Modelling Trust Dynamics in the Internet of Things: Simulator

This document presents the simulator developed in Java 8 for the framework described in *Modelling Trust Dynamics in the Internet of Things*, by Fernandez-Gago et al. This simulator is developed and documented by Anna Guinet, under the supervision of Davide Ferraris and Carmen Fernandez-Gago.

Contents

06/06/2017

I. Framework

The aim of the framework, presented in *Modelling Trust Dynamics in the Internet of Things* by Fernandez-Gago et al., is to select the best professional profile for accomplishing a task, in a company where employees carried smart devices.

In our use case, a gas company employs *operators*, and has to choose the best professional profile for accomplishing a *task*. The selection is based on the reputation and the years of experience of operators, by means of trust metrics. Each operator can carry four connected *things:* a *car*, a *smartphone*, a *PDA* and/or *smartglasses*. They always have at least one thing. The things carried define a *context* for each operator. Based on these contexts and the reputation score, the system can select the best operator.

The simulation is set for a day, from 7 a.m. to 10 p.m., where operators have an 8-hour working day and can arrive between 7 a.m. and 2 p.m.

II. Simulator

The simulator computes random values for operators and their contexts, and tasks.

1. Operator

Each operator is defined by:

- An ID.
- His/her first name.
- His/her last name.
- His/her professional profile.
- The arrival time at work: between 7 a.m. and 2 p.m. The work day lasts 8 hours.
- The number of tasks completed: between 0 and 15 tasks per day.
- A global reputation value: between 0.000 and 1.000.
- A location: a latitude and a longitude in decimal degrees.
- The years of experience: between 0 and 40 years.
- The availability: a Boolean.
- His/her contexts.

The simulator computes random values for the ID, the arrival time, the number of tasks completed, the global reputation value, the location, the years of experience, the availability and the contexts (see below).

2. Operator Context

Each operator context is defined by:

- An ID.
- A time which represents the context change during the working day.
- The set of things carried by an operator.

The simulator computes random values for the ID, the date and the set of things.

a. Set of things

Each operator holds a car, a smartphone, a PDA and smartglasses. Those things are specific to one operator and are identified by an ID.

First, the simulator computes random IDs to define the things owned by one operator. Then, the simulator generates a unique set of things from the owned things, specific to one context.

For each context, the operator has between one to four things, so there are fifteen unique possibilities. The simulator computes between 1 and 15 contexts per operator.

Each thing is defined by:

- An ID.
- A security certification: a Boolean.
- The date for the last supervision.
- The number of past vulnerabilities: between 0 and 30.
- A petrol level for the car.

The simulator computes the ID, the presence of a security certification, the date of the last supervision and the number of past vulnerabilities.

3. Task

Each task is defined by:

- An ID.
- An estimated duration: between 0 and 12 hours.
- An estimated complexity: between 0.00 and 1.00.
- A criticality: between 1 and 5 (five-star score).
- The preferred profile.
- A location: a latitude and a longitude in decimal degrees.
- A time: between 7 a.m. and 10 p.m.
- An expiration time: maximum 6 hours after the creation of the task, without exceeding the current working day. This time defines the end of the availability of the task.
- The things required for accomplishing a task: smartglasses, smartwatch and/or PDA (use of car depends on the distance).

The simulator computes the ID, the estimated duration, the estimated complexity, a criticality, the location, the departure time, the expiration time and the required things.

4. Location

The location is set in the Madrid area whose coordinates are extracted from the Google Maps Geocoding API[1].

Locations use the library Jcoord v1.1 developed by Jonathan Scott[2], and are expressed in decimal degrees.

III. Algorithm: Selection of the best Operator for a task

1. Pre-Selection of Candidates

Before choosing the best operator for a given task, a method pre-selects them. First, it checks whether the operators are in their working time and available, or not. Then, the method compares the remaining work hours and the estimated duration of the task: if the operator does not have enough time, he/she is not selected. Next, the method retrieves the current context of the operator and verifies that he/she holds the minimum required things for accomplishing the task. Finally, the distance between the operator and the task is calculated, and the method checks whether the operator needs his/her car and whether he/she has it. In our use case, the operator can walk to the task location if the distance is inferior to 1km. See the appendix D.a. for more details.

