# Linked List Iterator Implementation test:

Add():

To test this class we will need to create an instance of a linked list, we will create a linked list of Objects Job to test the constructor. We then will test the add method by adding a job to an empty list we created in this tester class. It should become the only element of the list and return true if it is compared with get back and get first. We then add to a list with one or more elements and the add should be right after the nodepointer, in this case right after the element added previously, so it should be at the back of the list.

HasNext():

The hasNext method will return that there no null at the node after the nodepointer, we will test it by creating a list and adding elements to it, and then checking that hasNext() returns true. After that, we move to the end of the list and check that at then the method returns false.

Next():

To test the next method we will create a list of jobs and an iterator and make sure that the next returns the element the iterator it was pointing at, so the first next returns the first element and the second next returns the second element.

HasPrevious():

To test this we will need to create a list of jobs and check that the first node doesn’t have previous, and then we move using the next() method and check that now the node pointer has previous.

Previous():

To test this we create a list and we reach the end of it with the next() method to reach the last node and then we call the previous method to check it returns the last node and the second previous will return the second to last node.

Remove():

To test this method we make a list of jobs, and then we call the remove method and check that the new front of the list is the second element added to front.

# Job Class Test:

To test this class we first need to call the constructor and make an instance of a job.

getId: we will test that it returns the same value given to it by the constructor.

getEarliestStart: we will test that it returns the same value given to it by the constructor.

getDeadline: we will test that it returns the same value given to it by the constructor.

getDuration: we will test that it returns the same value given to it by the constructor.

getProfit: we will test that it returns the same value given to it by the constructor.

Equals(): this method compares two jobs based on their Id assuming that a job Id is unique. To test it we will create two jobs that have the same Id number and the equals method should return true, and then call the method on another job.

Compareto(): this method is supposed to order jobs by id number, we check when the compareto returns positive 0 and negative.

Comparators: to test the comparators of this class, we create a linked list of unorganized Jobs and we sort it using this comparator.

Comparebyearlieststart: we check that the first element is the earliest start.

ComparebyProfit: we check that the first element is the one with the most profit

# Compound Job

To test this class we need to create an organized array of jobs before we can create an instance of it, after that we input the array into the constructor and check that the id of the compound job is the id of the first job, and the earliest start is the start of the first subjob and that the dealine is the deadline of the last job. And that duration is the addition of all the duration.

Getsubjobs(): we test that it returns the array we input in the first place in the constructor.

# Schedule Slot:

To test the schedule slot, we need to create an instance of one using the constructor, and to do that we need to create a job. And then using the getter methods to check that the scheduleslot has all the information from the job. We check it is the same job input, the same start time, and that the setStartime method works, we then get the end time which should be the start time + the duration.

For the compareto method, we will compare the three possibilities of comparison, one that gives back negative one that is zero and finally one that yields positive.

Comparators: to test the comparators of this class, we create a linked list of unorganized Slots check what the first element is and we sort it using this comparator and check that the first element is now the one that starts the earliest.

# Schedule Class

To test the schedule class we test the constructor and add some schedule slots into it, we then test the noJobAt() method which contains a loop so we test when there is no element in the loop and then we test when there is only one and then with many, we also test when the element is first or mid or last.

# Schedule Metrics: Schedule as late as possible

To test this class, we need to create a new schedule and jobs that we can schedule as late as possible because it is a loop we will schedule jobs on an empty list and then will schedule it around one job and then we will schedule a job in a full schedule. We will then schedule at the end of the list, the middle and then the beginning because this loop starts from the end. We check that all of them are scheduled exactly next to their deadline.

We also try to schedule a compound job in that list, we try a possible compound job and one that will return false

# Schedule as Early as possible

Testing will be very similar: To test this class, we need to create a new schedule and jobs that we can schedule as early as possible because it is a loop we will schedule jobs on an empty list and then will schedule it around one job and then we will schedule a job in a full schedule. We will then schedule at the end of the list, the middle and then the beginning . We check that all of them are scheduled exactly at their earliest.

We also try to schedule a compound job in that list, we try a possible compound job and one that will return false

# JobScheduler:

To test this class we can test the job scheduler method with Junit but the main method will be tested without Junit.

scheduleJob(): to test this method we need to create a list of jobs and then add those jobs to a linked list of jobs, the jobs will be sorted and scheduled into a new schedule, the order will depend on the comparators we tested earlier. We will run the scheduleJob and check that the job with the most profit and whose deadline is earliest is cheduled first and the one with latest deadline is last.

MainMethod: to test this, we will first run the main method without an input and make sure it throws the error it is supposed to throw, with an error message

And then we run the method with the JobList provided and make sur that I prints two schedules in the correct format with a profit comparator between the two schedules and tells us which schedule yields more profit.