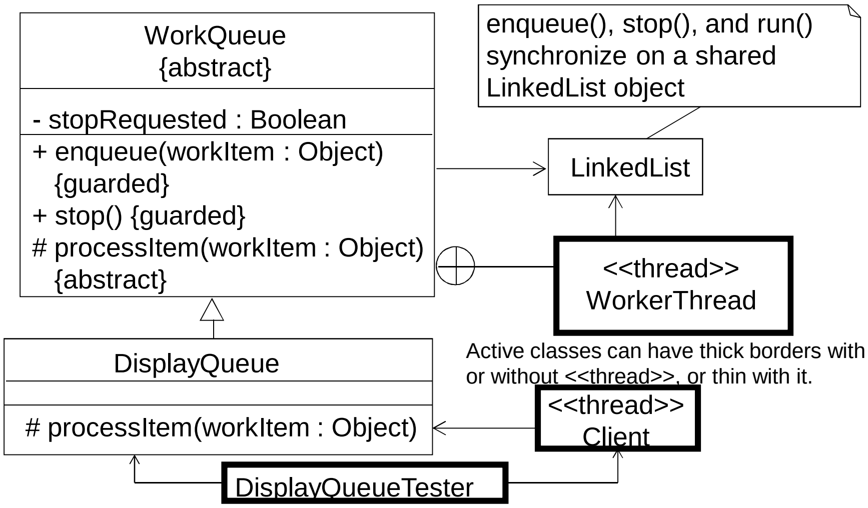
SYSC 3303 Real-Time Concurrent Systems

Assignment 4

[redacted]

Class Diagram (Required)



WorkQueue
{abstract}

- stopRequested : Boolean

+ enqueue(workItem : Object)
{guarded}
+ stop() {guarded}
# processItem(workItem : Object)
{abstract}

enqueue(), stop(), and run()
synchronize on a shared
LinkedList object

LinkedList

<<thread>>
WorkerThread

Active classes can have thick borders with or without <<thread>>, or thin with it.

DisplayQueue

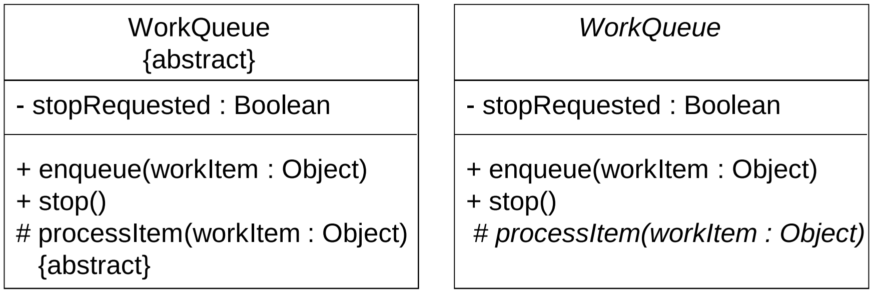
# processItem(workItem : Object)

<<thread>>
Client

DisplayQueueTester

Notes about Class Diagram

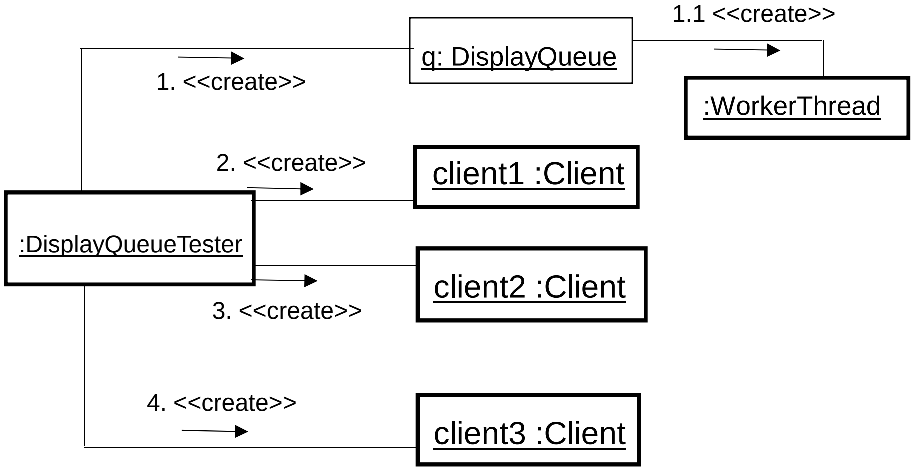
- Abstract things can be specified by rendering their names in *italics*, instead of using the `{abstract}` property


WorkQueue
{abstract}
- stopRequested : Boolean
+ enqueue(workItem : Object)
+ stop()
# processItem(workItem : Object) {abstract}
WorkQueue
- stopRequested : Boolean
+ enqueue(workItem : Object)
+ stop()
# processItem(workItem : Object)

