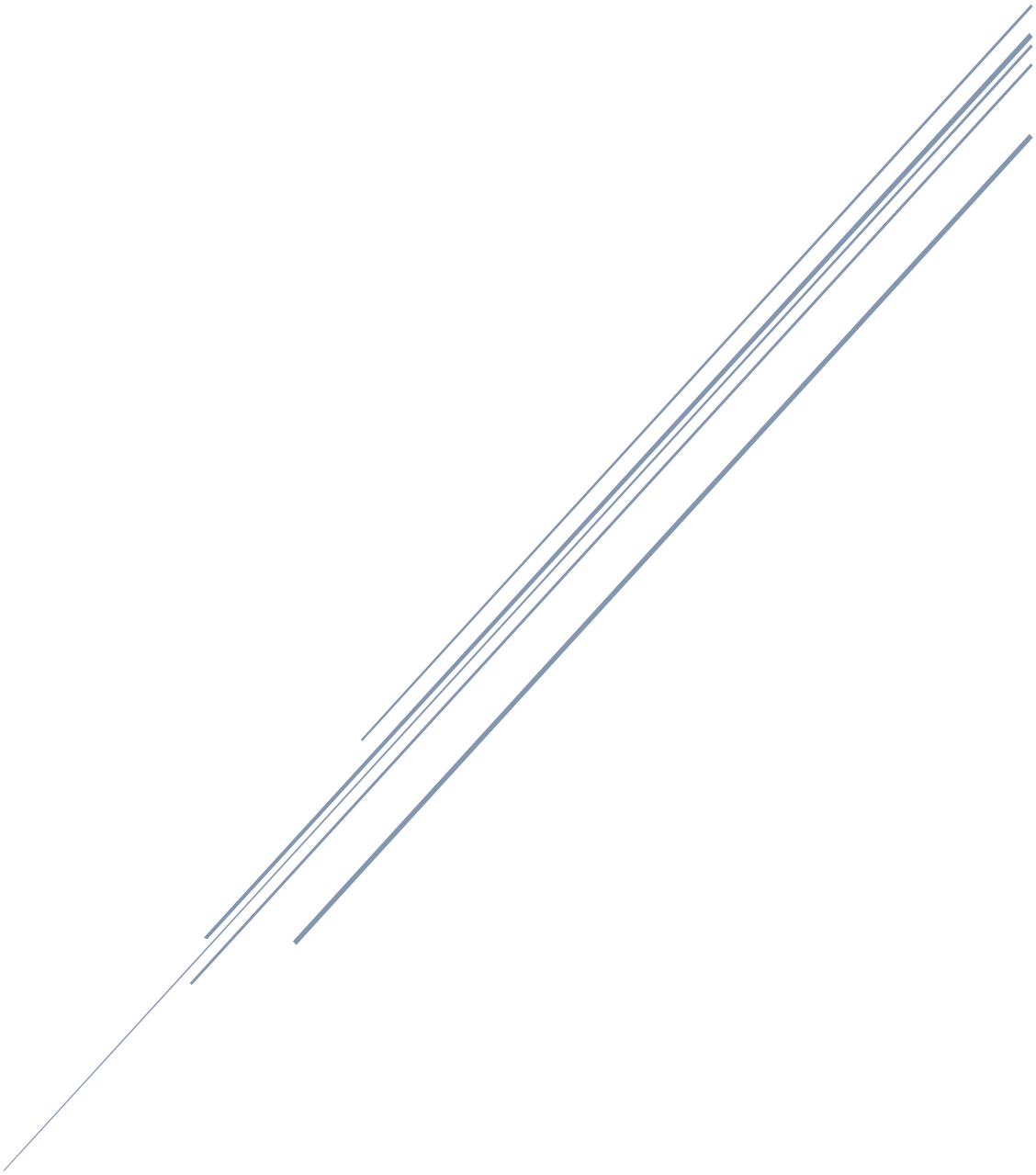
INF_2C SOFTWARE ENGINEERING

Coursework 2



Adam LI s1603732
Caesar ZHANG s1688201

1) UML Class diagram

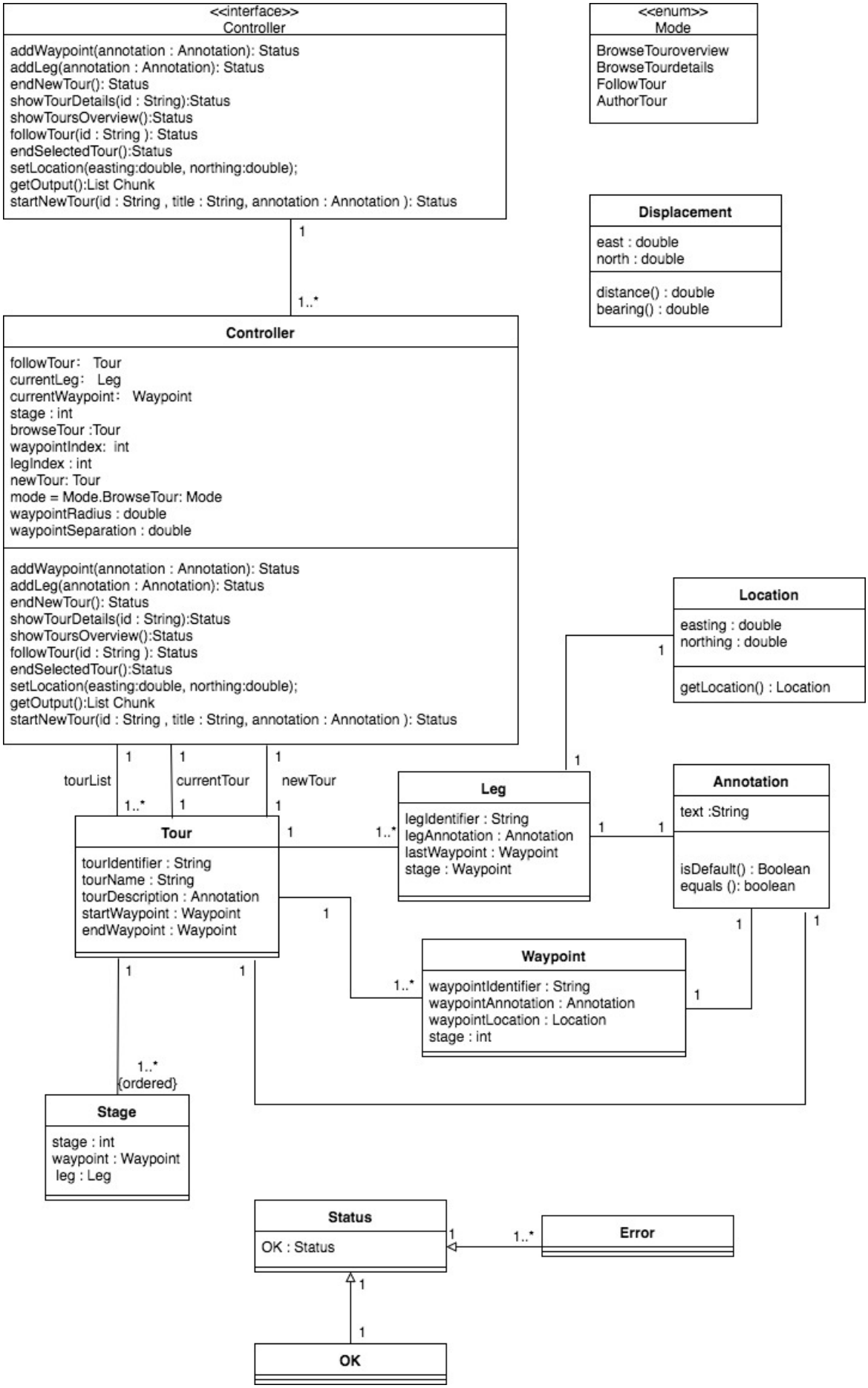

<<interface>>
Controller
addWaypoint(annotation : Annotation): Status
addLeg(annotation : Annotation): Status
endNewTour(): Status
showTourDetails(id : String):Status
showToursOverview():Status
followTour(id : String): Status
endSelectedTour():Status
setLocation(easting:double, northing:double);
getOutput():List Chunk
startNewTour(id : String , title : String, annotation : Annotation): Status
<<enum>>
Mode
BrowseTouroverview
BrowseTourdetails
FollowTour
AuthorTour
Displacement
east : double
north : double
distance() : double
bearing() : double
1
1..*
Controller
followTour: Tour
currentLeg: Leg
currentWaypoint: Waypoint
stage : int
browseTour :Tour
waypointIndex: int
legIndex : int
newTour: Tour
mode = Mode.BrowseTour: Mode
waypointRadius : double
waypointSeparation : double
addWaypoint(annotation : Annotation): Status
addLeg(annotation : Annotation): Status
endNewTour(): Status
showTourDetails(id : String):Status
showToursOverview():Status
followTour(id : String): Status
endSelectedTour():Status
setLocation(easting:double, northing:double);
getOutput():List Chunk
startNewTour(id : String , title : String, annotation : Annotation): Status
Location
easting : double
northing : double
getLocation() : Location
tourList
currentTour
newTour
Leg
legIdentifier : String
legAnnotation : Annotation
lastWaypoint : Waypoint
stage : Waypoint
Annotation
text :String
isDefault() : Boolean
equals (): boolean
Tour
tourIdentifier : String
tourName : String
tourDescription : Annotation
startWaypoint : Waypoint
endWaypoint : Waypoint
Waypoint
waypointIdentifier : String
waypointAnnotation : Annotation
waypointLocation : Location
stage : int
1..*
{ordered}
Stage
stage : int
waypoint : Waypoint
leg : Leg
Status
OK : Status
Error
OK

