EXERCISE 2-1

//RunMultiplier.groovy

```
def processList = [ new Producer ( outChannel: connect1.out() ),
                    new Multiplier ( inChannel: connect1.in(), outChannel:
connect2.out(), factor: 4 ),
                    //insert here an instance of multiplier with a multiplication
factor of 4
                    new Consumer ( inChannel: connect2.in() )
                  ]
```

//Multiplier.groovy

```
while (i > 0) {
  outChannel.write(i * factor)  // write i * factor to outChannel
  i = inChannel.read()   // read in the next value of i
}
```

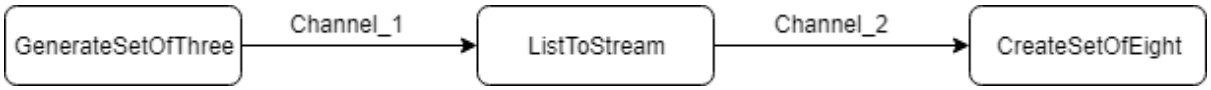
//Consumer.groovy

```
while ( i > 0 ) {
  println("The multiply value is $i")//insert a modified println statement
  i = inChannel.read()
}
```

//Output

```
next: 2
next: The multiply value is 8
3
next: The multiply value is 12
4
next: The multiply value is 16
5
next: The multiply value is 20
0
Finished

Process finished with exit code 0
```

EXERCISE 2-2



//GenerateSetsOfThree.groovy

```
for ( i in 0 ..< threeList.size)outChannel.write(threeList[i])

      outChannel.write([-1,-1,-1]) //write the terminating List as per exercise
definition
```

//ListToStream.groovy

```
while (inList[0] != -1) {
   // hint: output    list elements as single integers
   for (i in inList) {
      outChannel.write(i)
   }
   inList = inChannel.read()
}
```

//CreateSetsOfEight.groovy

```
while (v != -1){
   for ( i in 0 .. 7 ) {
      outList.add(v)
      v = inChannel.read()
      // put v into outList and read next input
   }
   println " Eight Object is ${outList}"
   outList = []
}
```

//Output

```
 Eight Object is [1, 2, 3, 4, 5, 6, 7, 8]
 Eight Object is [9, 10, 11, 12, 13, 14, 15, 16]
 Eight Object is [17, 18, 19, 20, 21, 22, 23, 24]
Finished

Process finished with exit code 0
```

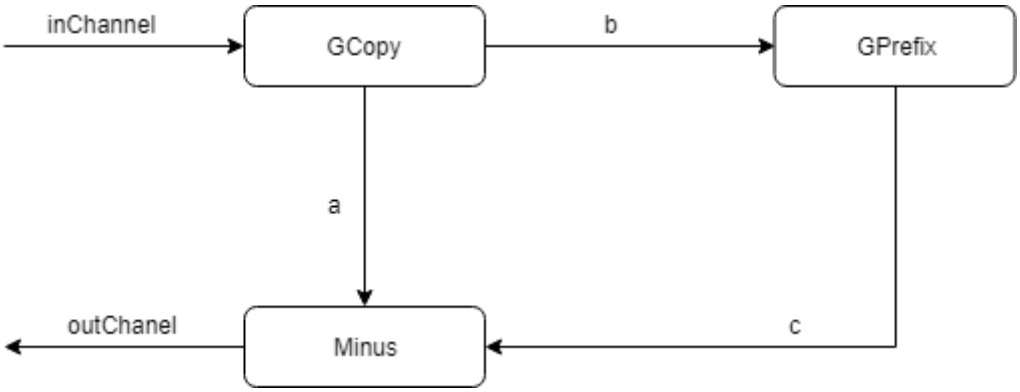
-The change requires is in the loop FOR, the loop specified the number of integers in each array.

-To parameterise the solution to a group size we can add a groupSize parameter and use it in the loop FOR.

-If the output require in the output stream is not a factor of the total number of integers in the input stream, the system will print out as many full group as it can but won't print the group that is not full.

EXERCISE 3-1

//First Version



//Differentiate.groovy

```
def differentiateList = [ new GPrefix ( prefixValue: 0,
                                        inChannel: b.in(),
                                        outChannel: c.out() ),
                          new GPCopy ( inChannel: inChannel,
                                       outChannel0: a.out(),
                                       outChannel1: b.out() ),
                          // insert a constructor for Minus
                          new  Minus ( inChannel0: a.in(),
                                         inChannel1: c.in(),
                                         outChannel: outChannel
                          )
             ]
```

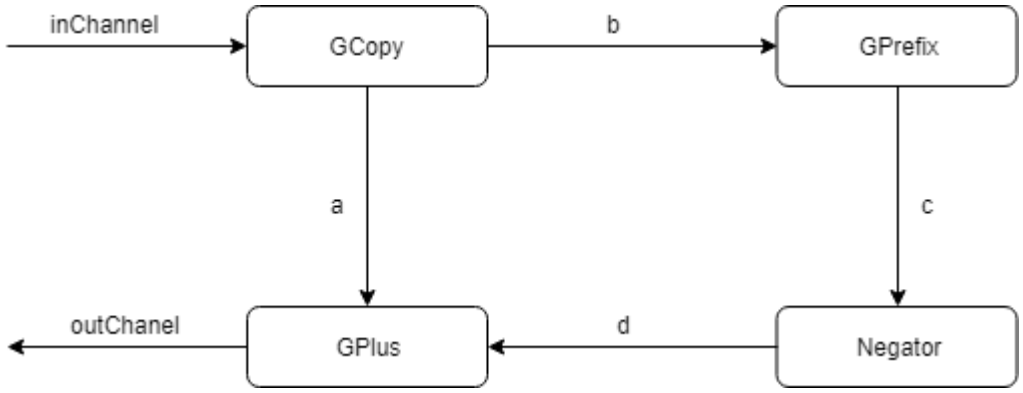
//Minus.groovy

```
while (true) {
  parRead2.run()
  outChannel.write(read0.value - read1.value)
  // output one value subtracted from the other
  // be certain you know which way round you are doing the subtraction!!
}
```

