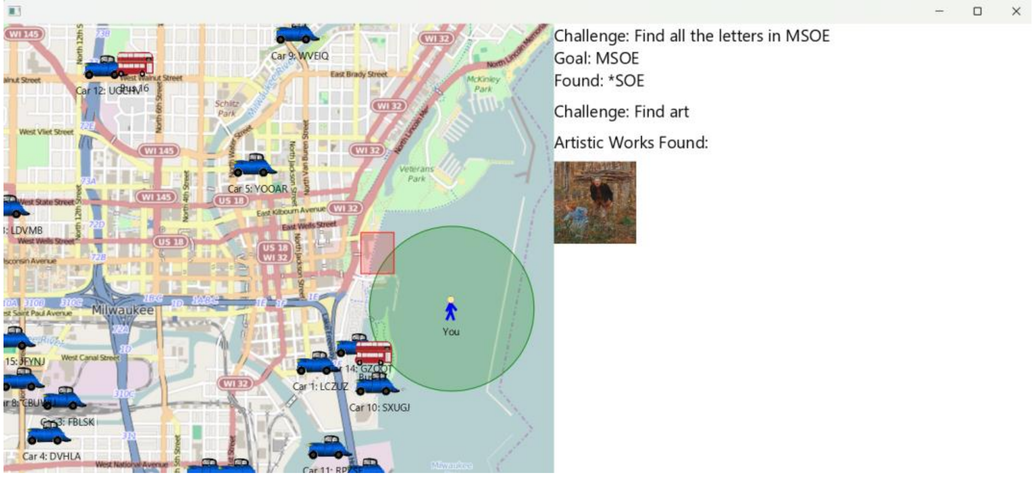
- **Challenge Text on UI:** to ensure that the challenges were displayed on the user interface, the application was run to ensure the MSOE challenge was displayed. Next, the character was dragged towards the bus to ensure that the challenge text for the Bus Challenge was being displayed. Other testing done was ensuring that once a car was tagged which has matching characters, the UI would update and remove some of the * characters.



Challenge: Find all the letters in MSOE
Goal: MSOE
Found: ***E

Challenge: Find art

Figure 1: here, we can see that the text is being updated after person tags Car 9

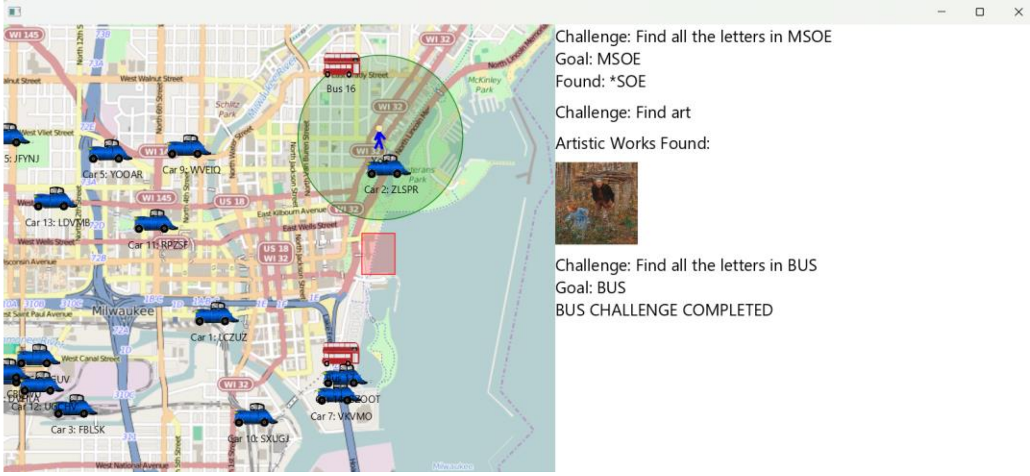


Challenge: Find all the letters in MSOE
Goal: MSOE
Found: *SOE

Challenge: Find art

Artistic Works Found:

Figure 2: here, we can see that the art is being shown, so the text is being updated



Challenge: Find all the letters in MSOE
Goal: MSOE
Found: *SOE

Challenge: Find art

Artistic Works Found:

Challenge: Find all the letters in BUS
Goal: BUS
BUS CHALLENGE COMPLETED

Figure 3: here, once the bus challenge is completed, we can see the UI being updated

Discussion

Conclusions, including what you learned from the tests that you ran.

With regards to my initial observer pattern design, once I had begun implementing extra challenges, I had quickly realized that I had to change my codebase and obtained a better understanding of how the observer pattern had worked. Through the code refactoring process, I had removed code from some classes to other classes, and with the help of debugging statements, I had a better idea of how identify when a person tags a car or a bus, and what kind of action would have to be taken next.

This lab helped me understand how useful the observer pattern was and appreciate it. If implemented correctly, it can make adding future challenges all the easier.

By running the testing, I obtained a better understanding in terms of:

- Where the code was working properly, and where the code wasn't working properly with the use of print statements. The print statements had also helped me to ensure the observer pattern was being implemented, as I could track how information was being passed around from one part of the code to another.