2. Selection of the best Operator

From the previous selected operators, a method chooses the best one according to their reputation and their years of experience.

Let R be the reputation and E the years of experience of an operator, the calculated score is:

$$score = \frac{R * E}{\max(R) * \max(E)}$$

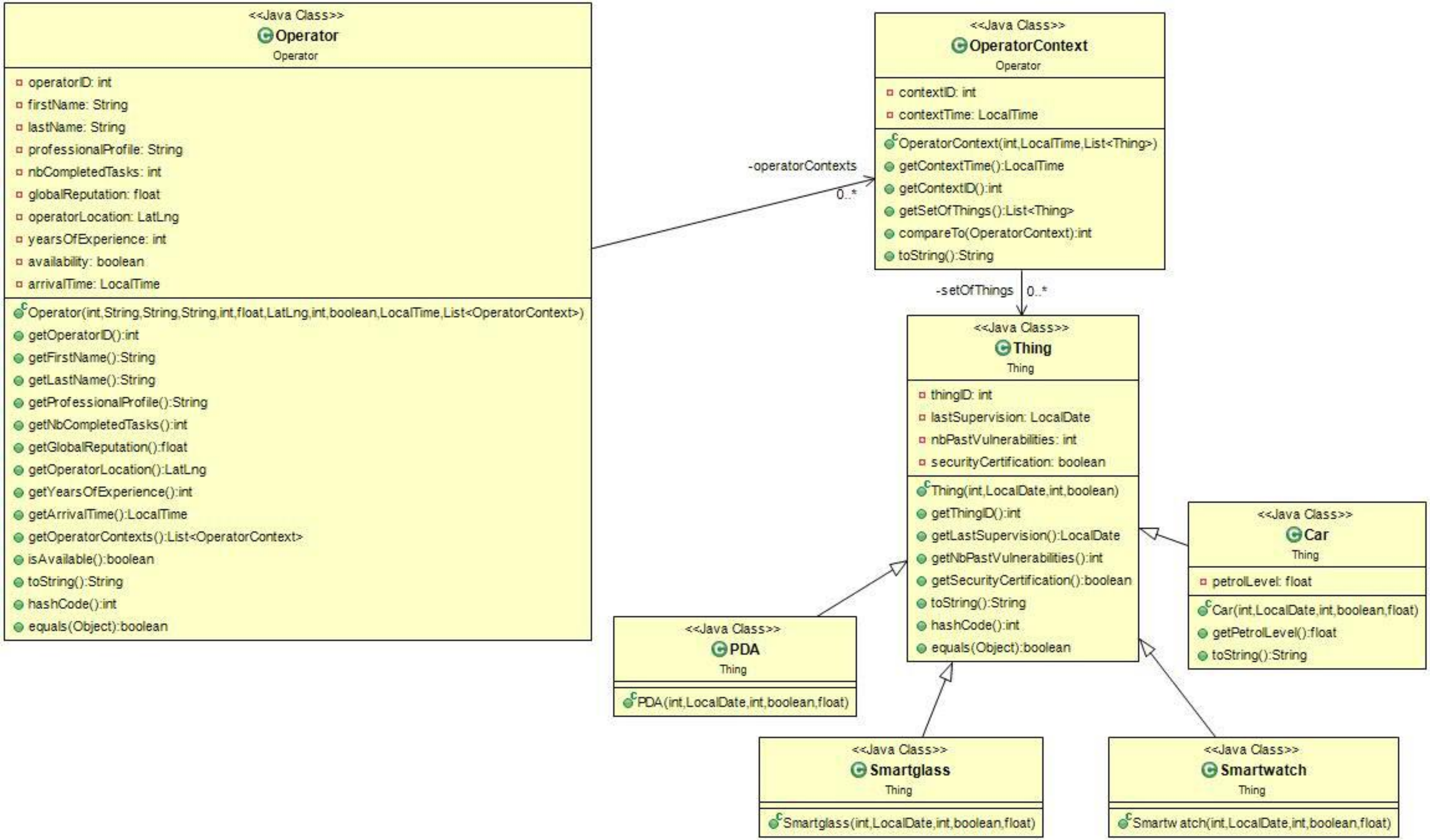
The operator with the highest score is chosen for the task.