//Output

```
Differentiated Numbers
0
1
2
3
4
5
6
7
8
9
10
11
12
13
```

//Second Version



//DifferentiateNeg.groovy

```
def differentiateList = [ new GPrefix ( prefixValue: 0,
                                        inChannel: b.in(),
                                        outChannel: c.out() ),
                          new GPCopy ( inChannel: inChannel,
                                       outChannel0: a.out(),
                                       outChannel1: b.out() ),
                          //insert a constructor for Negator
                          new Negator( inChannel: c.in(),
                                       outChannel: d.out() ),
                          new GPlus  ( inChannel0: a.in(),
                                       inChannel1: d.in(),
                                       outChannel: outChannel )
                        ]
```

//Negator.groovy

```
while (true) {
  def num = inChannel.read()
  outChannel.write(-num )
  //output the negative of the input value
}
```

//Output

```
Differentiated Numbers
0
1
2
3
4
5
6
7
8
9
10
```

-Personally, I've found the first solution more pleasant. With this solution we implement the solution with one process less. Not sure if this will be the best solution in bigger problems, but for a small problem like this one I would avoid implement an extra process.

EXERCISE 3-2

//

//GSquares.groovy

```
def testList =  [ new GNumbers    ( outChannel: N2I.out() ),
                  new GIntegrate ( inChannel: N2I.in(),
                                   outChannel: I2P.out() ),
                  new GSPairsB ( inChannel: I2P.in(),
                          outChannel: outChannel )
]
```

//Output B

```
Squares
1
4
9
16
25
36
49
64
81
100
121
```

//Questions

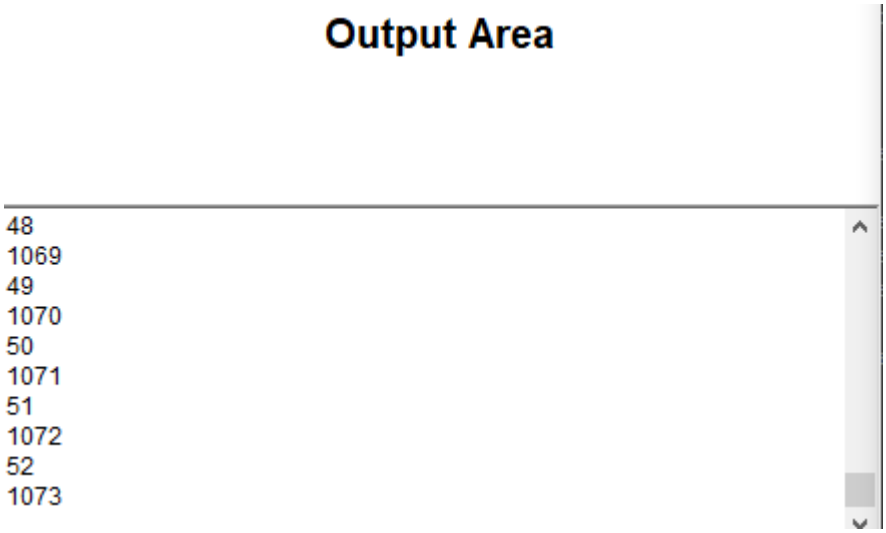
- With GSPairsA the application Deadlock without printing any data while with GSPairsB works perfectly.
- Because in GSPairsA GSCopy writes first to outChannel10 which is read by GPlus. After it, writes to outChannel11 which is read by GTails that by default read the first value but not retain it. Afterwards, GSCopy attemps to write to GPlus again via outChannel10 but the outChannel10 still have the first number so It can not get any other causing the Deadlock.

EXERCISE 3-3

- Because GParPrint is desing to handle all the printing and read from the input when is ready while GPrint is designed to print as soon as it get the input, which can create a very messy code which difficult the reading of the output data.

EXERCISE 4.1:

- It does not Reset the value, It start a new sequence and run both at the same time.
- If you add a second reset value to the sequence it stops working because there is just 3 Proccesors(Prefix, GCopy and GSuccesor) and any of those can get any other value without output the one that they have in, so it causes the DeadLock.



EXERCISE 4.2:

//ResetNumbers.groovy

```
def testList = [ new GPrefix ( prefixValue: initialValue,
                               outChannel: a.out(),
                               inChannel: c.in() ),
                 new GPCopy ( inChannel: a.in(),
                              outChannel0: outChannel,
                              outChannel1: b.out() ),
                 // requires a constructor for ResetSuccessor
                  new ResetSuccessor (
                            inChannel: b.in(),
                          outChannel: c.out(),
                          resetChannel: resetChannel


                  )
```

//ResetSuccessor.groovy

```
while (true) {
    def index = alt.priSelect()
    if (index == 0 ) {     // resetChannel input
        // deal with inputs from resteChannel and inChannel
        def resetValue = resetChannel.read()
        resetChannel.read()
        outChannel.write(resetValue)
    }
    // use a priSelect
    else {     //inChannel input
        outChannel.write(inChannel.read()+1)
    }
}
```

//Output

Output Area

```
1052
15
1053
16
1054
17
1055
18
1056
19
```

-It does not overcome the problem identify in the Exercise 1 because there is still no channel between Prefix and ResetSuccesor, so it cause a deadLock as soon as you add the third value.

