Module 10 Homework: Heaps of Books

George R. R. Martin has finaly finished *The Winds of Winter*, the next book in his hit series *A Song of Ice and Fire*. Demand is high, and you have been hired to design a program to track and serve orders.

In accordance with the CRC (Center for Reading Control) guidelines, copies of the book should be distributed not according to a "first come, first served" basis, but according to the following tiers:

- Phase 1
    - `FR` - First Readers: those who have been reading the books the longest get dibs
    - `EL` - Elderly: these folks have the least time remaining, and need to get this book the fast
- Phase 2
    - `K12` - Students can learn from the nuanced politics in the book to help create a better world
    - `Uni` - Students in university have had a hard year! They deserve a distraction
- Phase 3
    - `GP` - The general public can buy books once higher priority groups have had their chance

Books should be distributed according to phase (Phase 1, then Phase 2, then Phase 3) with ties broken by categories in the top-to-bottom order shown (an `FR` should be served before an `EL`, and a `K12` before a `Uni`). More phases may be added in the future, so your program **must** sort by phase, then category.

class: `Person`

Write a class to keep track of people. A `Person` is initialized with the following attributes:

- `name` - a **string** representing the name of the person buying the book
- `phase` - an **int** representing the phase in which this person fits
- `cat` - an optional **string** representing the category (`FR`, `EL`, `K12`, ...) of the person

You should be able to compare people with the standard python operators. When comparing people, a person who should be served first should compare as less than a person who should be served later.

```
>>> Person('Taylor Swift', 1, 'FR') < Person('Jake', 3, 'GP')
True
```

If a category is not specified, treat that `Person` as "greater than" (should be served after) the others in their phase.

class: `Line`

Book sellers will use this class to keep track of who should be served. In order to optimize turnaround time, this class should support the following ADT with the noted running times:

Magic Methods

- `init(n_avail, people)`
  - *O*(n)
  - initializes a new `Line` with a number of books available to sell `n_avail`
  - `people` is an optional collection of `Person` objects
- `len`
  - *O*(1)
  - returns the number of people in the `Line`

Non-magic Methods

- `add_buyer(person)`
  - *O*(logn)
  - adds a `Person` object to the `Line`
  - raise a `TypeError` if the argument passed is not a `Person` object
  - raise a `KeyError` if `person` is already in line or has already been sold a book
- `sell_book`
  - *O*(logn)
  - removes one copy of *The Winds of Winter*
  - removes **and returns** the next person in line to whom a book should be sold
  - raise a `ValueError` if no copies are available
  - raise an `IndexError` if the `Line` is empty
- `add_copies(n)`
  - *O*(1)
  - increases the number of books available to sell by `n`

Special Cases

- Due to high demand, only one copy of the book is available per person. Note the behavior in `add_buyer`: raise a `KeyError` if the `person` specified is already in line or has already been sold a book. This special behavior should not change the asymptotic running times of any methods.
- Assume that if two `Person` objects have the same `name`, they are the same person.
- **Your code must support an arbitrary number of phases**, not just 3. Your employer wants to re-use this code in the future with potentially more phases. You can assume phases 4+ do not have multiple categories.

Examples

Any examples below are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own tests, and think carefully about edge cases.

Initializing

```
>>> from TWOW import *
>>> p1 = Person('Jake', 3, 'GP')
>>> p2 = Person('Taylor Swift', 1, 'FR')
>>> p3 = Person('Noam Chomsky', 1, 'EL')
>>> people = {p1, p2, p3}
>>> line = Line(n_avail=2, people=people)
>>> len(line)
3
```

Selling

```
>>> line.sell_book()
Person(Taylor Swift, 1, FR)
>>> line.sell_book()
Person(Noam Chomsky, 1, EL)
>>> line.sell_book()
Traceback (most recent call last):
...
ValueError: No more books to sell!
>>> line.add_copies(5)
>>> line.sell_book()
Person(Jake, 3, GP)
```

Adding buyers

```
>>> line.add_buyer(p1)
Traceback (most recent call last):
...
KeyError: 'Jake already served'
>>> p4 = Person('Greninja', 2, 'K12')
>>> line.add_buyer(p4)
>>> line.sell_book()
Person(Greninja, 2, K12)
>>>
```

External Modules

Do not use any imported modules (`math`, `collections`, ...) when implementing functionality. It is okay to use imported modules for testing.

It is okay to import modules you write yourself (e.g. any data structures you write yourself).

Submitting

At a minimum, submit the following file:

- `TWOW.py`

Students must submit to Mimir individually by the due date (typically, the first Wednesday after this module opens at 11:59 pm) to receive credit.

Grading

- 30 - `Person`
    - 10 - `init`
    - 10 - `eq`
    - 10 - `lt`
- 70 - `Line` (These test cases largely depend on each other, so a function-by-function breakdown is not available)
    - `init`
    - `len`
    - `add_buyer`
    - `sell_book`
    - `add_copies`

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on several assignments