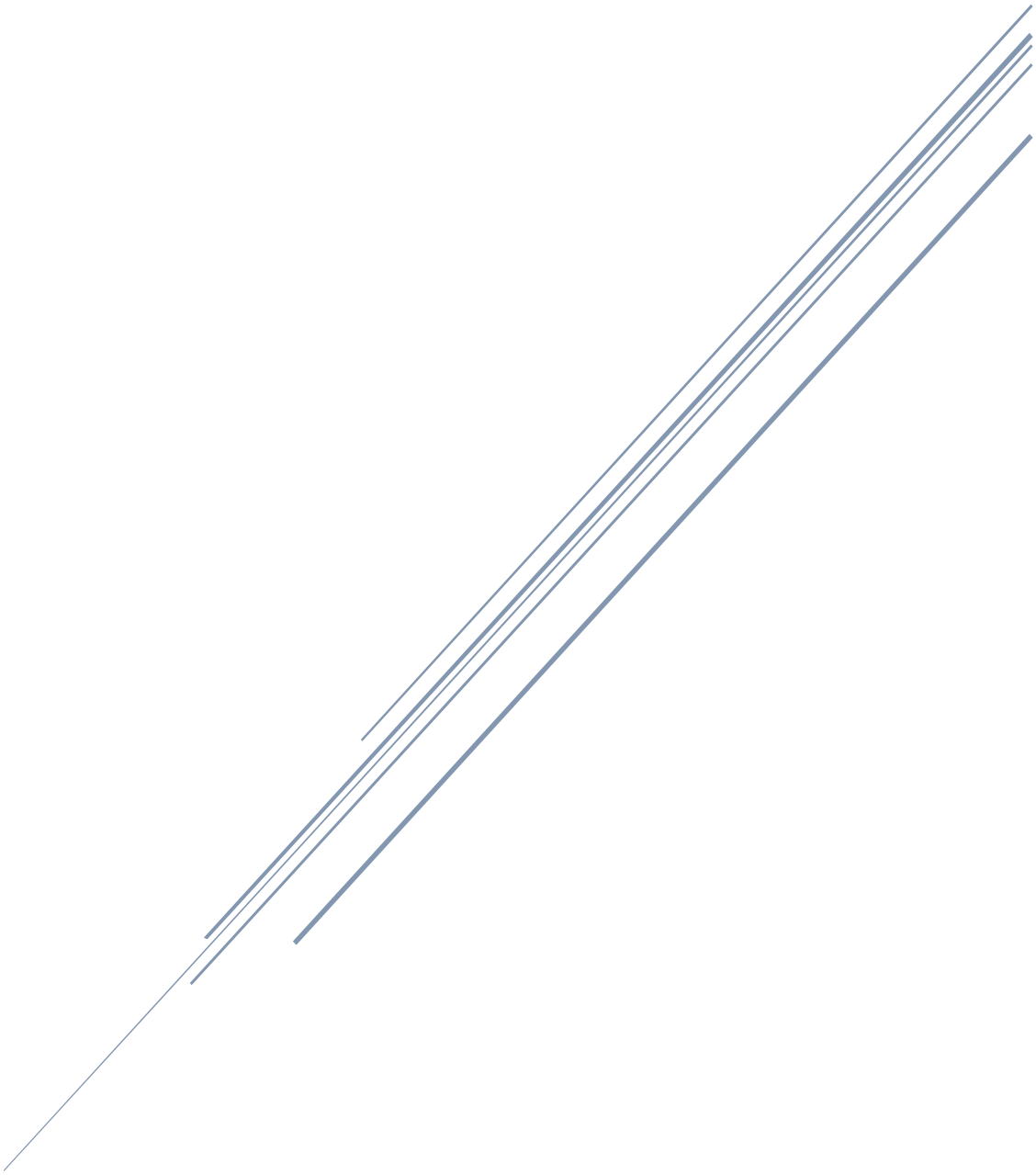
INF_2C SOFTWARE ENGINEERING

Coursework 2



Adam LI s1603732

Caesar ZHANG s1688201

2.1 Introduction

This tour-guiding system is designed with three functions:

1. Browsing tour:

   Users can browse the list of tours saved in the app and the specific detail of a tour if the user prefers.

2. Author tour:

   Tour Authors can create their own tours, including name, waypoints, legs, and annotations.

