```
"""
File: linkedpriorityqueue.py
"""

from node import Node
from linkedqueue import LinkedQueue

class LinkedPriorityQueue(LinkedQueue):
    """A link-based priority queue implementation."""

    def __init__(self, sourceCollection = None):
        """Sets the initial state of self, which includes the
        contents of sourceCollection, if it's present."""
        LinkedQueue.__init__(self, sourceCollection)

    def add(self, newItem):
        """Inserts newItem after items of greater or equal
        priority or ahead of items of lesser priority.
        A has greater priority than B if A < B."""
        if self.isEmpty() or newItem >= self.rear.data:
            # New item goes at rear
            LinkedQueue.add(self, newItem)
        else:
            # Search for a position where it's less
            probe = self.front
            while newItem >= probe.data:
                trailer = probe
                probe = probe.next
            newNode = Node(newItem, probe)
            if probe == self.front:
                # New item goes at front
                self.front = newNode
            else:
                # New item goes between two nodes
                trailer.next = newNode
            self.size += 1
```

The time and space analysis for `LinkedPriorityQueue` is the same as that of `LinkedQueue`, with the exception of the `add` method. This method now must search for the proper place to insert an item. Rearranging the links once this place is found is a constant time operation, but the search itself is linear, so `add` is now O(*n*).

Exercise

Suggest a strategy for an array-based implementation of a priority queue. Will its space/time complexity be any different from the linked implementation? What are the trade-offs?

Case Study

An Emergency Room Scheduler

As anyone who has been to a busy hospital emergency room knows, people must wait for service. Although everyone might appear to be waiting in the same place, they are actually in separate groups and scheduled according to the seriousness of their condition. This case study develops a program that performs this scheduling with a priority queue.

Request

Write a program that allows a supervisor to schedule treatments for patients coming into a hospital's emergency room. Assume that, because some patients are in more critical condition than others, patients are not treated on a strictly first-come, first-served basis, but are assigned a priority when admitted. Patients with a high priority receive attention before those with a lower priority.

Analysis

Patients come into the emergency room in one of three conditions. In order of priority, the conditions are ranked as follows:

1. Critical
2. Serious
3. Fair

When the user selects the Schedule option, the program allows the user to enter a patient's name and condition, and the patient is placed in line for treatment according to the severity of his or her condition. When the user selects the Treat Next Patient option, the program removes and displays the patient first in line with the most serious condition. When the user selects the Treat All Patients option, the program removes and displays all patients in order from patient to serve first to patient to serve last.

Each command button produces an appropriate message in the output area. Table 8-7 lists the interface's responses to the commands.

Table Table 8-7

Commands of the Emergency Room Program

User Command	Program Response
Schedule	Prompts the user for the patient's name and condition, and then prints `<patient name> is added to the <condition> list`.
Treat Next Patient	Prints `<patient name> is being treated`.

User Command	Program Response
Treat All Patients	Prints `<patient name> is being treated.`

Here is an interaction with the terminal-based interface:

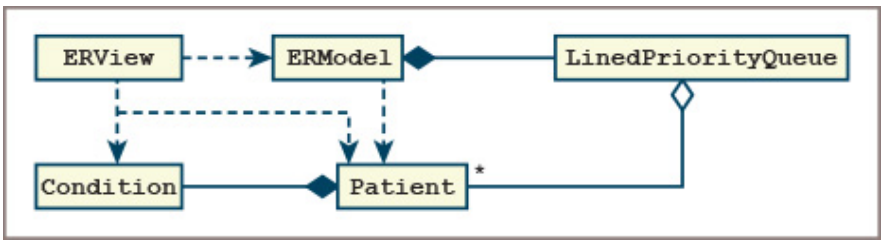
```
Main menu
1 Schedule a patient
2 Treat the next patient
3 Treat all patients
4 Exit the program
Enter a number [1-4]: 1
Enter the patient's name: Bill
Patient's condition:
1 Critical
2 Serious
3 Fair
Enter a number [1-3]: 1
Bill is added to the critical list.
Main menu
1 Schedule a patient
2 Treat the next patient
3 Treat all patients
4 Exit the program
Enter a number [1-4]: 3
Bill / critical is being treated.
Martin / serious is being treated.
Ken / fair is being treated.
No patients available to treat.
```

Classes

The application consists of a view class, called `ERView`, and a set of model classes. The view class interacts with the user and runs methods on the model. The class `ERModel` maintains a priority queue of patients. The class `Patient` represents patients, and the class `Condition` represents the three possible conditions. The relationships among the classes are shown in Figure 8-11.

Figure 8-11

The classes in the ER scheduling system



Design and Implementation

The `Patient` and `Condition` classes maintain a patient's name and condition. You can compare (according to their conditions) and view them as strings. Here is the code for these two classes:

```
class Condition(object):

    def __init__(self, rank):
        self.rank = rank

    def __ge__(self, other):
        """Used for comparisons."""
        return self.rank >= other.rank

    def __str__(self):
        if   self.rank == 1:   return "critical"
        elif self.rank == 2:   return "serious"
        else:                  return "fair"

class Patient(object):

    def __init__(self, name, condition):
        self.name = name
        self.condition = condition

    def __ge__(self, other):
        """Used for comparisons."""
        return self.condition >= other.condition

    def __str__(self):
        return self.name + " / " + str(self.condition)
```

The class `ERView` uses a typical menu-driven loop. You structure the code using several helper methods. Here is a complete listing:

```
"""
File: erapp.py

The view for an emergency room scheduler.
"""

from model import ERModel, Patient, Condition

class ERView(object):
    """The view class for the ER application."""

    def __init__(self, model):
        self.model = model

    def run(self):
        """Menu-driven command loop for the app."""
        menu = "Main menu\n" + \
            " 1 Schedule a patient\n" + \
            " 2 Treat the next patient\n" + \
            " 3 Treat all patients\n" \
            " 4 Exit the program\n"
        while True:
            command = self.getCommand(4, menu)
            if command == 1: self.schedule()
            elif command == 2: self.treatNext()
            elif command == 3: self.treatAll()
            else: break

    def treatNext(self):
        """Treats one patient if there is one."""
        if self.model.isEmpty():
            print("No patients available to treat.")
        else:
            patient = self.model.treatNext()
            print(patient, "is being treated.")

    def treatAll(self):
        """Treats all the remaining patients."""
        if self.model.isEmpty():
            print("No patients available to treat.")
        else:
            while not self.model.isEmpty():
                self.treatNext()

    def schedule(self):
        """Obtains patient info and schedules patient."""
        name = input("\nEnter the patient's name: ")
        condition = self.getCondition()
        self.model.schedule(Patient(name, condition))
        print(name, "is added to the", condition, "list\n")

    def getCondition(self):
        """Obtains condition info."""
        menu = "Patient's condition:\n" + \
            " 1 Critical\n" + \
            " 2 Serious\n" + \
            " 3 Fair\n"
```

```
            number = self.getCommand(3, menu)
            return Condition(number)

        def getCommand(self, high, menu):
            """Obtains and returns a command number."""
            prompt = "Enter a number [1-" + str(high) + "]: "
            commandRange = list(map(str, range(1, high + 1)))
            error = "Error, number must be 1 to " + str(high)
            while True:
                print(menu)
                command = input(prompt)
                if command in commandRange:
                    return int(command)
                else:
                    print(error)

# Main function to start up the application
def main():
    model = ERModel()
    view = ERView(model)
    view.run()

if __name__ == "__main__":
    main()
```

The class `ERModel` uses a priority queue to schedule the patients. Its implementation is left as a programming project for you.