EXERCISE 5-1

//RunQueue.groovy

```
def testList = [ new QProducer ( put: QP2Q.out(),
                                 iterations: 50,
                                 delay: 0),
                 new Queue ( put: QP2Q.in(),
                             get: QC2Q.in(),
                             receive: Q2QC.out(),
                             elements: 5 ),
                 new QConsumer ( get: QC2Q.out(),
                                 receive: Q2QC.in(),
                                 delay: 0 )
                ]
```

-Increasing the delay just make the program slower but it still works well because the program is synchronized.

EXERCISE 5-2

//Scale.groovy

```
while (true) {
  switch ( scaleAlt.priSelect(preCon) ) {
    case SUSPEND :
      //  deal with suspend input
      suspended = true
      preCon[SUSPEND] = false
      suspend.read()
      factor.write(scaling)
      println "Suspended"
      preCon[INJECT] = true
      break
    case INJECT:
      //  deal with inject input
      scaling = injector.read()
      println "Injected scaling is $scaling"
      suspended = false
      preCon[SUSPEND] = true
      preCon[INJECT] = false
      timeout = timer.read() + DOUBLE_INTERVAL
      timer.setAlarm(timeout)
      break
    case TIMER:
      //  deal with Timer input
      timeout = timer.read() + DOUBLE_INTERVAL
      timer.setAlarm(timeout)
      scaling = scaling * 2
      println "Normal Timer: new scaling is ${scaling}"
      break
    case INPUT:
      //   deal with Input channel
      def inValue = inChannel.read()
      def result = new ScaledData()
      result.original = inValue
      if (suspended == true)
        result.scaled = inValue
      else
        result.scaled = inValue * scaling
      outChannel.write(result)
      break
  } //end-switch
} //end-while
```

I believe the precondition solution to be more elegant mostly for three reason:

- In the nested loop solution there is two whiles, one inside the other one.
- In the nested loop solution, there is two different inChannels.
- The precondition solution allows the users to make any future changes much easier.

EXERCISE 6-1

//TestThreeToEight.groovy

```
package exercises.c2

import jcsp.lang.*
import groovyJCSP.*

class TestThreeToEight extends GroovyTestCase {

    void testQueue() {

        ByteArrayOutputStream output = new ByteArrayOutputStream()
        System.setOut(new PrintStream(output))

        One2OneChannel connect1 = Channel.createOne2One()
        One2OneChannel connect2 = Channel.createOne2One()

        def processList = [ new GenerateSetsOfThree ( outChannel: connect1.out() ),
                            new ListToStream ( inChannel: connect1.in(),
                                    outChannel: connect2.out() ),
                            new CreateSetsOfEight ( inChannel: connect2.in() )
        ]
        new PAR (processList).run()

        def expected = " Eight Object is [1, 2, 3, 4, 5, 6, 7, 8]\r\n" +
                " Eight Object is [9, 10, 11, 12, 13, 14, 15, 16]\r\n" +
                " Eight Object is [17, 18, 19, 20, 21, 22, 23, 24]\r\n" +
                "Finished\r\n"

        assertEquals (expected.toString(), output.toString())

        System.setOut(null)

    }
}
```

//Output

```
TestThreeToEight (exercises.c2)   212 ms
    testQueue                     212 ms

"C:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...

Process finished with exit code 0
```

EXERCISE 7-1

//Client.groovy

```
void run () {
  def iterations = selectList.size
  println "Client $clientNumber has $iterations values in $selectList"

  for ( i in 0 ..< iterations) {
    def key = selectList[i]
    requestChannel.write(key)
    println "C $clientNumber reque ${key}"
    def v = receiveChannel.read()
    println "C $clientNumber rec ${key}"
  }

  println "Client $clientNumber has finished"
}
```

//Deadlocked ScreenShot

```
C 1 reque 13
C 0 rec 12
C 1 rec 13
C 0 reque 3
C 1 reque 14
C 0 rec 3
C 1 rec 14
C 0 reque 14
C 0 rec 14
C 1 reque 15
C 1 rec 15
C 0 reque 15
C 0 rec 15
C 1 reque 6
C 0 reque 16
```

//NotDeadLoacked ScreenShot

```
C 1 rec 17
C 0 rec 7
C 1 reque 8
C 1 rec 8
C 0 reque 18
C 0 rec 18
C 1 reque 19
C 1 rec 19
C 0 reque 9
C 0 rec 9
C 1 reque 20
C 0 reque 10
C 1 rec 20
C 0 rec 10
Client 1 has finished
Client 0 has finished
```

-The deadlocked is produced when either both servers clients requests or receives at the same time.