[1] https://developers.google.com/maps/documentation/geocoding/intro

[2] http://www.jstott.com/jcoord/

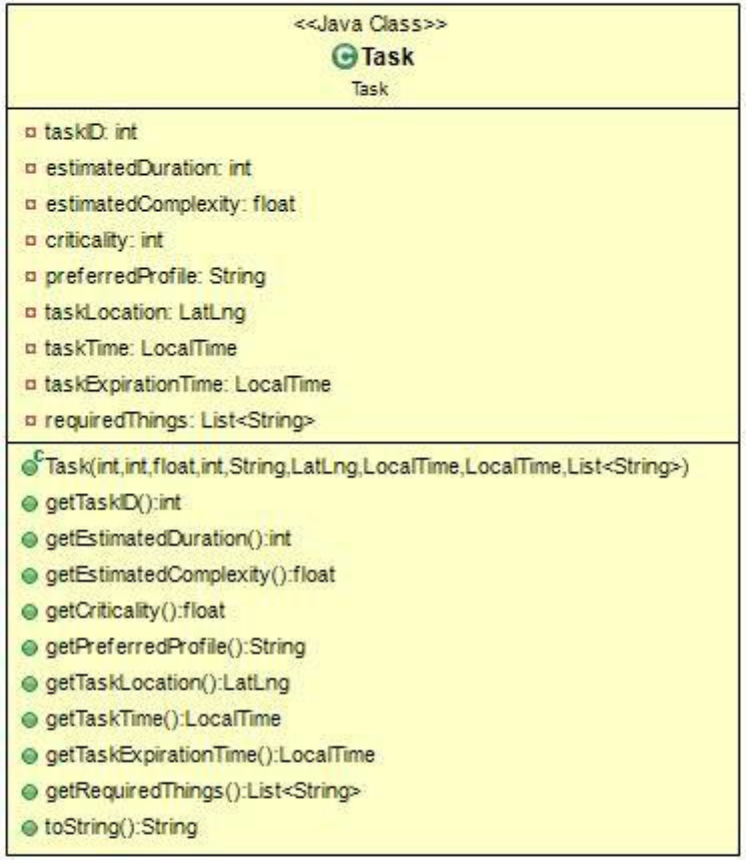
IV. Appendices: UML Diagrams

A. Operator, Operator context and Things classes



<<Java Class>>
Operator
Operator
operatorID: int
firstName: String
lastName: String
professionalProfile: String
nbCompletedTasks: int
globalReputation: float
operatorLocation: LatLng
yearsOfExperience: int
availability: boolean
arrivalTime: LocalTime
Operator(int,String,String,String,int,float,LatLng,int,boolean,LocalTime,List<OperatorContext>)
getOperatorID():int
getFirstName():String
getLastName():String
getProfessionalProfile():String
getNbCompletedTasks():int
getGlobalReputation():float
getOperatorLocation():LatLng
getYearsOfExperience():int
getArrivalTime():LocalTime
getOperatorContexts():List<OperatorContext>
isAvailable():boolean
toString():String
hashCode():int
equals(Object):boolean
-operatorContexts
0..*
<<Java Class>>
OperatorContext
Operator
contextID: int
contextTime: LocalTime
OperatorContext(int,LocalTime,List<Thing>)
getContextTime():LocalTime
getContextID():int
getSetOfThings():List<Thing>
compareTo(OperatorContext):int
toString():String
-setOfThings
0..*
<<Java Class>>
Thing
Thing
thingID: int
lastSupervision: LocalDate
nbPastVulnerabilities: int
securityCertification: boolean
Thing(int,LocalDate,int,boolean)
getThingID():int
getLastSupervision():LocalDate
getNbPastVulnerabilities():int
getSecurityCertification():boolean
toString():String
hashCode():int
equals(Object):boolean
<<Java Class>>
Car
Thing
petrolLevel: float
Car(int,LocalDate,int,boolean,float)
getPetrolLevel():float
toString():String
<<Java Class>>
PDA
Thing
PDA(int,LocalDate,int,boolean,float)
<<Java Class>>
Smartglass
Thing
Smartglass(int,LocalDate,int,boolean,float)
<<Java Class>>
Smartwatch
Thing
Smartw atch(int,LocalDate,int,boolean,float)