3. Following tour:

   Users can follow tours from the start point to the endpoint of the tour. The system loads the annotations of each waypoint and leg from database. The system could also calculate the distance and directions from the current position to the next waypoint using real time locations provided by GPS. Throughout the trip, the system displays annotations, distance, and directions on the screen to make the user's trip more convenient.

For further information on requirements, please refer to:

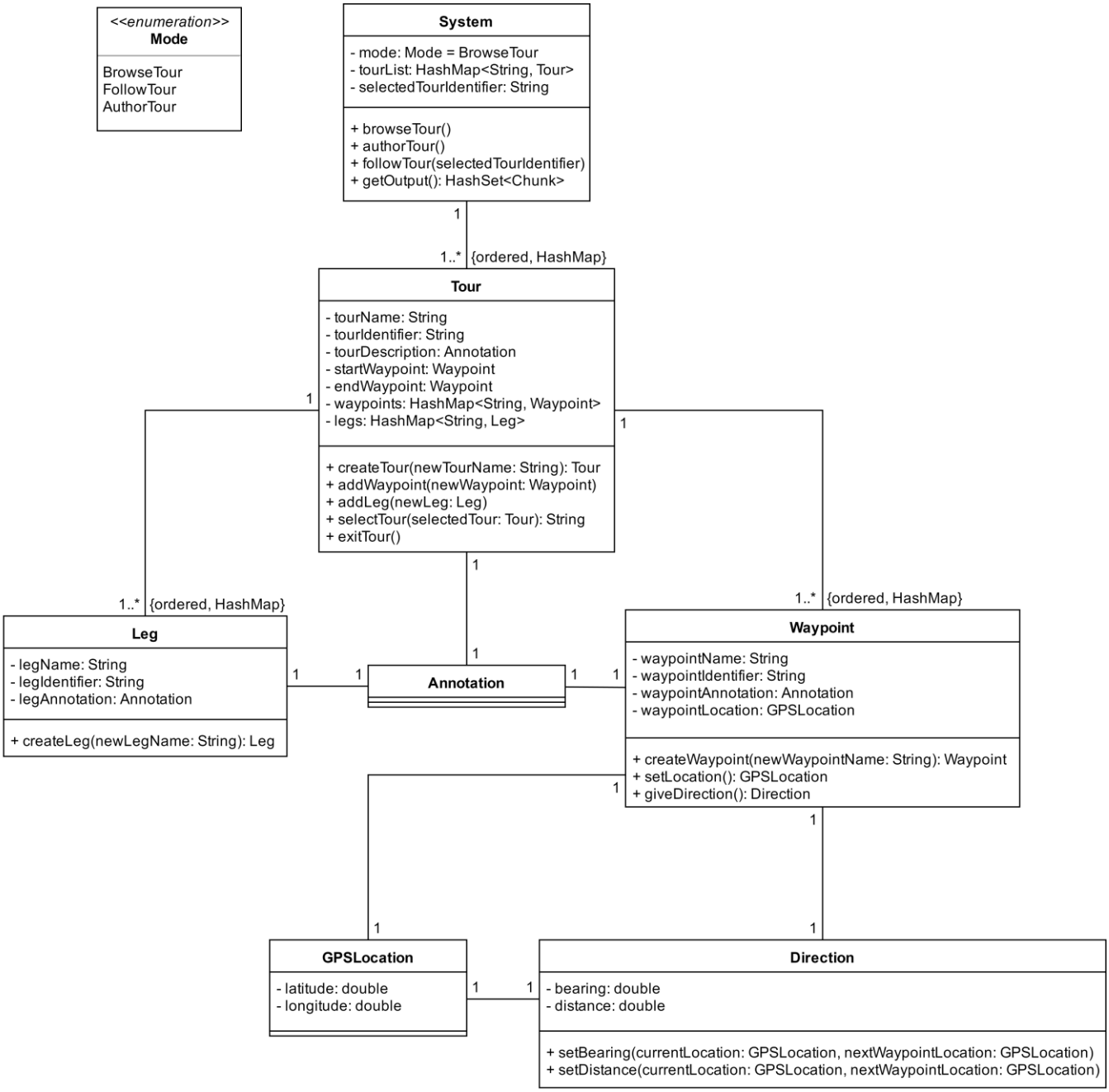
Coursework 1 Instructions

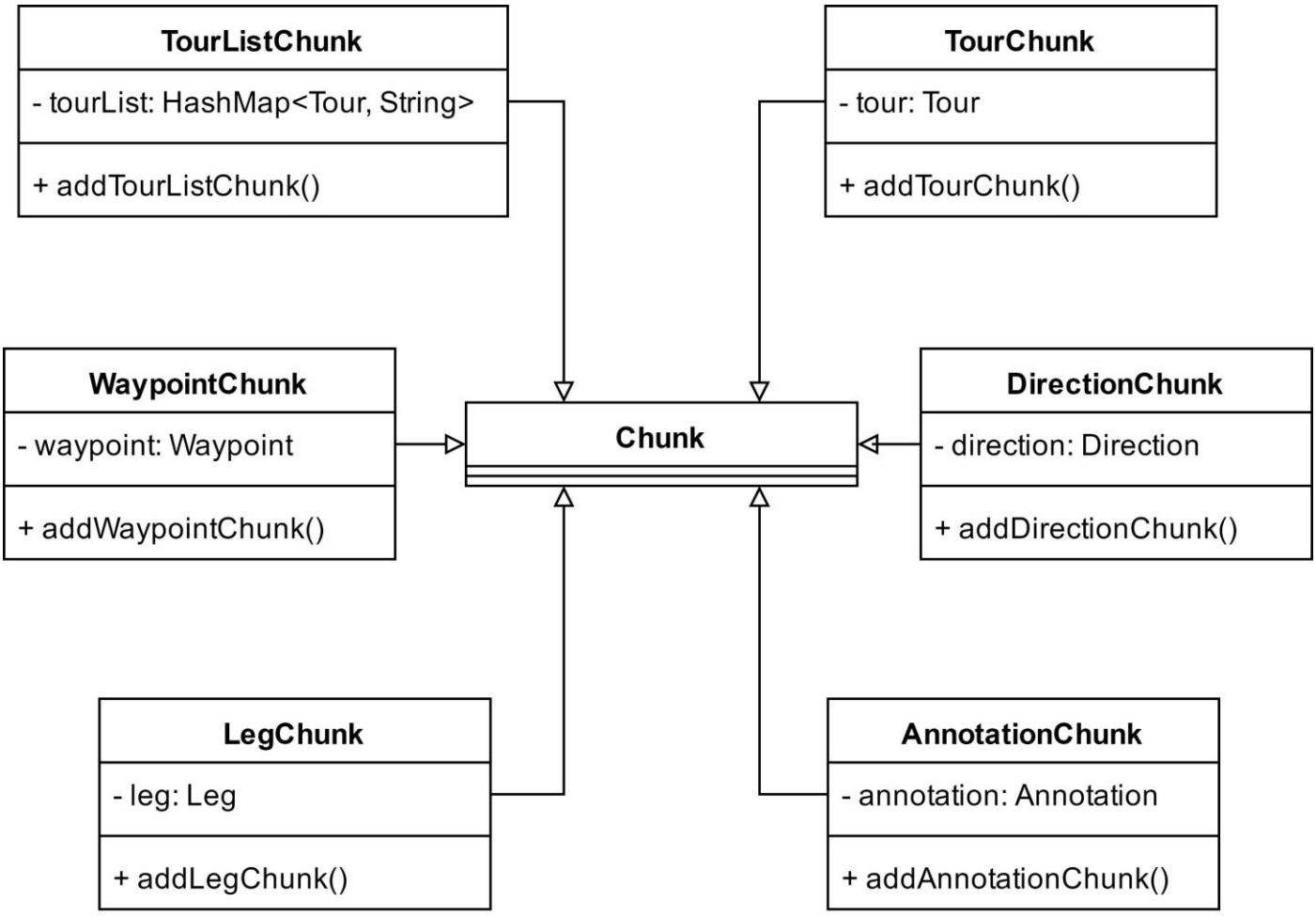
For further information on design, please refer to:

Coursework 2 Instructions

2.2 Static Model (UML Class Diagram and High-level Description):

2.2.1 UML Class Model:





2.2.2 High-level Description:

1. Justification for the design:
   a) Methods to increase cohesion and reduce coupling:

      Most of the functions in each class only take its own class variables as arguments or act as a setter of the class variables which increases cohesion e.g. the variables included in class Tour (“tourName”, “tourIdentifier”, “tourDescription” etc.) and the functions included in Tour (“createTour()”, “addWaypoint()”, “addLeg()”, “selectTour()”, “exitTour()”) are all attributes of tour. Also, each class minimizes the use of variables or functions from other classes to reduces coupling. In our design, there is only one pair of classes that has relatively high coupling: “Direction” and “GPS Location”.