EXERCISE 8-1

//Client.groovy

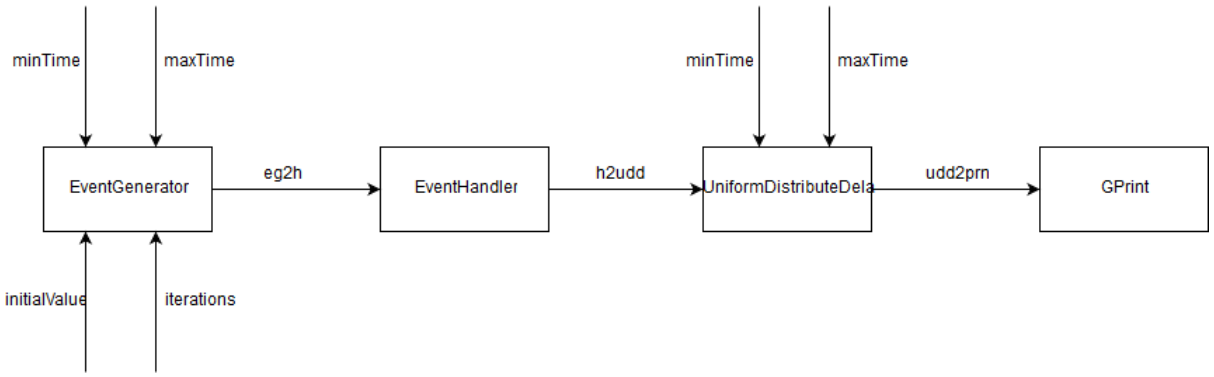
```
  def inOrder = true

  void run () {
    def iterations = selectList.size
    println "Client $clientNumber has $iterations values in $selectList"

    for ( i in 0 ..< iterations) {
      def key = selectList[i]
      requestChannel.write(key)
      //println "C $clientNumber reque ${key}"
      def v = receiveChannel.read()
      //println "C $clientNumber rec ${key}"
        if(v != key * 10)
          inOrder = false
    }

    println "Client $clientNumber has finished"
        if (inOrder == true)
          println "Client $clientNumber in order"
        else
          println "Client $clientNumber out of order"
  }
}
```

//Output

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
Client 1 has 10 values in [11, 12, 13, 4, 15, 16, 17, 18, 19, 20]
Client 0 has 10 values in [1, 2, 3, 4, 5, 6, 7, 18, 9, 10]
Client 0 has finished
Client 1 has finished
Client 0 in order
Client 1 in order
```

EXERCISE 9-1

//Diagram



//CountingMissing.groovy

```
def prev

void run () {
    def e = inChannel.read()
    prev =  e.data
    while (true) {
        prev = e.data
        e = inChannel.read()
        if (e.data != 100 && e.data != prev + e.missed +1) {
            passed = false
            println "Incorrect"
        }

        outChannel.write(e)

    }
}
```

//RunSingleStream.groovy

```
new UniformlyDistributedDelay ( inChannel:h2udd.in(),
                                outChannel: udd2c.out(),
                                minTime: 1000,
                                maxTime: 2000 ),

new CountingMissing ( inChannel: udd2c.in(),
                      outChannel: c2prn.out() ),


new GPrint ( inChannel: c2prn.in(),
        heading : "Event Output",
        delay: 0)
]
```

EXERCISE 9-2

The times associates with the Event Generation Steam is directly proportional to the number of missed events, decreasing them will increase the number of events missed and will make that source finish earlier than the rest (Test 1, Source 1) and increasing them will decrease the number of events missed and will make the source need more time to finish(Test 2, Source 1)

The times associate to the Event Processing System is also directly proportional to the number of missed events. Decreasing the min and the max decrease the number of event missed (Test 3) and increasing them also increase the number of event missed (Test 4).

Test1

```
minTimes = [ 0, 20, 30, 40, 50, 10, 20, 30, 40 ]
maxTimes = [ 5, 150, 200, 50, 60, 30, 60, 100, 80 ]
```

```
minTime: 10,
maxTime: 400
```

```
Number of event sources between 1 and 9 ? 9
Event Generator for source 7 has started
Event Generator for source 8 has started
Event Output
Event Generator for source 6 has started
Event Generator for source 4 has started
Event Generator for source 5 has started
Event Generator for source 2 has started
Event Generator for source 9 has started
Event Generator for source 1 has started
Event Generator for source 3 has started
EventData -> [source: 1, data: 100, missed: 0]
Source 1 has finished
EventData -> [source: 1, data: 101, missed: 0]
EventData -> [source: 6, data: 600, missed: 0]
EventData -> [source: 7, data: 700, missed: 0]
EventData -> [source: 8, data: 800, missed: 0]
EventData -> [source: 9, data: 900, missed: 0]
EventData -> [source: 1, data: 102, missed: 0]
EventData -> [source: 2, data: 200, missed: 0]
Source 6 has finished
EventData -> [source: 3, data: 300, missed: 0]
EventData -> [source: 4, data: 400, missed: 0]
EventData -> [source: 5, data: 500, missed: 0]
EventData -> [source: 6, data: 601, missed: 0]
EventData -> [source: 7, data: 707, missed: 6]
EventData -> [source: 8, data: 806, missed: 5]
EventData -> [source: 9, data: 915, missed: 14]
EventData -> [source: 1, data: 198, missed: 95]
```

Test2

```
minTimes = [ 195, 20, 30, 40, 50, 10, 20, 30, 40 ]
maxTimes = [ 200, 150, 200, 50, 60, 30, 60, 100, 80 ]
```