B. Task class

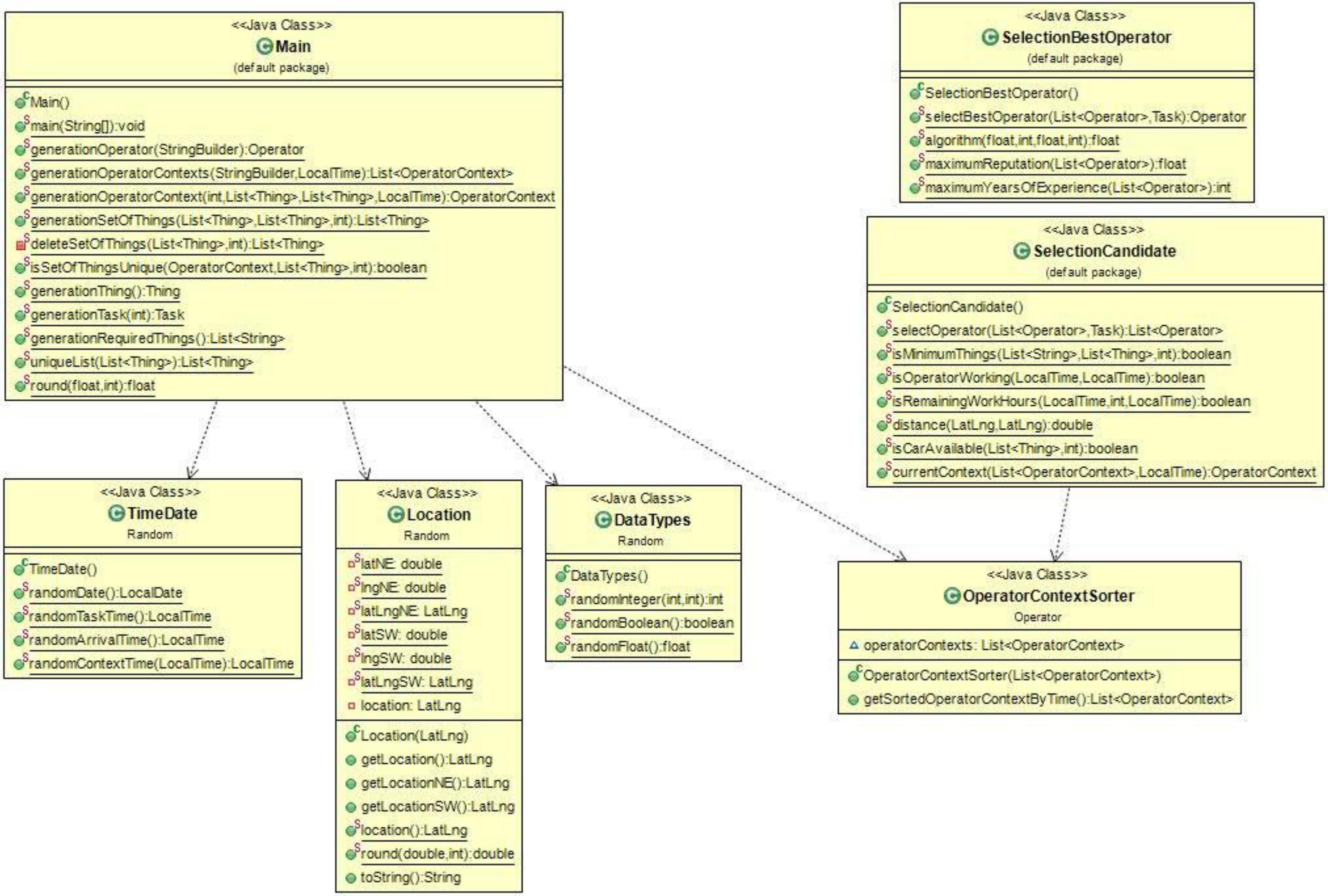


<<Java Class>>
Task
Task

- taskID: int
- estimatedDuration: int
- estimatedComplexity: float
- criticality: int
- preferredProfile: String
- taskLocation: LatLng
- taskTime: LocalTime
- taskExpirationTime: LocalTime
- requiredThings: List<String>

- Task(int,int,float,int,String,LatLng,LocalTime,LocalTime,List<String>)
- getTaskID():int
- getEstimatedDuration():int
- getEstimatedComplexity():float
- getCriticality():float
- getPreferredProfile():String
- getTaskLocation():LatLng
- getTaskTime():LocalTime
- getTaskExpirationTime():LocalTime
- getRequiredThings():List<String>
- toString():String

