Computer Assignment 1: The Arrival Process in Simkit

This will guide you through the process of creating and running a simple Event Graph model using Simkit. It is self-contained; all the information you should need is in this document. [[1]](#footnote-1)Furthermore, it will walk you through several intermediate steps in creating the final model. You will only be graded on the *final* version however, not the intermediate versions.

Objectives

* Implement simple Event Graph in Simkit
* Learn about SimEntityBase class
* Learn methods of Schedule to initialize and start simulation
* Generate random variates using RandomVariate and RandomVariateFactory
* waitDelay() method of SimEntityBase
* Schedule methods:
* Schedule.setVerbose()
* Schedule.stopAtTime()
* Schedule.reset()
* Schedule.startSimulation()
* Schedule.getSimTime()

Before You Start

Make sure you have downloaded and installed the Simkit library. Also, define a User Library in your IDE. This is so your classes will be able to find the necessary ones in Simkit.

The Arrival Process

The discrete-event simulation model of an arrival process specifies a single state, the cumulative number of arrivals, and one event, an Arrival, with times between arrivals either deterministic or random. The ArrivalProcess Event Graph model is defined as follows:

Parameter

*  is the sequence of (possibly random) times between the occurrences of the Event.

State

* N is the number of times the Event has occurred. Its initial value is 0.

Event Graph:



Figure 1. ArrivalProcess Event Graph

Implementation

The ArrivalProcess model will be implemented in stages.

1. Create the ArrivalProcess class as a subclass of SimEntityBase and write the import.

Define the RandomVariate parameter and add setter/getter methods and the constructor

Create the RunArrivalProcess main class and verify the parameter is correctly passed

Define the state variable and its getter method

Write the reset() and doRun() methods

Test by executing the model in RunArrivalProcess

Write the doArrival() method and test

Add a SimplePropertyDumper to display state changes.

Run the final test.

Comment your code!

### Create the ArrivalProcess class

Define a subclass of SimEntityBase called (imaginatively) ArrivalProcess; be sure to put it in a package called mv3302. In Java, package names are lower-case by convention. In Netbeans, you can first type in “extends SimEntityBase” after which a small light bulb will appear to the left of that line. Click on it and select “add import for SimEntityBae.” The SimEntityBase class is in the simkit package, so your class should have (at this point) the following code:[[2]](#footnote-2)

package mv3302;

import simkit.SimEntityBase;

/\*\*

\*

\* @author <your name>

\*/

public class ArrivalProcess extends SimEntityBase {

}

**Important:** Your classes *must* be in a package called mv3302 (lower-case “mv”).

### Define RandomVariate parameter

Define the parameter interarrivalTimeGenerator as follows:

private RandomVariate interarrivalTimeGenerator;

You will have to add an import statement:

import simkit.random.RandomVariate;

Finally, define setters and getters for this variable. Your IDE should be able to automatically generate these for you.[[3]](#footnote-3) Add one constructor with a RandomVariate argument and a second with an empty signature. Your code should now look like this:[[4]](#footnote-4)

public ArrivalProcess(RandomVariate interarrivalTimeGenerator) {

this.setInterarrivalTimeGenerator(interarrivalTimeGenerator);

}

public ArrivalProcess() { }

public RandomVariate getInterarrivalTimeGenerator() {

return interarrivalTime;

}

public void setInterarrivalTimeGenerator(

RandomVariate interarrivalTimeGenerator) {

this.interarrivalTimeGenerator = interarrivalTimeGenerator;

}

### Start the Main Class

At this point you should write a main class (i.e. a class with nothing but a main() method) to test your ArrivalProcess class; call your main class RunArrivalProcess. Put it in a package called mv3302.run. Obtain an instance of RandomVariate as shown below and use it to pass to the constructor for ArrivalProcess. You will have to add appropriate imports[[5]](#footnote-5) for this to compile.

public static void main(String[] args) {

RandomVariate interarrvalTimeGenerator =   
 RandomVariateFactory.getInstance("Exponential", 3.2);

ArrivalProcess arrivalProcess =

new ArrivalProcess(interarrvalTimeGenerator);

System.out.println(arrivalProcess);

}

The output should be:

mv3302.ArrivalProcess.1

interarrvalTimeGenerator = Exponential (3.2)

Do not proceed until you get this output.