```
minTime: 1,
maxTime: 400
```

```
EventData -> [source: 1, data: 180, missed: 0]
EventData -> [source: 1, data: 182, missed: 1]
EventData -> [source: 1, data: 184, missed: 1]
EventData -> [source: 1, data: 185, missed: 0]
EventData -> [source: 1, data: 186, missed: 0]
EventData -> [source: 1, data: 187, missed: 0]
EventData -> [source: 1, data: 188, missed: 0]
EventData -> [source: 1, data: 189, missed: 0]
EventData -> [source: 1, data: 190, missed: 0]
EventData -> [source: 1, data: 191, missed: 0]
Source 1 has finished
EventData -> [source: 1, data: 193, missed: 1]
EventData -> [source: 1, data: 194, missed: 0]
EventData -> [source: 1, data: 196, missed: 1]
EventData -> [source: 1, data: 197, missed: 0]
EventData -> [source: 1, data: 198, missed: 0]
```

Test 3

```
minTimes = [ 10, 20, 30, 40, 50, 10, 20, 30, 40 ]
maxTimes = [ 100, 150, 200, 50, 60, 30, 60, 100, 80 ]
```

```
minTime: 1,
maxTime: 2
```

```
EventData -> [source: 7, data: 706, missed: 0]
EventData -> [source: 6, data: 612, missed: 0]
EventData -> [source: 5, data: 504, missed: 0]
EventData -> [source: 9, data: 904, missed: 0]
EventData -> [source: 6, data: 613, missed: 0]
EventData -> [source: 8, data: 803, missed: 0]
EventData -> [source: 4, data: 406, missed: 0]
EventData -> [source: 6, data: 614, missed: 0]
EventData -> [source: 7, data: 707, missed: 0]
EventData -> [source: 5, data: 505, missed: 0]
EventData -> [source: 6, data: 615, missed: 0]
EventData -> [source: 4, data: 407, missed: 0]
EventData -> [source: 2, data: 203, missed: 0]
EventData -> [source: 6, data: 616, missed: 0]
EventData -> [source: 8, data: 804, missed: 0]
EventData -> [source: 1, data: 104, missed: 0]
EventData -> [source: 9, data: 905, missed: 0]
EventData -> [source: 7, data: 708, missed: 0]
EventData -> [source: 6, data: 617, missed: 0]
EventData -> [source: 5, data: 506, missed: 0]
EventData -> [source: 4, data: 408, missed: 0]
EventData -> [source: 6, data: 618, missed: 0]
EventData -> [source: 9, data: 906, missed: 0]
EventData -> [source: 6, data: 619, missed: 0]
EventData -> [source: 2, data: 204, missed: 0]
EventData -> [source: 7, data: 709, missed: 0]
EventData -> [source: 6, data: 620, missed: 0]
EventData -> [source: 4, data: 409, missed: 0]
EventData -> [source: 5, data: 507, missed: 0]
EventData -> [source: 8, data: 805, missed: 0]
EventData -> [source: 6, data: 621, missed: 0]
EventData -> [source: 1, data: 105, missed: 0]
EventData -> [source: 3, data: 302, missed: 0]
```

Test 4

```
minTimes = [ 10, 20, 30, 40, 50, 10, 20, 30, 40 ]
maxTimes = [ 100, 150, 200, 50, 60, 30, 60, 100, 80 ]
```

```
minTime: 350,
maxTime: 400)
```

```
Source 4 has finished
EventData -> [source: 3, data: 303, missed: 2]
EventData -> [source: 4, data: 416, missed: 15]
EventData -> [source: 5, data: 519, missed: 18]
Source 1 has finished
Source 5 has finished
EventData -> [source: 6, data: 681, missed: 79]
Source 9 has finished
EventData -> [source: 7, data: 749, missed: 48]
EventData -> [source: 8, data: 835, missed: 34]
Source 8 has finished
EventData -> [source: 9, data: 944, missed: 43]
EventData -> [source: 1, data: 151, missed: 50]
EventData -> [source: 2, data: 242, missed: 40]
EventData -> [source: 3, data: 332, missed: 28]
Source 2 has finished
EventData -> [source: 4, data: 490, missed: 73]
EventData -> [source: 5, data: 580, missed: 60]
EventData -> [source: 6, data: 698, missed: 16]
EventData -> [source: 7, data: 798, missed: 48]
EventData -> [source: 8, data: 883, missed: 47]
EventData -> [source: 9, data: 998, missed: 53]
EventData -> [source: 1, data: 198, missed: 46]
EventData -> [source: 2, data: 282, missed: 39]
Source 3 has finished
EventData -> [source: 3, data: 366, missed: 33]
EventData -> [source: 4, data: 498, missed: 7]
EventData -> [source: 5, data: 598, missed: 17]
EventData -> [source: 8, data: 898, missed: 14]
```

EXERCISE 9-3

After see how the three multiplexer works I can determinate than the FairMultiplexer is the one that miss less data. The PriMultiplexer is a little bit slower than the FairMultiplexer but miss big chunks of data in once in few of the sources. And the basic Multiplexer, which miss more data than the FairMultiplexer but looks like the slowest and also lose more data than the FairMultiplexer.

/Multiplexer

