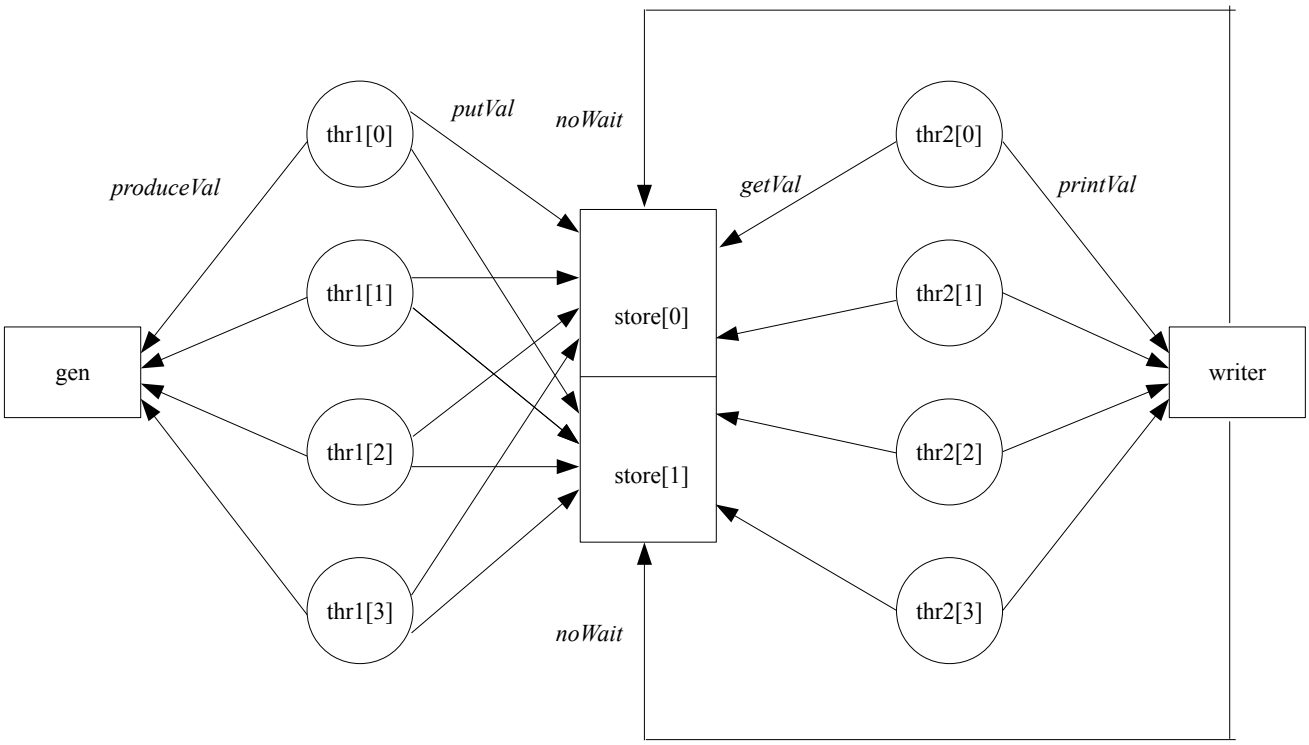
Solution

1. Representing the active entities by circles and the passive entities by squares, sketch a diagram which illustrates the interaction that is taking place and describe in simple words the role played by the threads of each type (do not use more than one or two sentences in your explanation).



Type 1 threads read successively values from *gen* and write them in double to *store*[0], if they are multiple of 3, and as they are to *store*[1], otherwise; they terminate when a null value is read. Type 2 threads read values from *store*[0], the first two, and from *store*[1], the last two, and print them in conjunction with the *ids* of the threads involved in its processing; they terminate when there are no new values to be read.

2. Assume that, when the program is run, one gets the following printing

```
The value processed by 1 and by 2 was 24.
The value processed by 3 and by 1 was 3.
The value processed by 1 and by 2 was 12.
The value processed by 2 and by 4 was 5.
The value processed by 4 and by 3 was 4.
The value processed by 4 and by 4 was 11.
The value processed by 3 and by 2 was 36.
The value processed by 1 and by 3 was 2.
The value processed by 4 and by 4 was 7.
The value processed by 2 and by 1 was 30.
The value processed by 4 and by 4 was 16.
The value processed by 1 and by 1 was 18.
```

   Bear in mind that, because of the randomness which was introduced, this is not the only possible result. In fact, it is not even correct. There are three errors in the printing at lines 2, 6 and 9, respectively. Identify them and justify your claims carefully.

The error in line 2 regards the value that was printed, 3. Being multiple of 3, the correct value should be 6.
The error in line 6 regards the value that was printed, 11. 11 is not stored in *gen*.
The error in line 9 is less obvious, it stems from a causality violation. The values processed by type 1 thread 4 and by type 2 thread 4, printed in lines 9 and 11, 7 and 16, respectively, appear in the reverse order of storage in *gen* (values processed by the same pair of threads must be printed in storage order).

3. Explain how the termination of the program is always ensured.

As their life cycle was designed, type 2 threads do not control directly their termination. When the last value is printed, the thread that has done it terminates immediately. However, the other threads are either blocked, or will be soon, in the implicit condition variables of the monitors *store*[0] and *store*[1]. Thus, the operation *noWait* is then called on by this thread on the two monitors to wake up the remaining threads and allow their termination.

4. Change the code so that it becomes possible to process pairs of values (the values of each pair should be added). Start your answer by pointing out the changes that have to be made. They should be minimal.

The minimal changes are:

1 - Upon instantiation of object *writer*, replace 12 by 6, since there are 6 pairs .

2 - The printing instruction of the method *printVal* must be adapted since there are now 2 type 1 threads and 1 type 2 to process the pair.

3 - The state diagram that describes the operation of data type *StoreRegion*, must be changed to include an extra state (two values must be accumulated in the internal variable *mem* before a 2 type thread is allowed to read the result), which implies that the methods *putVal*, *getVal* and *noWait* have to suffer minor changes.

I leave the writing of the code to you.