### Define the State Variable

In Simkit models, each state variable is implemented by an instance variable with protected access. The arrival process Event Graph model has a single state variable. In order to cut down on clutter, Event Graphs tend to use state variables with short names—N for the cumulative number of arrivals in Figure 1, for example. Computer programs, on the other hand, should have variable (and method) names that are as self-describing as possible. Therefore, call your state variable numberArrivals. Generate a getter method for num­berArrivals, but *not* a setter method. In general, state variables should always have public getter methods, but never public setter methods, since this is how Simkit determines whether an instance variable is in fact a state variable.

### Define reset() and doRun()

The Run event is special - it is implemented in Simkit by two methods: reset() and doRun(). These must be defined with no arguments to work properly. The reset() method is reponsible for setting all state variables to their initial values, so it looks like this:

public void reset() {

super.reset();

numberArrivals = 0;

}

In general, do *not* initialize state variables in the constructor, only in the reset() method.

The doRun() method first fires a PropertyChange and then schedules the first Arrival event as follows:

public void doRun() {

firePropertyChange("numberArrivals", getNumberArrivals());

waitDelay("Arrival", interarrivalTimeGenerator);

}

Each scheduling edge in a Simkit program is implemented by a call to waitDelay() method.

### Test implementation of Run event

Even though the ArrivalProcess class is not complete, you can still test what has been implemented so far. Executing a Simkit model involves setting the properties of Schedule (the Event List in Simkit), then calling Schedule.Reset() followed by Schedule.startSimulation(). Therefore, add the following to RunArrivalProcess at the end of the main() method:

Schedule.stopAtTime(15.0);

Schedule.setVerbose(true);

Schedule.reset();

Schedule.startSimulation();

Run the program; your output should now look like this:

ArrivalProcess.1

interarrivalTimeGenerator = Exponential (3.200)

\*\* Event List 0 -- Starting Simulation \*\*

0.000 Run <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- Starting Simulation \*\*

Time: 0.0000 CurrentEvent: Run [1]

\*\* Event List 0 -- \*\*

0.645 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 0.6455 CurrentEvent: Arrival [1]

\*\* Event List 0 -- \*\*

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 15.0000 CurrentEvent: Stop [1]

\*\* Event List 0 -- \*\*

<< empty >>

\*\* End of Event List -- \*\*

Note that an Arrival event is scheduled for time 0.6455 and seems to actually “occur” at that time, yet nothing happens and no other Arrival events are scheduled. That is because the waitDelay() method schedules an event whether it exists in that class or not. When this Arrival event occurs, since there is no such event (we haven’t implemented it yet), nothiong happens.

### Write doArrival()

The final addition to the ArrivalProcess class is to write the doArrival() method. Time-varying state variables, like numberArrivals, fire “Property Changes” which include the “old” value and the “new” value. So before actually making the state transition, the “old” value is saved and then passed as a the second argument to firePropertyChange(), as follows:

public void doArrival() {

int oldNumberArrivals = getNumberArrivals();

numberArrivals = numberArrivals + 1;

firePropertyChange("numberArrivals",oldNumberArrivals,

getNumberArrivals());

. . .