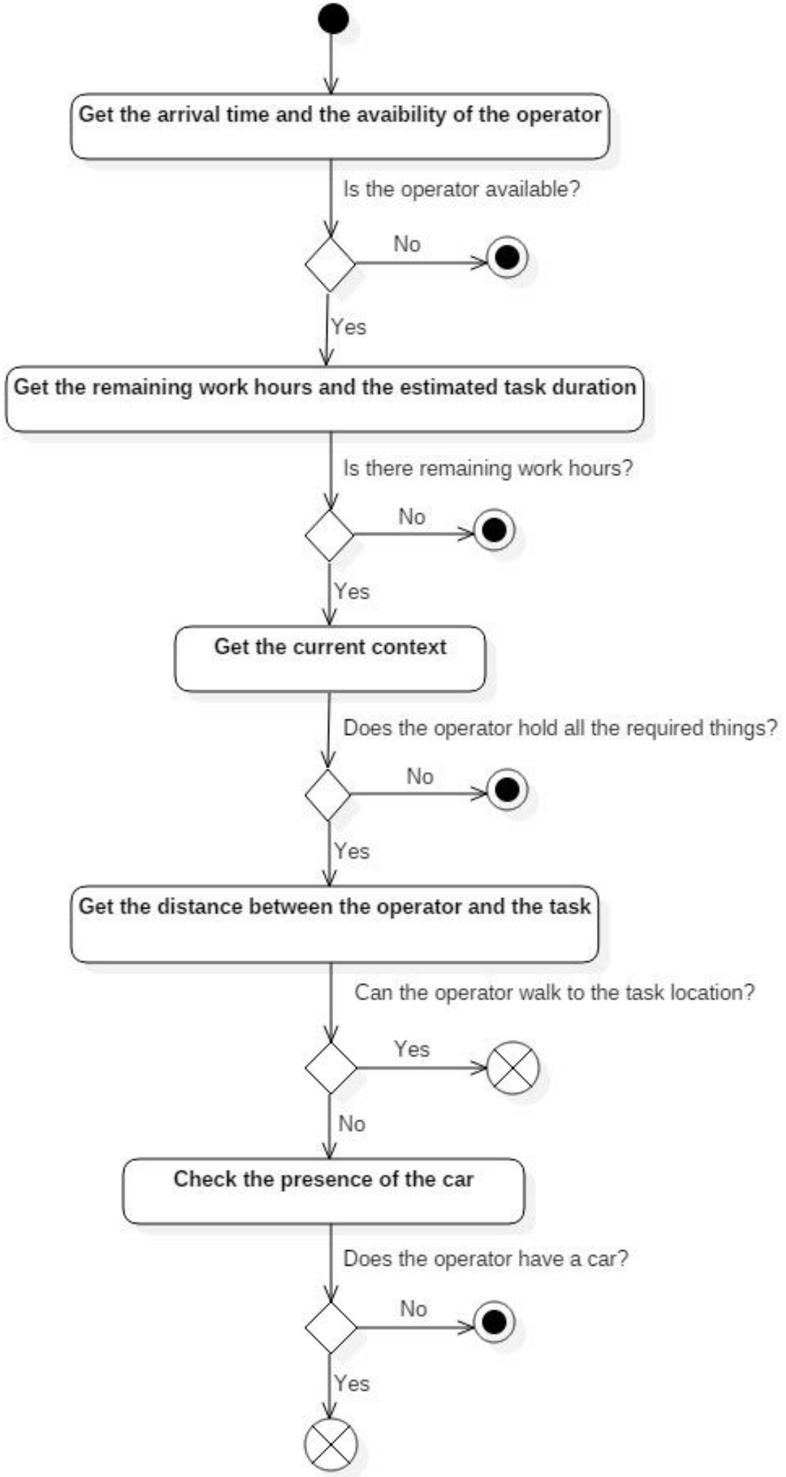
C. Main class



<<Java Class>>
Main
(default package)
Main()
main(String[]):void
generationOperator(StringBuilder):Operator
generationOperatorContexts(StringBuilder,LocalTime):List<OperatorContext>
generationOperatorContext(int,List<Thing>,List<Thing>,LocalTime):OperatorContext
generationSetOfThings(List<Thing>,List<Thing>,int):List<Thing>
deleteSetOfThings(List<Thing>,int):List<Thing>
isSetOfThingsUnique(OperatorContext,List<Thing>,int):boolean
generationThing():Thing
generationTask(int):Task
generationRequiredThings():List<String>
uniqueList(List<Thing>):List<Thing>
round(float,int):float
<<Java Class>>
SelectionBestOperator
(default package)
SelectionBestOperator()
selectBestOperator(List<Operator>,Task):Operator
algorithm(float,int,float,int):float
maximumReputation(List<Operator>):float
maximumYearsOfExperience(List<Operator>):int
<<Java Class>>
SelectionCandidate
(default package)
SelectionCandidate()
selectOperator(List<Operator>,Task):List<Operator>
isMinimumThings(List<String>,List<Thing>,int):boolean
isOperatorWorking(LocalTime,LocalTime):boolean
isRemainingWorkHours(LocalTime,int,LocalTime):boolean
distance(LatLng,LatLng):double
isCarAvailable(List<Thing>,int):boolean