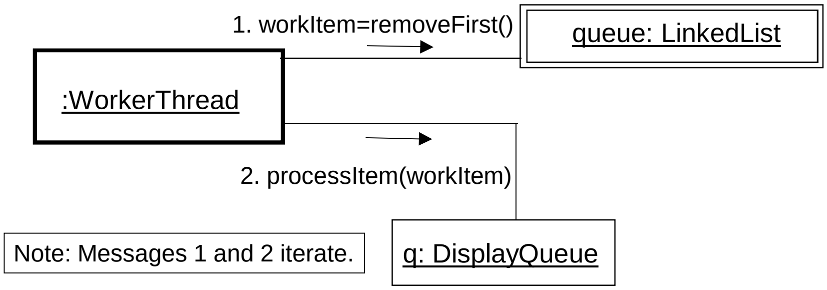
Collaboration Diagrams

- The assignment asked you to draw one collaboration diagram for each active object
- In addition, you could also combine the diagrams into one larger one
- You did not have to show the diagrams including the commented out `displayQ.stop()`in Client's run method, but you could have. (Slide 8 shows a version with this code uncommented.)

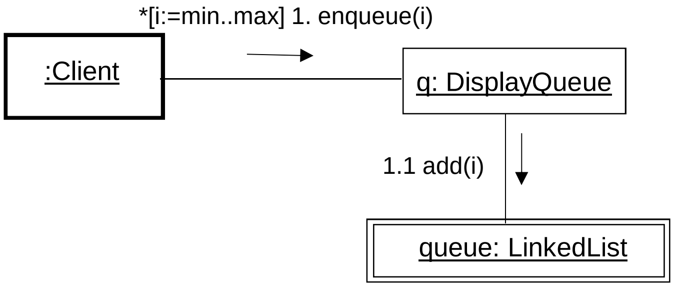
DisplayQueueTester Collaboration Diagram



WorkerThread Collaboration Diagram



Client Collaboration Diagram (At Least One Required)



*[i:=min..max] 1. enqueue(i)

:Client

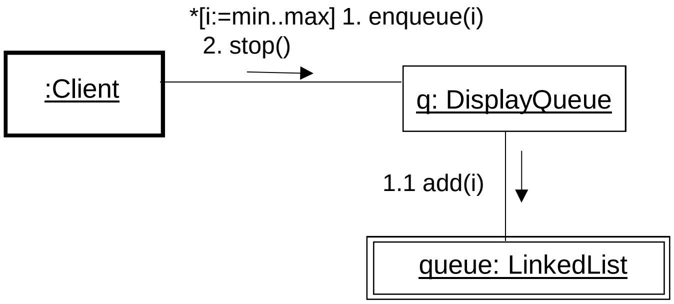
q: DisplayQueue

1.1 add(i)

queue: LinkedList

Note: The same collaboration diagram applies for client1, client2, and client3.

Client Collaboration Diagram with stop() shown (Optional)



*[i:=min..max] 1. enqueue(i)
2. stop()

:Client

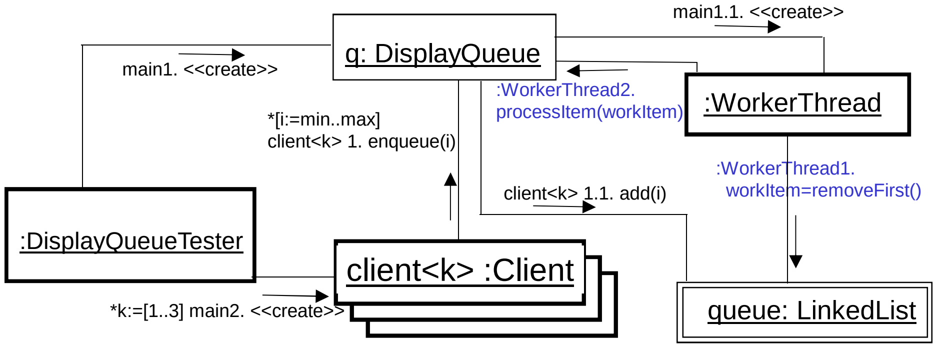
q: DisplayQueue

1.1 add(i)

queue: LinkedList

Note: The same collaboration diagram applies for client1, client2, and client3.