Finally, write the waitDelay() call for doArrival():

waitDelay("Arrival", interarrivalTimeGenerator);

Your ArrivalProcess class is now complete (except for the comments). In the RunArrivalProcess class, add the following line after Schedule.startSimulation():

System.out.println(“At time “ + Schedule.getSimTime() +

“ there have been “ + arrivalProcess.getNumberArrivals() + “ arrivals”

Re-run your model to get the following output:

ArrivalProcess.1

interarrivalTimeGenerator = Exponential (3.2)

\*\* Event List 0 -- Starting Simulation \*\*

0.000 Run <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- Starting Simulation \*\*

Time: 0.0000 CurrentEvent: Run [1]

\*\* Event List 0 -- \*\*

0.645 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 0.6455 CurrentEvent: Arrival [1]

\*\* Event List 0 -- \*\*

0.648 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 0.6485 CurrentEvent: Arrival [2]

\*\* Event List 0 -- \*\*

2.801 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 2.8010 CurrentEvent: Arrival [3]

\*\* Event List 0 -- \*\*

9.292 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

Time: 9.2922 CurrentEvent: Arrival [4]

\*\* Event List 0 -- \*\*

15.000 Stop <Stop.2>

19.982 Arrival <ArrivalProcess.1>

\*\* End of Event List -- \*\*

Time: 15.0000 CurrentEvent: Stop [1]

\*\* Event List 0 -- \*\*

19.982 Arrival <ArrivalProcess.1>

\*\* End of Event List -- \*\*

At time 15.0 there have been 4 arrivals

### Add SimplePropertyDumper to Main Class to verifyState Transitions

PropertyChanges are fired in case certain objects (PropertyChangeListeners) are interested in when the state changes value. One use of this machanism is a listener that simply prints the property, its old value, and its new value to the command line. An instance of the class SimplePropertyDumper does this. Add the following code to your main method after the ArrivalProcess is instantiated, but before the calls to Schedule are made:

SimplePropertyDumper simplePropertyDumper = new SimplePropertyDumper();

arrivalProcess.addPropertyChangeListener(simplePropertyDumper);

### Final test run

When you run it now, the result should be:

ArrivalProcess.1

interarrivalTimeGenerator = Exponential (3.200)

\*\* Event List 0 -- Starting Simulation \*\*

0.000 Run <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- Starting Simulation \*\*

numberArrivals: 0

Time: 0.0000 CurrentEvent: Run [1]

\*\* Event List 0 -- \*\*

0.645 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

numberArrivals: 0 => 1

Time: 0.6455 CurrentEvent: Arrival [1]

\*\* Event List 0 -- \*\*

0.648 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

numberArrivals: 1 => 2

Time: 0.6485 CurrentEvent: Arrival [2]

\*\* Event List 0 -- \*\*

2.801 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

numberArrivals: 2 => 3

Time: 2.8010 CurrentEvent: Arrival [3]

\*\* Event List 0 -- \*\*

9.292 Arrival <ArrivalProcess.1>

15.000 Stop <Stop.2>

\*\* End of Event List -- \*\*

numberArrivals: 3 => 4

Time: 9.2922 CurrentEvent: Arrival [4]

\*\* Event List 0 -- \*\*

15.000 Stop <Stop.2>

19.982 Arrival <ArrivalProcess.1>

\*\* End of Event List -- \*\*

Time: 15.0000 CurrentEvent: Stop [1]

\*\* Event List 0 -- \*\*

19.982 Arrival <ArrivalProcess.1>

\*\* End of Event List -- \*\*

At time 15.0 there have been 4 arrivals

Note the lines indicating state transitions for the numberArrivals state variable that appear just before each Event List snapshot. These are the result of the SimplePropertyDumper instance “hearing” the numberArrivals PropertyChangeEvent.

### Comment Your Code

Before it is considered complete, the important methods of your ArrivalProcess class must be commented as well as the RunArrivalProcess class. Please write brief and succinct, but decsriptive, comments. You do not have to comment setters or getters *unless* they do something “unusual.” Since nothing unusual happens here, comments for setters and getters will be considered optional.

## Deliverables

Push your Netbeans MV3302 project to Gitlab by the due date.