```
Number of event sources between 1 and 9 ? 3
Event Output
Event Generator for source 1 has started
Event Generator for source 3 has started
Event Generator for source 2 has started
EventData -> [source: 1, data: 100, missed: 0]
EventData -> [source: 1, data: 101, missed: 0]
EventData -> [source: 2, data: 200, missed: 0]
EventData -> [source: 3, data: 300, missed: 0]
EventData -> [source: 1, data: 102, missed: 0]
EventData -> [source: 2, data: 201, missed: 0]
EventData -> [source: 3, data: 301, missed: 0]
EventData -> [source: 1, data: 111, missed: 8]
EventData -> [source: 2, data: 208, missed: 6]
EventData -> [source: 3, data: 312, missed: 10]
EventData -> [source: 1, data: 131, missed: 19]
EventData -> [source: 2, data: 222, missed: 13]
EventData -> [source: 3, data: 322, missed: 9]
EventData -> [source: 1, data: 144, missed: 12]
EventData -> [source: 2, data: 233, missed: 10]
EventData -> [source: 3, data: 326, missed: 3]
EventData -> [source: 1, data: 151, missed: 6]
EventData -> [source: 2, data: 237, missed: 3]
EventData -> [source: 3, data: 329, missed: 2]
EventData -> [source: 1, data: 163, missed: 11]
EventData -> [source: 2, data: 243, missed: 5]
EventData -> [source: 3, data: 335, missed: 5]
EventData -> [source: 1, data: 176, missed: 12]
Source 1 has finished
EventData -> [source: 2, data: 253, missed: 9]
EventData -> [source: 3, data: 342, missed: 6]
EventData -> [source: 1, data: 186, missed: 9]
EventData -> [source: 2, data: 266, missed: 12]
EventData -> [source: 3, data: 346, missed: 3]
EventData -> [source: 1, data: 198, missed: 11]
EventData -> [source: 2, data: 275, missed: 8]
EventData -> [source: 3, data: 354, missed: 7]
EventData -> [source: 2, data: 280, missed: 4]
Source 2 has finished
EventData -> [source: 3, data: 361, missed: 6]
EventData -> [source: 2, data: 286, missed: 5]
EventData -> [source: 3, data: 365, missed: 3]
EventData -> [source: 2, data: 294, missed: 7]
EventData -> [source: 3, data: 370, missed: 4]
EventData -> [source: 2, data: 298, missed: 3]
EventData -> [source: 3, data: 373, missed: 2]
EventData -> [source: 3, data: 376, missed: 2]
EventData -> [source: 3, data: 377, missed: 0]
EventData -> [source: 3, data: 378, missed: 0]
EventData -> [source: 3, data: 379, missed: 0]
EventData -> [source: 3, data: 383, missed: 3]
EventData -> [source: 3, data: 384, missed: 0]
EventData -> [source: 3, data: 386, missed: 1]
EventData -> [source: 3, data: 388, missed: 1]
EventData -> [source: 3, data: 389, missed: 0]
EventData -> [source: 3, data: 390, missed: 0]
Source 3 has finished
EventData -> [source: 3, data: 391, missed: 0]
```

/PriMultiplexer

```
Number of event sources between 1 and 9 ? 3
Event Generator for source 2 has started
Event Generator for source 1 has started
Event Output
Event Generator for source 3 has started
EventData -> [source: 3, data: 300, missed: 0]
EventData -> [source: 1, data: 100, missed: 0]
EventData -> [source: 1, data: 101, missed: 0]
EventData -> [source: 1, data: 104, missed: 2]
EventData -> [source: 1, data: 107, missed: 2]
EventData -> [source: 1, data: 116, missed: 8]
EventData -> [source: 1, data: 119, missed: 2]
EventData -> [source: 1, data: 125, missed: 5]
EventData -> [source: 1, data: 131, missed: 5]
EventData -> [source: 1, data: 134, missed: 2]
EventData -> [source: 1, data: 135, missed: 0]
EventData -> [source: 1, data: 140, missed: 4]
EventData -> [source: 1, data: 141, missed: 0]
EventData -> [source: 1, data: 143, missed: 1]
EventData -> [source: 1, data: 145, missed: 1]
EventData -> [source: 1, data: 147, missed: 1]
EventData -> [source: 1, data: 148, missed: 0]
EventData -> [source: 1, data: 154, missed: 5]
EventData -> [source: 1, data: 160, missed: 5]
EventData -> [source: 1, data: 165, missed: 4]
EventData -> [source: 1, data: 168, missed: 2]
EventData -> [source: 1, data: 172, missed: 3]
EventData -> [source: 1, data: 173, missed: 0]
Source 1 has finished
EventData -> [source: 1, data: 179, missed: 5]
EventData -> [source: 1, data: 187, missed: 7]
EventData -> [source: 1, data: 193, missed: 5]
EventData -> [source: 1, data: 198, missed: 4]
EventData -> [source: 2, data: 200, missed: 0]
EventData -> [source: 2, data: 269, missed: 68]
EventData -> [source: 2, data: 271, missed: 1]
EventData -> [source: 2, data: 276, missed: 4]
EventData -> [source: 2, data: 277, missed: 0]
EventData -> [source: 2, data: 279, missed: 1]
EventData -> [source: 2, data: 282, missed: 2]
EventData -> [source: 2, data: 283, missed: 0]
EventData -> [source: 2, data: 284, missed: 0]
EventData -> [source: 2, data: 285, missed: 0]
Source 2 has finished
EventData -> [source: 2, data: 286, missed: 0]
EventData -> [source: 2, data: 288, missed: 1]
EventData -> [source: 2, data: 293, missed: 4]
EventData -> [source: 2, data: 297, missed: 3]
EventData -> [source: 2, data: 298, missed: 0]
EventData -> [source: 3, data: 301, missed: 0]
EventData -> [source: 3, data: 368, missed: 66]
EventData -> [source: 3, data: 370, missed: 1]
EventData -> [source: 3, data: 372, missed: 1]
EventData -> [source: 3, data: 373, missed: 0]
EventData -> [source: 3, data: 374, missed: 0]
EventData -> [source: 3, data: 376, missed: 1]
```

/FairMultiplexer

