

Data Structures & Algorithms - Lessons Learned Reflections

Micah J. Courey

CSC400 – Data Structures and Algorithms

Colorado State University – Global Campus

Russell Frith

May 7th, 2017

Data Structures & Algorithms - Lessons Learned Reflections

In order to effectively and efficiently design computer programs it's essential to learn and implement data structures and algorithms. Abstract data types (ADT) are models of different techniques for organizing data. A data structure is an implementation of an abstract data type in a specific programming language. An Algorithms is a set of steps used to solve a particular problem. Data structures and algorithms allow software developers to design programs without the need to reinvent the wheel every time they need to store and organize data.

The Bag ADT

A bag is a abstract data type that can store multiple different objects as well as duplicate objects in no particular order and objects can be added to the bag until the bag reaches it's maximum capacity. The benefits of using a bag in Java would vary depending on what the programmer is trying to accomplish with their program. If the order of the objects does not matter than a programmer would benefit from using a bag ADT. If order does matter than they would want to use a stack or queue instead of a bag. A possible implementation would be if you wanted to create a raffle program you could insert several objects which could be the players names into the bag. Since an object can be removed from a bag in no particular order this means that the object that is removed would be a random object. It would work just like in a real life raffle game where you reach into a bag of filled with raffle tickets and pull a ticket out of the bag.

Big-Oh Notation

Big-O notation is used to determine how an algorithm scales as the dataset increases. Some algorithms take a constant time to execute, while other algorithm's execution time increases as the data set increases. An algorithm's time complexity is one of the most important factors in determining the most efficient algorithm for the task at hand. The other important factor is space complexity which measures an algorithm's memory usage.

The benefit of Big-O is that it is used to determine the worst case running time of an algorithm. An algorithm will often run much above the worst case. For instance when performing a binary search it is rarely going to find its target value at the last index. Worst case runtime is important because it's useful to know the slowest case so you can determine if this will be too slow for the system your algorithm will run on. While Big O is an important measurement it's also useful to consider other factors such as the average case. Big Theta can be useful for looking at the average case of an algorithm. Using Big O, Big Theta and Big Omega is a very useful way to determine which algorithm is the best option for the particular problem that you're try to solve based on the size of your particular dataset.

The Stack ADT

A stack is a data structure that stores and retrieves objects in a last in first out order which is commonly abbreviated as LIFO. Last in first out means that the last object inserted into the stack will be the first object to be removed from the stack. Operations can be performed on the stack including push and pop. Push adds an object to top of the stack while the pop operation removes an item from the top of the stack. One example of how you could use a stack is if you want to reverse the order of a list of objects. You could use a stack because all the items that you enter into the stack would be removed in a LIFO manner which in practice would reverse the ordering of the objects.

Recursive Methods

There are a few basic components required to create a recursive method. First in order for a method to be considered recursive, the method must call itself from within the method's body. The next important component to a recursive method is a base case. Without a base case the method will not know when to stop and thus will continue looping indefinitely. The base case is the condition that will terminate the recursive process which will end the method calls and return a final value.

The easiest way to implement a base case is through the use of a conditional statement such as if/else. In an if/else conditional statement the base case would go in the if statement and the recursion process in which the method calls itself would go in the else statement. This will cause the program to continue iterating through itself until it reaches the base case and it will then return the final value.

Sorting Algorithms

Sorting refers to the process of rearranging the order of objects based on specified characteristics. There are many different sorting algorithms and some of them sort faster than others especially when sorting large datasets. Insertion Sort and Selection Sort have a Big-Oh of $O(n^2)$. Mergesort and Heapsort have a faster Big-Oh time complexity of $O(n \log(n))$. Quicksort is a fast sorting method with an average case of $O(n \log(n))$ but it's worst case is $O(n^2)$. Although quicksort's worst case is $O(n^2)$, it doesn't reach it's worst case often so it's still considered a fast sorting algorithm.

Queues, Deques and Priority Queues

queues are a very useful data structure for storing and organizing entries. A standard queue has a first-in, first-out insertion and removal behavior similar to most real world waiting lines. deque on the other hand allows removal of items from either end so in practice it can behave as a queue or a stack. Lastly a priority queue does not provide removal from a specific location, instead the priority queue releases items in sorted order.