Figure 1

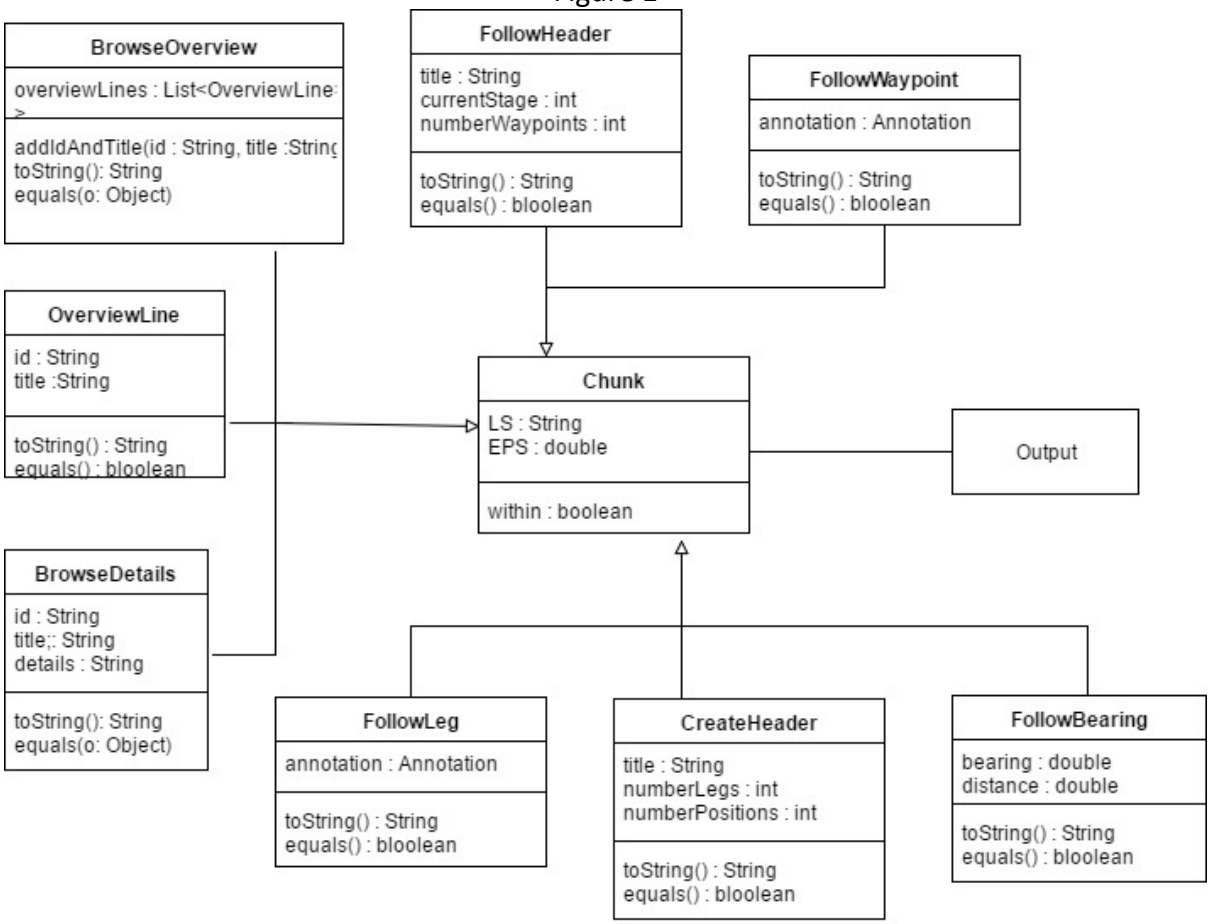


Figure 2

2) High-level design description:
   a) The description and the rationale of each class:
      i) Class “Leg”:
      This class contains four class variables, “legIdentifier”, “legAnnotation”, “lastWaypoint” and “stage”. The rationale behind the design is that an instance of “Leg” is designed to have a unique identifier, an annotation serves as description, a last waypoint indicating the waypoint from which a user travels from, and a stage denoting which stage of the tour this leg belongs to.

      ii) Class “Location”:
      This is the dummy class contains two variables “easting” and “northing” indicating the location’s position in the east and north respectively of some reference position.

      iii) <enum> “Mode”:
      This is a data type allows the “mode” variable in class “ControllerImp” to be set to one of the “BrowseTouroverview”, “BrowseTourdetails”, “FollowTour” and “AuthorTour”.