```
Number of event sources between 1 and 9 ? 3
Event Generator for source 1 has started
Event Generator for source 2 has started
Event Generator for source 3 has started
Event Output
EventData -> [source: 1, data: 100, missed: 0]
EventData -> [source: 1, data: 101, missed: 0]
EventData -> [source: 2, data: 200, missed: 0]
EventData -> [source: 3, data: 300, missed: 0]
EventData -> [source: 1, data: 102, missed: 0]
EventData -> [source: 2, data: 202, missed: 1]
EventData -> [source: 3, data: 303, missed: 2]
EventData -> [source: 1, data: 111, missed: 8]
EventData -> [source: 2, data: 206, missed: 3]
EventData -> [source: 3, data: 308, missed: 4]
EventData -> [source: 1, data: 120, missed: 8]
EventData -> [source: 2, data: 209, missed: 2]
EventData -> [source: 3, data: 312, missed: 3]
EventData -> [source: 1, data: 129, missed: 8]
EventData -> [source: 2, data: 219, missed: 9]
EventData -> [source: 3, data: 317, missed: 4]
EventData -> [source: 1, data: 138, missed: 8]
EventData -> [source: 2, data: 224, missed: 4]
EventData -> [source: 3, data: 323, missed: 5]
EventData -> [source: 1, data: 150, missed: 11]
EventData -> [source: 2, data: 234, missed: 9]
EventData -> [source: 3, data: 329, missed: 5]
EventData -> [source: 1, data: 160, missed: 9]
EventData -> [source: 2, data: 238, missed: 3]
EventData -> [source: 3, data: 332, missed: 2]
EventData -> [source: 1, data: 163, missed: 2]
EventData -> [source: 2, data: 241, missed: 2]
EventData -> [source: 3, data: 336, missed: 3]
Source 1 has finished
EventData -> [source: 1, data: 173, missed: 9]
EventData -> [source: 2, data: 253, missed: 11]
EventData -> [source: 3, data: 344, missed: 7]
EventData -> [source: 1, data: 191, missed: 17]
EventData -> [source: 2, data: 264, missed: 10]
EventData -> [source: 3, data: 351, missed: 6]
EventData -> [source: 1, data: 198, missed: 6]
EventData -> [source: 2, data: 268, missed: 3]
EventData -> [source: 3, data: 357, missed: 5]
EventData -> [source: 2, data: 276, missed: 7]
EventData -> [source: 3, data: 360, missed: 2]
EventData -> [source: 2, data: 279, missed: 2]
EventData -> [source: 3, data: 361, missed: 0]
EventData -> [source: 2, data: 280, missed: 0]
EventData -> [source: 3, data: 363, missed: 1]
EventData -> [source: 2, data: 281, missed: 0]
EventData -> [source: 3, data: 364, missed: 0]
EventData -> [source: 2, data: 285, missed: 3]
EventData -> [source: 3, data: 369, missed: 4]
Source 2 has finished
EventData -> [source: 2, data: 289, missed: 3]
```

EXERCISE 11-1

//RunScaler.groovy

```
def data = Channel.one2one()
def timedData = Channel.one2one()
def scaledData = Channel.one2one()
def oldScale = Channel.one2one()
def newScale = Channel.one2one()
def pause = Channel.one2one()

def network = [ new GNumbers ( outChannel: data.out() ),
                new GFixedDelay ( delay: 1000,
                        inChannel: data.in(),
                        outChannel: timedData.out() ),

                new Scale ( inChannel: timedData.in(),
                        outChannel: scaledData.out(),
                        factor: oldScale.out(),
                        suspend: pause.in(),
                        injector: newScale.in(),
                        multiplier: 2,
                        scaling: 2),

                new ControllerInterface ( inChannel: oldScale.in(),
                        suspend: pause.out(),
                        inject: newScale.out(),
                        print: scaledData.in(),
                        initialScale: 2 )

]

new
```

//ControllerInterface.groovy

```
    void run() {
        def controllerCanvas = new ActiveCanvas()
        def scaleConfig = Channel.one2one()
        def suspendConfig = Channel.one2one()
        def uiEvents = Channel.any2one( new OverWriteOldestBuffer(5) )
        def network = [ new ControllerManager ( fromScale: inChannel,
                toScaleSuspend: suspend,
                toScaleInject: inject,
                buttons: uiEvents.in(),
                toUISuspend: suspendConfig.out(),
                toUILabel: scaleConfig.out(),
                START_SCALE: initialScale ),
                        new UserInterface   ( controllerCanvas: controllerCanvas,
                                canvasSize: canvasSize,
                                scaleValueConfig: scaleConfig.in(),
                                suspendButtonConfig: suspendConfig.in(),
                                printValueConfig: print,
                                buttonEvent: uiEvents.out()  )
        ]
        new PAR ( network ).run()
    }
}
```

//Scale.groovy