Conclusions and Final Reflections

Knowing and properly implementing data structures and algorithms is an important step to becoming an effective software developer. They allow the developer to implement a clear and efficient solution to a programming problem. Knowing the common data structures and algorithms also allows the developer to easily transition to several different programming languages with only the need to learn syntax differences between languages.

References

- Carrano, F. M., Henry, T., & Tahliliani, M. P. (2016). Data Structures and Abstractions with Java (4th ed.). Harlow, Essex: Pearson Education Limited.
- Budd, T. A. (2006). An Active Learning approach to Data Structures using C. Retrieved March 15, 2017, from http://web.engr.oregonstate.edu/~sinisa/courses/OSU/CS261/CS261_Textbook/Chapter08.pdf
- Khan Academy. (2017). Khan Academy. Retrieved 30 March 2017, from <https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/asymptotic-notation>
- The Idiot's Guide to Big O - DZone Java. (2017). DZone. Retrieved 30 March 2017, from <https://dzone.com/articles/idiots-guide-big-o>

Micah Courey

CSC400: Data Structures and Algorithms

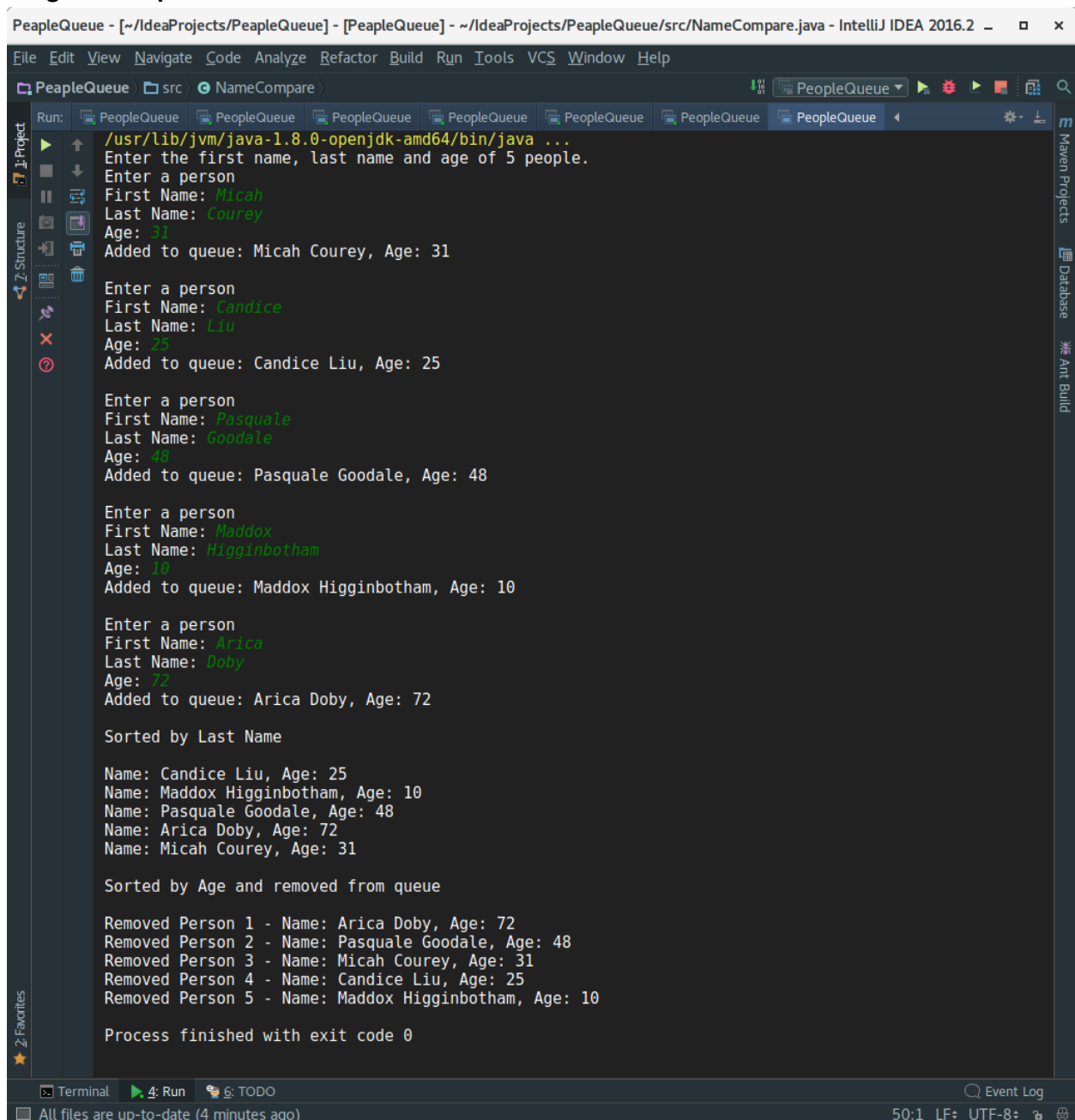
Portfolio Project - Final Program

Option #1

May 7th, 2017

Program Screenshots

Program Output



```
PeopleQueue - [~/IdeaProjects/PeopleQueue] - [PeopleQueue] - ~/IdeaProjects/PeopleQueue/src/NameCompare.java - IntelliJ IDEA 2016.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
PeopleQueue src NameCompare
Run: PeopleQueue PeopleQueue PeopleQueue PeopleQueue PeopleQueue PeopleQueue PeopleQueue PeopleQueue
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Enter the first name, last name and age of 5 people.
Enter a person
First Name: Micah
Last Name: Courey
Age: 31
Added to queue: Micah Courey, Age: 31