   b) The use of abstraction:

      The class diagram uses abstraction by creating abstract classes “chunk” and “Annotation” which focuses on ‘output of information’ and ‘introduction of tour’ ignoring the exact detail and implement those methods differently in each module. This explains the design straightforward to implementers and avoid being confused in the design level.

   c) The use of encapsulation / information hiding:

      For all classes in the UML class diagram, variables are declared as private variables (e.g. in class leg variable

"legName", "legIdentifier" and "legAnnotation" are all declared as private variables) and can only be accessed via getter and setter so that they are hidden from other classes.

d) The use of decomposition, modularization:

The UML diagram decomposes the overall functioning of the system into different classes e.g. "System", "Tour", "Leg", "Waypoint", "GPS Location", "Direction" and so on, each assumes distinct responsibility. Also, each subordinate function is modularized, which means that the system is more maintainable and more reusable because updating, repairing, and replacing one module does not affect another.

In the UML class design, the sub-functions, namely, "viewTourList()", "browsetour()", "followTour()", which display annotation of waypoints and legs on the screen, are realized in different classes, and each class only contains functions related to its functioning, which decreases coupling and increase cohesion. This also increase maintainability and reusability.

e) The alternatives considered and discarded:

i) The current choice is more concise and more logical than the first draft because it only creates classes when necessary and has deleted the unnecessary classes. For example, the class "TourList" was created to store the function "selectTour()". But now "tourList" is simplified as a class variable in class "system", since "tourList" could be a set of tours for users to browse so that it does not need to be a single class and the function "selectTour" is stored in the class "System". This action also increases cohesion of the class "System", because "TourList" became a variable of the class "Tour". On the contrary, if "TourList" is a separate class, the coupling will increase because the class "TourList" is related to the class "Tour".

ii) Another design I tried and discarded was putting the class "Direction", class "Waypoint" and class "Leg" together into the class "Tour". However, I found that there was too much content for a single class, and decided to separate them into different classes to increase maintainability.

2. Assumptions concerning ambiguities and missing information:

a) All tours are considered free of charge since we ignore steps involving payment.

b) All devices, networking and users are working properly since the UML diagram does not contain methods for exception handling.

c) Users follows the instructions from the system step by step and would not deviate from the route since our class has no related functions for rerouting.

d) The user can only exit the "FollowTour" mode when the last waypoint of the tour is reached.

e) There is no location attribute for a leg.

3. Explanatory notes of class diagram:

a) <<enumeration>> Mode:

This Mode of type <<enumeration>> defines variable mode equals to one of those three Strings "BrowseTour", "FollowTour" and "AuthorTour", which means that the system can be operated in the three modes.

b) Class “System”:

Class “System” has three fields. Field “mode” shows the current state of system, whether user is following a tour, browsing a tour, or authoring a tour, it is set to “BrowseTour” by default. Field “tourList” of type “HashMap<String, Tour>” is the list in which tours are stored. The first parameter String is the unique identifier of a tour, where the second parameter is an instance of the class “Tour”, storing the information of a specific tour which the user would like to follow. Field “selectTourIdentifier” of type String records the identifier of a tour which displays different types of output(chunk) e.g. direction to next waypoint, the name and annotation of leg, the name and annotation of waypoint, a list of tours to select from, and the specific description of an individual tour.

Functions “browerTour()”, “authorTour()”, “followTour(selectedTourIdentifier)” are the three functions the app should realize when entering different modes. Function “getOutput()” is called when an input message about outputting is shown and it returns a HashSet of “Chunk” containing the output content.

c) Class “Tour”:

This class contains variables which describes the detail of a tour, including tour’s “tourIdentifier”, “tourDescription”, “startWaypoint”, “endWaypoint”, “waypoints” and “legs”.