```
   void run () {
        def SECOND = 1000
        def DOUBLE_INTERVAL = 5 * SECOND
        def NORMAL_SUSPEND  = 0
        def NORMAL_TIMER    = 1
        def NORMAL_IN       = 2
        def SUSPENDED_INJECT = 0
        def SUSPENDED_IN     = 1
        def timer = new CSTimer()
        def normalAlt = new ALT ( [ suspend, timer, inChannel ] )
        def suspendedAlt = new ALT ( [ injector, inChannel ] )
        def timeout = timer.read() + DOUBLE_INTERVAL
        timer.setAlarm ( timeout )

        outChannel.write( "Original                 Scaled\n" )

        while (true) {
            switch ( normalAlt.priSelect() ) {

                case NORMAL_SUSPEND :
                    suspend.read()
                    def suspended = true
                    outChannel.write( "Suspended\n" )
                    while ( suspended ) {

                        switch ( suspendedAlt.priSelect() ) {

                            case SUSPENDED_INJECT:
                                scaling = injector.read()
                                outChannel.write( "Injected scaling is $scaling\n" )
                                suspended = false
                                timeout = timer.read() + DOUBLE_INTERVAL
                                timer.setAlarm ( timeout )
                                break

                            case SUSPENDED_IN:
                                def inValue = inChannel.read()
                                def result = new ScaledData()
                                result.original = inValue
                                result.scaled = inValue
                                outChannel.write ( result.toString() + "\n" )
                                break
                        }  // end-switch
                    } //end-while
                    break

                case NORMAL_TIMER:
                    timeout = timer.read() + DOUBLE_INTERVAL
                    timer.setAlarm ( timeout )
                    scaling = scaling * multiplier
                    outChannel.write( "Normal Timer: new scaling is $scaling\n")
                    break

                case NORMAL_IN:
                    def inValue = inChannel.read()
                    def result = new ScaledData()
                    result.original = inValue
                    result.scaled = inValue * scaling
                    outChannel.write ( result.toString() + "\n" )
                    break

            } //end-switch
        } //end-while
    } //end-run
} // end Scalecale
```

//ControllerManager

```
def ChannelInput print

def ChannelInput fromScale
def ChannelOutput toScaleSuspend
def ChannelOutput toScaleInject
def int START_SCALE
def ChannelInput buttons
def ChannelOutput toUILabel
def ChannelOutput toUISuspend

void run() {
    def scale = START_SCALE
    toUILabel.write( scale.toString() )


    while (true) {

        def buttonDoing = buttons.read() // Read buttonDoing from UI
        if (buttonDoing == "Suspend Scaler") {
            toUISuspend.write("Scaler is suspended") // Change button to suspended when clicked
            toScaleSuspend.write(0)// Suspend Scale process
            scale = fromScale.read() // Send scaler and send it to Interface
            toUILabel.write(scale.toString())
            def integerScale = false// Wait till buttonDoing is an integer value
            while (!integerScale) {
                buttonDoing = buttons.read()
                try {
                    buttonDoing = Integer.parseInt(buttonDoing)// Get new scale value from buttonDoing
                    scale = buttonDoing
                    integerScale = true
                } catch (NumberFormatException e) {
                    System.out.println("Wrong number");
                }
            }
            toUISuspend.write("Suspend Scaler")// Change button back to suspend
            toUILabel.write(scale.toString())
            toScaleInject.write(scale)
        }

    }
}
```

//ScaledData.groovy

```
def int original
def int scaled

def String toString () {
    def s = " " + original + "\t\t" + scaled
    return s
}
```

//UserInterface.groovy

```
def ActiveCanvas controllerCanvas
def int canvasSize
def ChannelInput scaleValueConfig
def ChannelInput suspendButtonConfig
def ChannelInput printValueConfig
def ChannelOutput buttonEvent


void run() {
    def root = new ActiveClosingFrame ("Scaling Controller")
    def mainFrame = root.getActiveFrame()

    def gprint = new ActiveTextArea(printValueConfig, null) // Printing Current Values
    def suspendButton = new ActiveButton(suspendButtonConfig, buttonEvent, "SUSPEND")// Suspend button
    def newScaleLabel = new Label("Enter New Scale")// New scale
    newScaleLabel.setAlignment(Label.CENTER)
    def newScale = new ActiveTextEnterField(null, buttonEvent) //Getting new factor
    Panel newScalePanel = new Panel (new GridLayout (2, 2))
    newScalePanel.add (newScale.getActiveTextField ())

    def scaleValue = new ActiveLabel (scaleValueConfig)// Printing new scaler with label
    scaleValue.setAlignment(Label.LEFT)
    def scaleLabel = new Label ("Current Scale ")
    scaleLabel.setAlignment(Label.RIGHT)

    def printCont = new Container() // Setting up container
    printCont.setLayout(new GridLayout (1, 1))
    printCont.add(gprint)
    def currentCont = new Container()
    currentCont.setLayout(new GridLayout (1, 1))
    currentCont.add(scaleLabel)
    currentCont.add(scaleValue)
    def newCont = new Container()
    newCont.setLayout(new GridLayout (2, 1))
    newCont.add(newScaleLabel)
    newCont.add(newScalePanel)
    def susCont = new Container()
    susCont.setLayout(new GridLayout (1, 1))
    susCont.add(suspendButton)

    mainFrame.setLayout(new BorderLayout())
    mainFrame.add(printCont, BorderLayout.CENTER)
    mainFrame.add(currentCont, BorderLayout.NORTH)
    mainFrame.add(newCont, BorderLayout.EAST)
    mainFrame.add(susCont, BorderLayout.SOUTH)

    mainFrame.pack()
    mainFrame.setVisible(true)

    def network = [ root, controllerCanvas, gprint, scaleValue, newScale, suspendButton]
    new PAR (network).run()

}
```

//Interface