## Frequently Asked Questions

1. Why do I need a doRun() method?

Every simulation model needs to be initialized by putting at least one event on the event list at the beginning. Run is the one special event name that is always put on the event list. You create a Run event by writing a doRun() method.

1. What happens if I don't put a doRun() in my model?

Nothing. Literally. Try it and see.

1. How does Simkit know to put the doRun() method on the Event List?

It uses a feature of Java called reflection. You can read about reflection in your Java book, but you do not need to understand reflection to write effective models in Simkit.

1. Do I have to use waitDelay()? Why can't I invoke doRun() or doArrival() directly?

You should *never* directly invoke a ‘do’ method yourself, but should always use waitDelay(). The reason is that there is bookkeeping that must be done about which events have occurred and which events are scheduled to occur. Directly invoking a ‘do’ method circumvents this bookkeeping and will cause your model to behave strangely.

1. What is the purpose of the reset() method?

The primary purpose of reset() is to restore the state variables to their initial values. The wy the current assignment is executed, there is not a specific need for reset() because the initial value of numberArrivals is 0 and there is only one run here. However, when you start doing multiple simulation runs, reset() will be essential

1. My program compiles and runs, but after the first Arrival the Event List is empty. What gives?

You are probably mis-spelling “Arrival” in the call to waitDelay() in doRun(). Or you have spelled “doArrival” incorrectly. Check to make sure that you have spelled it correctly, including capitalization; for example, if you see ‘arrival’ on the Event list, then you have not capitalized it in your waitDelay() method. The same goes for the waitDelay() in doArrival().

1. What are setter and getter methods?

Instance variables in a class should never be directly accessed outside the class. If an instance variable is meant to have its value set by another object, the programmer should provide a setter method to do so. If other objects should be able to see what the value of the variable is, the programmer should provide a getter method. The Java convention for setter methods is set<name> and for getter methods is get<name>, where <name> is the name of the instance variable. So an instance variable called foo of type int would have a setter defined to be

public void setFoo(int f) { foo = f; }

and a getter defined to be

public int getFoo() { return foo; }

1. Why doesn't numberArrivals have a setter method?

The value of numberArrivals should be determined by the simulation model, not arbitrarily set by the program. At any time, the value of numberArrivals should be exactly equal to the number of Arrival events that have occurred. In general, state variables should not be given public setter methods, since their values are determined by what has occurred in the simulation.

1. Why can’t I just use ‘new’ to instantiate the RandomVariate?

Try it and see what happens. As it turns out, you will not be able to instantiate a RandomVariate because it is an interface, not a concrete class. As you will see with future models, the use of RandomVariateFactory to get a RandomVariate is very flexible and powerful; for now, you’ll just have to take my word for it.

1. What are those numbers in brackets next to the events on the Event List?

The numbers in brackets count the number of times each type of event has occurred in the current simulation run.

1. I’m not getting the parameter values in my output.

Make sure that you have defined *both* setters and getters for each of your parameters and that they match in name and type. A parameter called foo that is type int should be declared and have setters and getters as follows:

private int foo;

. . .

public void setFoo(int f) { foo = f; }

public int getFoo() { return foo; }

The int signature of the setter *must* match the return type of the getter. Also, the names must exactly match, including capitalization. The most reliable way to ensure that your getters and setters are correct is to have your IDE generate them. In NetBeans, for example, right-click on the variable and select Refactor | Encapsulate Fields.

1. Assuming you have first installed Simkit. [↑](#footnote-ref-1)
2. Although the code in the handout will not show comments, for brevity, your code should be commented. [↑](#footnote-ref-2)
3. In NetBeans, right-click on the variable and select Refactor | Encapsulate Fields. [↑](#footnote-ref-3)
4. If you generated your setters and getters using “Refactor | Encapsulate Fields” you will also have some generated comments in your code. [↑](#footnote-ref-4)
5. Both RandomVariate and RandomVariateFactory are in the simkit.random package. Use the same light bulb trick as before to generate the import. [↑](#footnote-ref-5)