currentContext(List<OperatorContext>,LocalTime):OperatorContext
<<Java Class>>
TimeDate
Random
TimeDate()
randomDate():LocalDate
randomTaskTime():LocalTime
randomArrivalTime():LocalTime
randomContextTime(LocalTime):LocalTime
<<Java Class>>
Location
Random
latNE: double
lngNE: double
latLngNE: LatLng
latSW: double
lngSW: double
latLngSW: LatLng
location: LatLng
Location(LatLng)
getLocation():LatLng
getLocationNE():LatLng
getLocationSW():LatLng
location():LatLng
round(double,int):double
toString():String
<<Java Class>>
DataTypes
Random
DataTypes()
randomInteger(int,int):int
randomBoolean():boolean
randomFloat():float
<<Java Class>>
OperatorContextSorter
Operator
operatorContexts: List<OperatorContext>
OperatorContextSorter(List<OperatorContext>)
getSortedOperatorContextByTime():List<OperatorContext>

D. Selection algorithm

a. Pre-selection method


Get the arrival time and the avaibility of the operator
Is the operator available?
No
Yes
Get the remaining work hours and the estimated task duration
Is there remaining work hours?
No
Yes
Get the current context
Does the operator hold all the required things?
No
Yes
Get the distance between the operator and the task
Can the operator walk to the task location?
Yes
No
Check the presence of the car
Does the operator have a car?
No
Yes