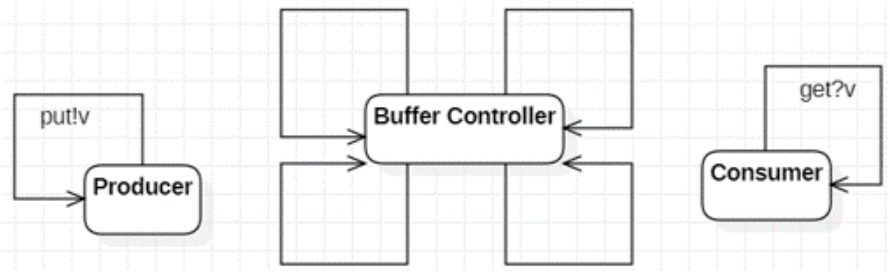
Assignment 5 Part 2

(to be done by the same team as in Part 1)

Due: Fri, December 11 (11:59 pm), for Parts 1 and 2

Part 2: Communicating State Charts



Download and unzip `A5_Part2.zip`. It has four files: `Circular_Buffer.mdj`, `Circular_Buffer.java`, `Circular_Buffer.txt`, and `MessagePassing.jar`.

File `Circular_Buffer.mdj` contains a ***StarUML State Chart*** (reproduced above) giving the outline of the controller for a circular buffer. This buffer is accessed by two *concurrent* processes, `Producer` and `Consumer`, who communicate with the buffer controller using two *channels,* `put` and `get`. The producer repeatedly does a `put!` and the consumer similarly does a `get?`, and the buffer controller performs the respective complementary operations.

File `Circular_Buffer.java` contains the outline of a Java implementation of the above state chart using the ***Message Passing Library*** discussed in Lecture 15 – examples were posted in `Sample Programs` → `MessagePassingExamples.zip`. The file gives the main program (class `Driver`), the classes `Producer` and `Consumer`, and also the outline of class `Circular_Buffer`.

Class `Circular_Buffer` uses an integer array, `data`, of size `n` in order to hold its data. It has a field `count` that gives, at any given time, the number of values that can be taken out of the buffer. It also has two indices `p` and `g` to point, respectively, to the places in the array where the next value is to be put by the producer and taken out by the consumer. These indices are incremented (modulo `n`) as put/get operations take place. The actual insertion and retrieval of values are performed by two methods `put()` and `get()` respectively.

What you should do:

1. Complete the state chart `Circular_Buffer.mdj` by providing suitable labels on the ***four transitions*** shown. Each label is of the form ***event [guard] / action*** – Lecture 26 includes a demo on how to construct StarUML State Charts. These labels should collectively express the synchronization policy of the buffer controller, namely, that:

   (i) when the buffer is empty (`count == 0`) only a `put` operation is permitted;
   (ii) when the buffer is full (`count == n`) only a `get` operation is permitted;
   (iii) otherwise, both `put` and `get` are permitted - the selection is non-deterministic.

In specifying the *transition labels* in ***StarUML***:

(i) each ***event*** is a channel send/receive, and is specified as a *Trigger Event*;
(ii) each ***guard*** is a boolean expression and specified in the *Properties* section; and
(iii) each ***action*** is a call on one of the methods `put()` or `get()`, and specified as an *Effect Behavior* → *OpaqueBehavior*.

2. Complete the `run()` method in class `Circular_Buffer` providing an implementation of the above synchronization policy. As the state chart specifies that the buffer controller operates in a repetitive cycle, the top level of the `run()` method should be a `while(true) {…}` loop.

   In Eclipse, right-click on the project, then select *Build Path* → *Configure Build Path* → *Add External JAR* → *browse and select* `MessagePassing.jar`.

   Run the completed program under JIVE after adding `Scheduler.*` to *Debug Configurations* → *JIVE* → *Exclusion Filter*. Check that the *Console* output shows the strings Put 1, …, Put 50 as well as Get 1, …, Get 50. The 'Put' and 'Get' strings will not be in strict alternation, but their respective values should be in ascending order – this property is to be checked as specified below.

   Save the *Execution Trace* in a file, `Circular_Buffer.csv`, and load it into the *Property Checker*:

   - Add `Driver.op` and `Driver.val` to the *Key Attributes*.
   - Enter `Driver.op = op, Driver.val=val` in the *Abbreviations* textbox.
   - Choose Method Granularity – this is important.
   - Copy the contents of the file `Circular_Buffer.txt` into the *Properties* textbox.
   - Press *Validate* and check that all properties are satisfied; otherwise, the program has an error which needs to be corrected.

What to Submit:

Prepare a top-level directory named *A5_Part2_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A5_Part2_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.) In this directory, place the updated `Circular_Buffer.mdj` and `Circular_Buffer.java`, and also `Circular_Buffer.csv`. Compress the directory and submit the compressed file using the `submit_cse522` command (grads) or the `submit_cse410` command (undergrads). Only one submission per team is required.

End of Assignment 5 Part 2