Function “createTour()” takes an argument “newTourName” of type String and return a new tour instance. Function “addWaypoint()” takes an instance of “Waypoint” and add it to the variable “waypoints”. Function “addLeg()” takes an instance of “Leg” and add it to the variable “legs”. Function “selectTour()” takes an instance of “Tour” and returns a “tourIdentifier” of the type String, passing the value to the class “System” telling the class which tour to follow. The function “exitTour()” will end the “FollowTour” mode and return to “BrowseTour” mode.

d) Abstract Class “Annotation”:

This is the abstract class that defines the operations that can be performed on annotation. Also, the class holds data types like formatted text, pictures, video and audio to provide information for waypoints, legs and tours.

e) Class “Leg”:

This class contains three variables “legName”, “legIdentifer” and “legAnnotation” and one function, “createLeg()” which takes an argument “newLegName” of type String and returns an instance of “Leg” as a newly created leg. Instances of leg are ranked in order and stored in a HashMap in class “Tour”.

f) Class “Waypoint”:

This class includes four variables “waypointName”, “waypointIdentifier”, “waypointAnnotation” and “waypointLocation”, which indicates the name, the identifier, the annotation, and the location of a waypoint. Function “createWaypoint()” takes a String as new waypoint name and return an instance of a “Waypoint”. Function “setLocation()” returns an instance of “GPSLocation” and “giveDirection()” returns an instance of “Direction”. Instances of waypoints are ranked in order and stored in a HashMap in class “Tour”.

g) Abstract Class “GPSLocation”:

This is an abstract class containing “latitude” and “longitude” parameters. Collectively, they can locate the position of a phone.

h) Class “Direction”:

This class contains two variables: “bearing” and “distance”. Function “setBearing()” takes “currentLocation”, “nextWaypointLocation” and “GPSLocation” as arguments and calculates the angle from one waypoint to another, “setDistance()” takes “currentLocation”, “nextWaypointLocation” and “GPSLocation” as arguments and calculates the distance to next waypoint.

i) Abstract Class “Chunk”:

This is an abstract class inherited by “DirectionChunk”, “WaypointChunk”, “Annota-tionChunk”, “LegChunk”, “TourlistChunk” and “TourChunk” so that they can be stored concurrently in a HashSet for the function “getOutput()” to return.

j) Class “DirectionChunk”:

This is a subclass of the class “Chunk”. This class has a variable “distance” and a function “addDirectionChunk()” adding the class variable to the HashSet in the class “System”.

k) Class “WaypointChunk”:

This is a subclass of the class “Chunk”. This class has a variable “waypoint” and a function “addWaypointChunk()” adding the class variable to the HashSet in the class “System”.

l) Class “AnnotationChunk”:

This is a subclass of the class “Chunk”. This class has a variable “annotation” and a function “addAnnotationChunk()” adding the class variable to the HashSet in the class “System”.

m) Class “LegChunk”:

This is a subclass of the class “Chunk”. This class has a variable “leg” and a function “addLegChunk()” adding the class variable to the HashSet in the class “System”.

n) Class “TourListChunk”:

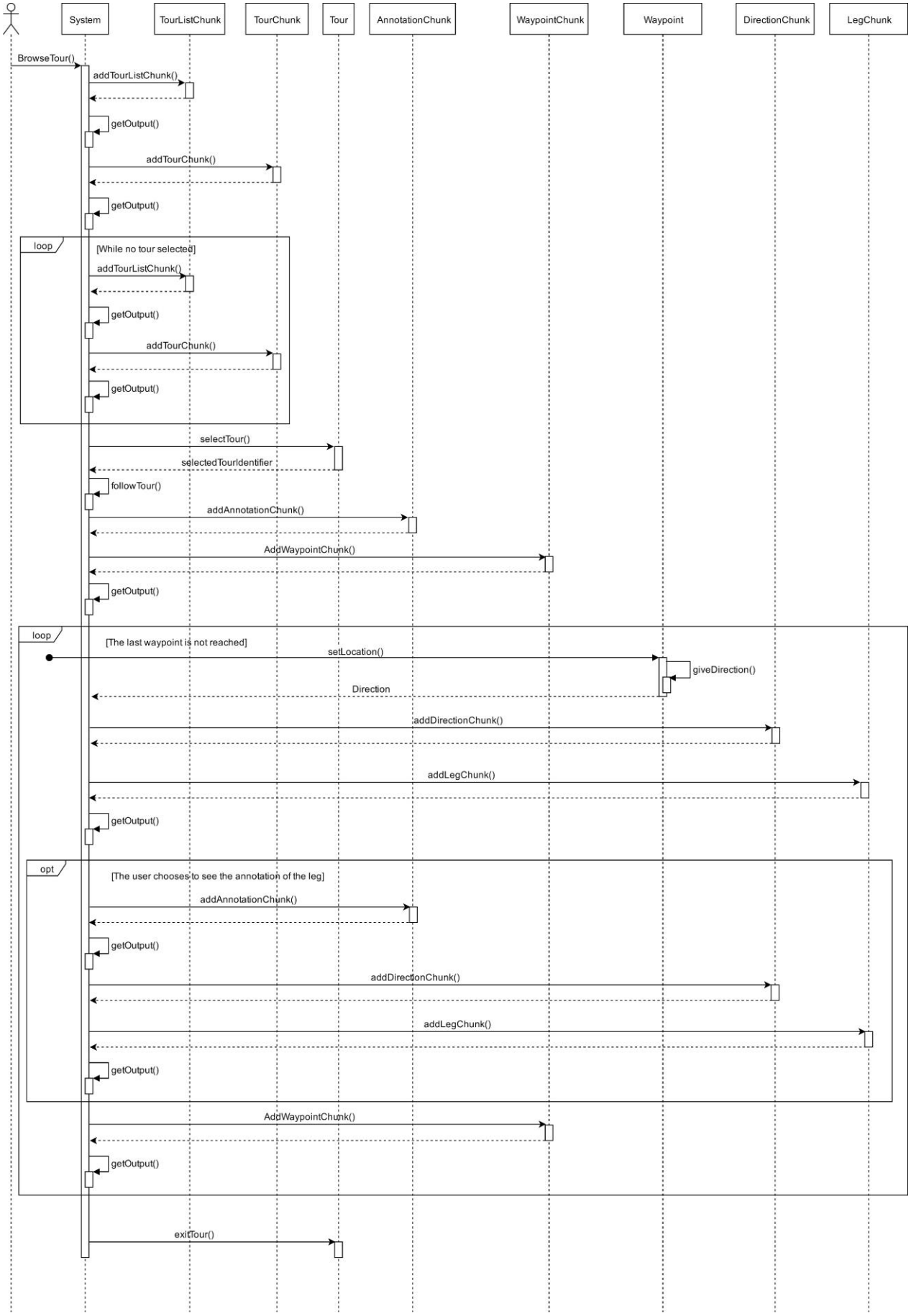
This is a subclass of the class “Chunk”. This class has a variable “tourList” and a function “addTourListChunk()” adding the class variable to the HashSet in the class “System”.

o) Class “TourChunk”:

This is a subclass of the class “Chunk”. This class has a variable “tour” and a function “addTourChunk()” adding the class variable to the HashSet in the class “System”.

2.3 Dynamic Models:

2.3.1 UML Sequence Diagram:



2.3.2 Behavior Descriptions:

aUser -- BrowseTour() --> System

System -- addTourListChunk() --> TourListChunk

System <---------- TourListChunk

System -- getOutput() --> System

System -- addTourChunk() --> TourChunk

System <---------- TourChunk

System -- getOutput() --> System

LOOP [while no tour selected]

System -- addTourListChunk() --> TourListChunk

System <---------- TourListChunk

System -- getOutput() --> System

System -- addTourChunk() --> TourChunk

System <---------- TourChunk

System -- getOutput() --> System

ENDLOOP

System -- selectTour() --> Tour

System <-- selectedTourIdentifier -- Tour

System -- followTour() --> System

System -- addAnnotationChunk() --> AnnotationChunk

System <---------- AnnotationChunk

System -- addWaypointChunk() --> WaypointChunk

System <---------- WaypointChunk

System -- getOutput() --> System

LOOP [The last waypoint is not reached]

---setLocation() --> Waypoint //blank on the left side indicates it is a found message

Waypoint -- giveDirection() --> Waypoint

System <-- Direction -- Waypoint

System -- addDirectionChunk() --> DirectionChunk

System <---------- DirectionChunk

System -- addLegChunk() --> LegChunk

System <---------- LegChunk

System -- getOutput() --> System

OPT [The user chooses to see the annotation of the leg]

System -- addAnnotationChunk() --> AnnotationChunk

System <---------- AnnotationChunk

System -- getOutput() --> System

System -- addDirectionChunk() --> DirectionChunk

System <---------- DirectionChunk

System -- addLegChunk() --> LegChunk

System <---------- LegChunk

System -- getOutput() --> System

ENDOPT

System -- addWaypointChunk() --> WaypointChunk

System <---------- WaypointChunk

System -- getOutput() --> System

ENDLOOP

System -- exitTour() --> Tour