Combined Collaboration Diagram (Optional)



main1.1. <<create>>

q: DisplayQueue

main1. <<create>>

:WorkerThread2.
processItem(workItem)

:WorkerThread

*[i:=min..max]
client<k> 1. enqueue(i)

:WorkerThread1.
workItem=removeFirst()

client<k> 1.1. add(i)

:DisplayQueueTester

client<k> :Client

*k:=[1..3] main2. <<create>>

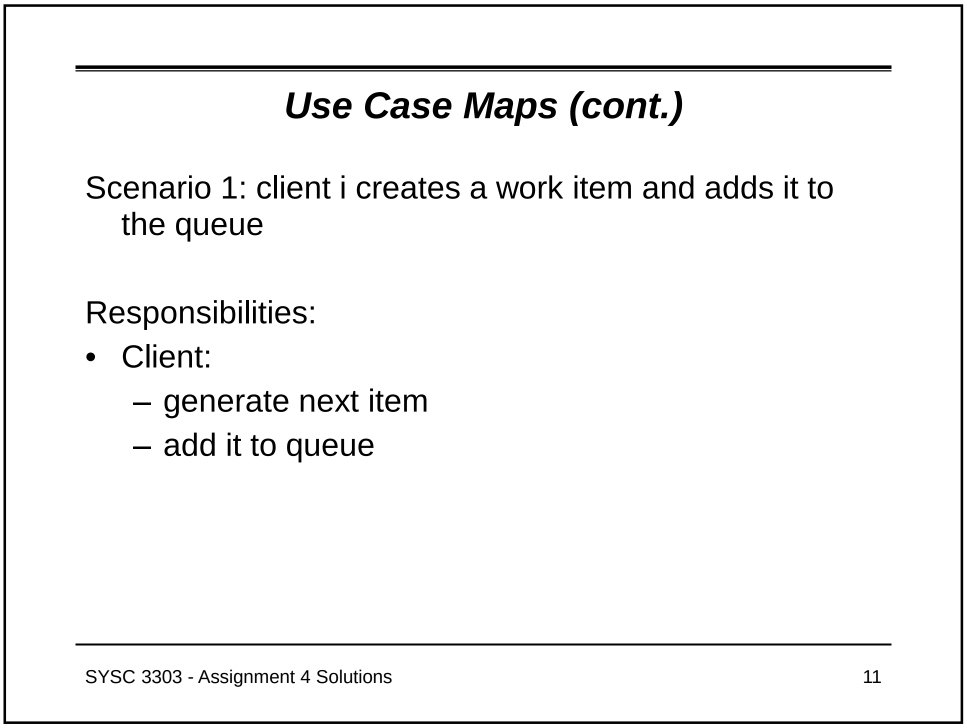
queue: LinkedList

Note: The three client threads could each be shown separately.

Note: WorkerThread Msgs 1 & 2 iterate.

Use Case Maps (UCMs)

- Scenarios (you did not have to include these in your solution):
  - client i creates a work item and adds it to queue
  - worker thread consumes a work item
  - Note: we could also consider this to be one scenario: client creates a work item, and worker consumes it as per slide 27 of the UCM package, giving the resulting UCM on slide 16 of this package.

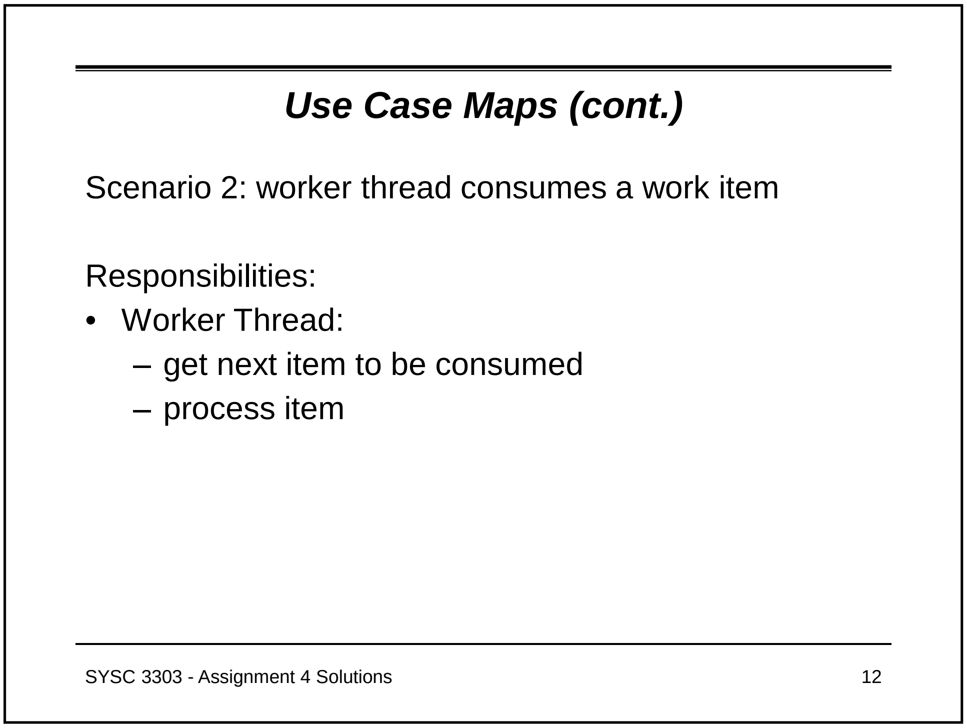


Use Case Maps (cont.)

