A similar example for the University Registration System (a list of items is processed)

Process waitlist for a section: Clerk(trusted actor) makes a request to process a waitlist for a section, and provides section id. If section is found, system displays all students in section's waitlist one by one, and for each student, the clerk is asked if student should be registered now, or left on waitlist, or dropped from waitlist(i.e, not being allowed to register for section). With each student, system will use the response to act accordingly (register student in section and debit tuition to student's balance, leave as is, or remove from waitlist; note that if student is being registered, they should also be removed from waitlist).

Detailed use case:

Actor	System response
1. Request to process waitlist	
	2. Ask for sectionId
3. Enter sectionId	
	4. If section does not exist, or waitlist is empty, exit with message.
	5. Display name and studentId of next student. Ask if student should be registered, dropped from waitlist, or left on waitlist.
6. Enter response	
	7. If student is to be registered, system registers student in section and debit tuition to student's balance, and removes student from waitlist. If student is to be dropped, they are removed from waitlist; otherwise no action is taken. If more students in waitlist, goto Step 5.
	8. Print message that there are no more students and exit

Sequence of steps to be performed:

The UI initiates all the actions listed in the System Response column.
Step 2 is trivial.
For Step 4 through 7, UI does the following:
1. calls a method in the façade (Registration System) to search for the section; façade delegates the action to SectionList.
2. if section is found, asks façade to get waitlist for section; façade delegates to section; section returns an iterator [Note: 1 and 2 can be combined; getting waitlist for a sectionID requires search anyway]
3. UI loops through iterator, requesting actor response for each student
    3.1 If response is remove_from_waitlist, asks façade to remove student by calling the method removeFromWaitlist(sectionId, studentId) on façade.
    3.2 If response is to register student, asks façade to register student by calling the method registerStudent(sectionId, studentId); then calls method to remove student
    3.3 Reports result

Patterns Used: Façade is implicit; need to use iterator to access list.

Actor	**UI**	**Facade**	**SectionList**	**Section**		**Student**
1:Process Waitlist for section	2:getWait List(secId)	3:search Section (sectId)	4: Return Section			
		5:getWait List()		6:create iterator	**7:Waitlist (Iterator)**	
		9:return iterator		8:return iterator		
	10:Loop (while hasNext())					
	11:getStudent()					
	12:Display()					
13:enter response						
	14(a):if okToAdd Register(secId, studId)	15(a): make RegnRecord object		16(a):Add student (RegnRecord)		17(a):Add section (RegnRecord)
	14(b): if okToAdd OR removeFromWa - itlist Remove(secId, studId)	15(b): RemoveWait (studId)		16b: student removed		
	EndLoop					

Additional Notes:

1, RegnRecord is the object that keeps info about each particular student in the section. This is an association class, but becomes a separate class because student-section is a many-many relationship. When we designed the operation "register student in section" we decided that RegnRecord would be created in the façade and sent to student and section

2. For both options "Ok to add" and "Remove from Waitlist" the student has to be removed from the Waitlist

3. Note that the waitlist gets modified as we are iterating through it. Hence, during implementation, section will return a copy of the waitlist.

4. All objects/classes are in bold in top row. Waitlist Iterator object is created is Step 7. Hence it is shown in the third row.