iv) Class “Stage”:
This class contains three variables “stage”, “waypoint” and “leg”. The rationale behind the design is that an instance of “stage” is expected to have an index indicating position of this stage in a tour, also a stage should contain a “waypoint” and a “leg” (except for the first stage only containing a “leg” and the last stage only containing a “waypoint”).

v) The class “Tour”:
This class has eight class variables, namely, “tourIdentifier”, “tourName”, “tourDescription”, “startWaypoint”, “endWaypoint”, “waypoints”, “legs” and “stages”. The rationale behind the design is that an instance of class “Tour” is expected to have its unique identifier, its own name, starting waypoint, end waypoint, unique description, a list of waypoints, a list of legs, and a list of stages.

vi) The class “Waypoint”:
This class has three variables, namely “waypointIdentifier”, “waypointAnnotation” and “waypointLocation”. The rationale behind the design is that an instance of class “waypoint” is expected to have its unique identifier, its own annotation, its unique location and the stage it belongs to.

vii) Class “ControllerImp”:
A variable “mode” of type mode is added to indicate the state the system is currently in, this is necessary because some actions can only be performed under some circumstance e.g. a new tour can only be created if the system is in browse tour mode. A variable “tourList” is added to store tours maintained by system. Variables “waypointRadius” and “waypointSeparation” are created to store the values passed in by the constructor.

b) The change of design from coursework2:
The class “Direction” in coursework2 is cancelled because the function of this class is realized in “getOutput()” function in “ControllerImp”. Also, there are more fields added to the “ControlerImp” class.

3) Implementation decisions:

a) Class “Leg”:

The constructor contains only “LegIdentifier” and “annotation” because the function “addLeg” where an instance of “Leg” is initialized only provides “legIdentifier” and “annotation”.

b) Class “Location”:

The rationale behind the implementation is that the function “setLocation” in “ControllerImp” where an instance of Location is initialized only has two arguments. This class also contains a getter for current location which is called by class “ControllerImp” to get the current location of the user.

c) The class “Tour”:

The class “Tour” contains three Array Lists to store waypoints, legs and stages for a specific tour because each instance of class “Tour” contains more than one waypoints, legs, and tours.

The constructor only contains three arguments (tour’s identifier, tour’s name, tour’s description) because the function “startNewTour” where an instance of a new tour initialized only takes those three parameters.

This class also contains getters and setters for every class variable.

d) The class “Waypoint”:

The constructor of this class only has two parameters because the function “addWaypoint” where an instance of “waypoint” is initialized only provides “waypointIdentifier” and “waypointAnnotation”. This class also provides getters and setters for each of its class variable.

e) Class “ControllerImp”:

The type “HashMap<String, Tour>” is chosen for variable “tourList” because for each instance of class “Tour”, there is a unique identifier serves as the key of type “HashMap”. “waypointRadius” is used to check whether the user is within a waypoint and “waypointSeparation” is used to determine whether the location to create a waypoint is far away enough from the last waypoint.

The functions for “createTour” mode uses four class variables:
Variable “waypointIndex” indicates the index of a waypoint to be added to a tour. This is concatenated with the tour’s identifier to create the waypoint’s identifier. Variable “legIndex” indicates the index of a leg to be added to a tour. This is concatenated with the tour’s identifier to create the leg’s identifier. Variable “newTour” indicates the tour that is currently being created. Variable “lastSetWaypoint” indicates the last waypoint set by tour author in the process of authoring tour. Attributes of a new tour e.g. waypoints, identifier, legs and annotation are added into the variable by functions “startNewTour”, “addWaypoint”, “addLeg”.

The functions for “BrowseTourOverVIew” mode and “BrowseTourDetail” mode use only one class variable:
The variable “browseTour” of type “Tour” indicates the tour that is currently being browsed.

The functions for “FollowTour” mode only use six class variable:
Variable “followTour” of type “Tour” indicates the tour that is currently being followed. Variable “currentLeg” of type “Leg” indicates the leg the tourist is currently on if the user is currently on a leg. Variable “currentWaypoint” of type “Waypoint” indicates the waypoint the tourist is currently on if the tourist is currently on a waypoint. Variable “nextWaypoint” of type “Waypoint” indicates the waypoint the tourist is travelling to. Variable “stage” of type “int” indicates the waypoint the tourist is currently on. Variable “stages” of type “List<Stage>” indicates the waypoint the tourist is currently on.

The functions for all modes only use three class variable: “easting” and “northing” are the position of a user respectively of some objects. The variable “displacement” is used by function “getOutput()” to calculate the direction to the next waypoint.

The function “tooClose” takes user’s current location and the location of last way point to calculate whether current location too close from the last waypoint to create another waypoint. If the distance is too close, the function will return true.

The “getOutput()” function creates different types of output chunks according to the mode the system is currently in and adds them to a list which is later called by other functions.