Scenario 1: client i creates a work item and adds it to the queue

Responsibilities:

- Client:
  - – generate next item
  - – add it to queue



Use Case Maps (cont.)

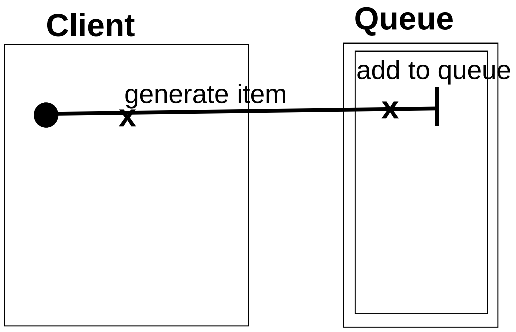
Scenario 2: worker thread consumes a work item

Responsibilities:

- Worker Thread:
  - – get next item to be consumed
  - – process item

Use Case Map: Map 1 of 2

Note: You had to include one UCM of the system or one for each scenario in your solution.



Client

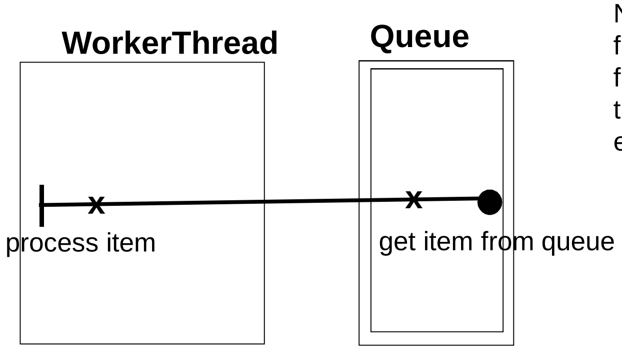
Queue

generate item

add to queue

Note: we are **done** after max-min+1 items have been generated and added to the queue.

Use Case Map: Map 2 of 2



WorkerThread

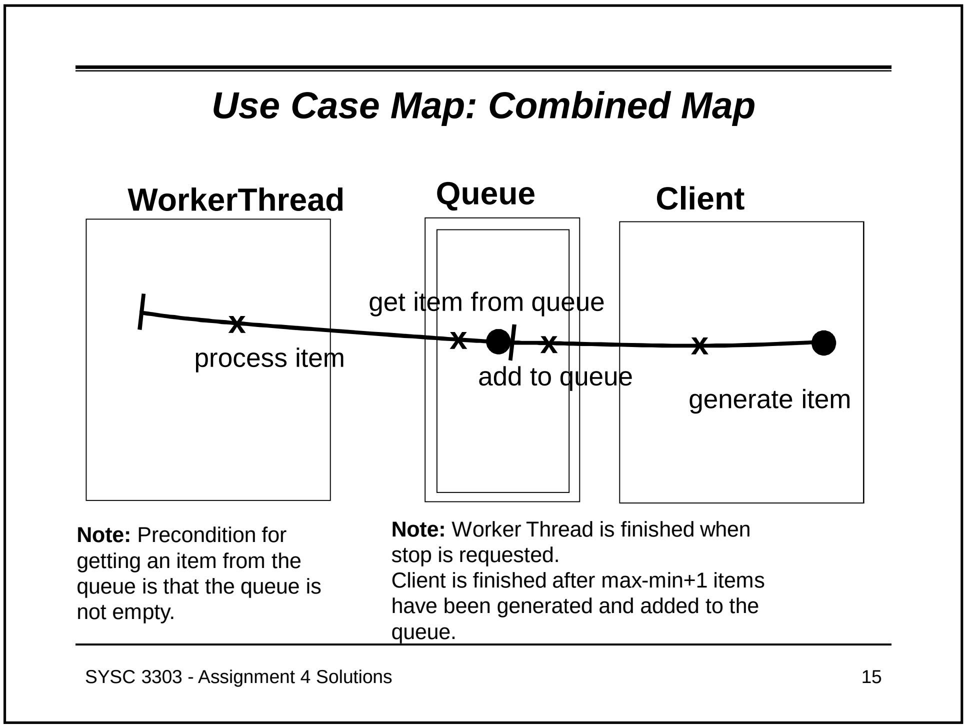
Queue

process item

get item from queue

Note: Precondition for getting an item from the queue is that the queue is not empty.

Note: we are **done** when stop is requested.



Use Case Map: Combined Map

Note: Precondition for getting an item from the queue is that the queue is not empty.

Note: Worker Thread is finished when stop is requested.
Client is finished after max-min+1 items have been generated and added to the queue.