Enter a person
First Name: Candice
Last Name: Liu
Age: 25
Added to queue: Candice Liu, Age: 25

Enter a person
First Name: Pasquale
Last Name: Goodale
Age: 48
Added to queue: Pasquale Goodale, Age: 48

Enter a person
First Name: Maddox
Last Name: Higginbotham
Age: 10
Added to queue: Maddox Higginbotham, Age: 10

Enter a person
First Name: Arica
Last Name: Doby
Age: 72
Added to queue: Arica Doby, Age: 72

Sorted by Last Name

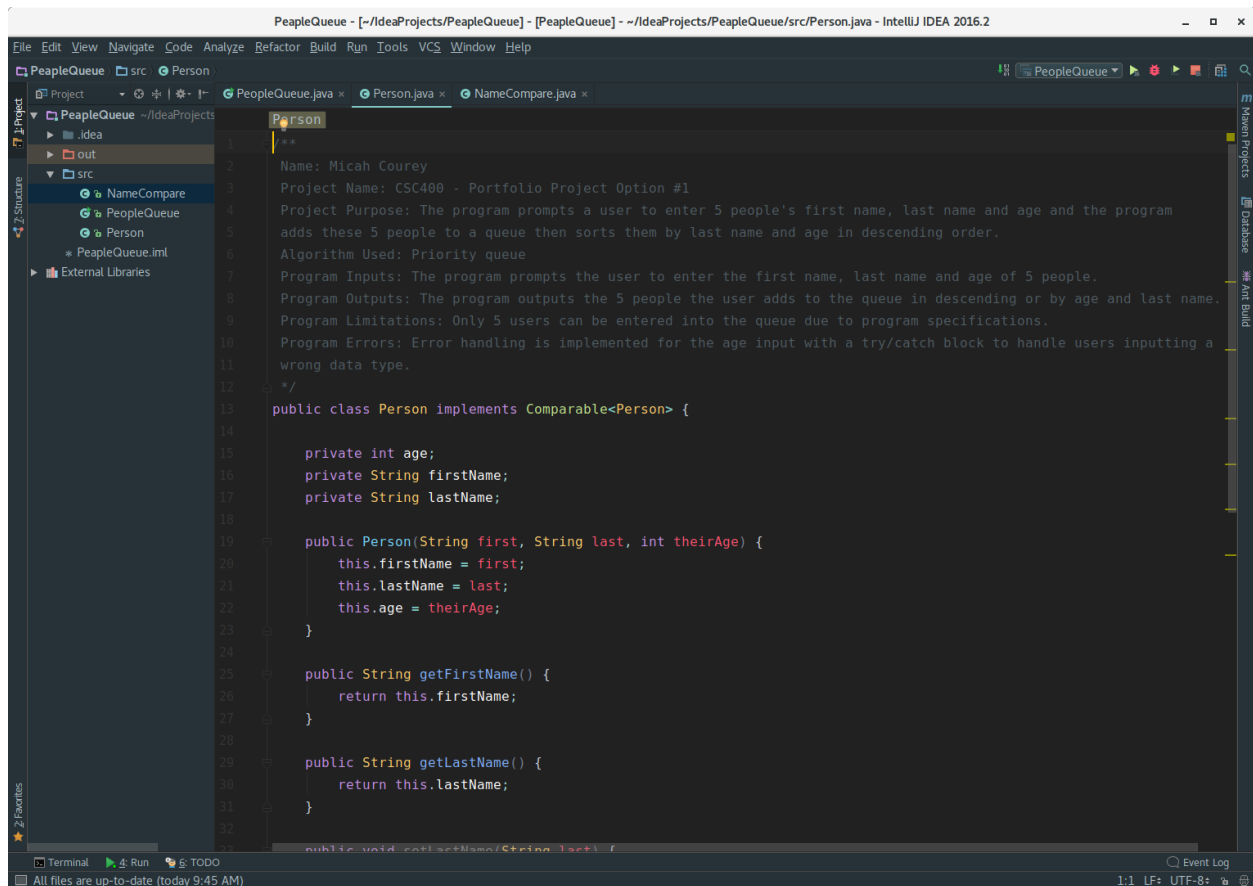
Name: Candice Liu, Age: 25
Name: Maddox Higginbotham, Age: 10
Name: Pasquale Goodale, Age: 48
Name: Arica Doby, Age: 72
Name: Micah Courey, Age: 31

Sorted by Age and removed from queue

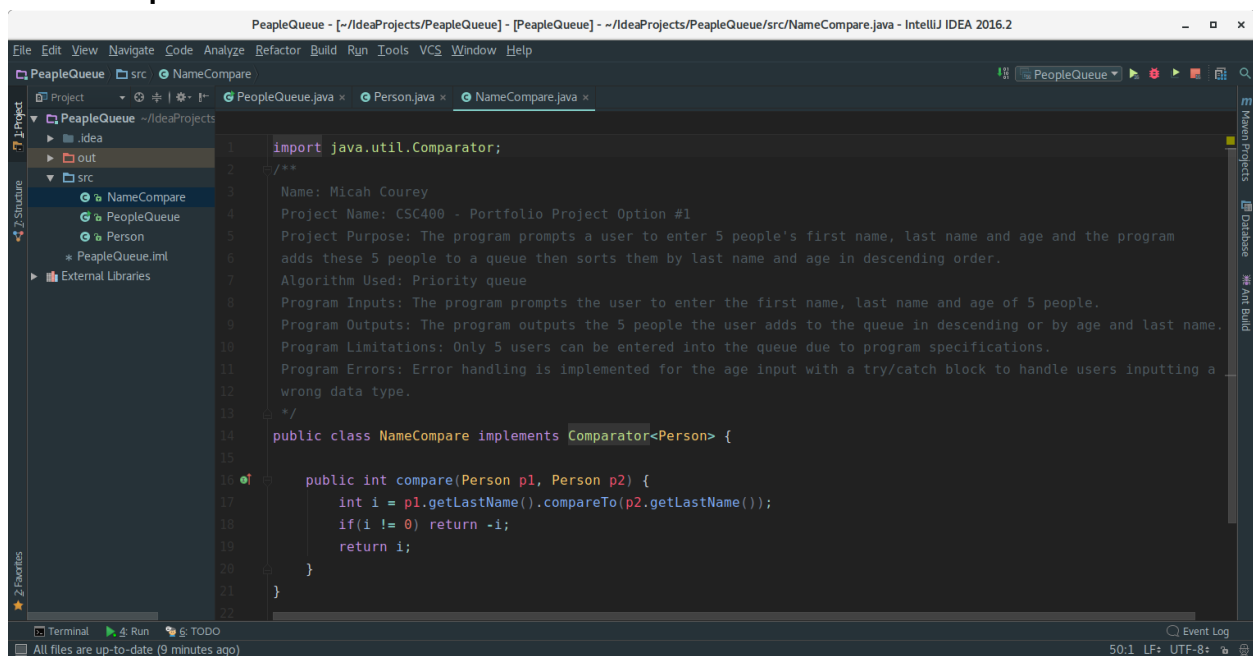
Removed Person 1 - Name: Arica Doby, Age: 72
Removed Person 2 - Name: Pasquale Goodale, Age: 48
Removed Person 3 - Name: Micah Courey, Age: 31
Removed Person 4 - Name: Candice Liu, Age: 25
Removed Person 5 - Name: Maddox Higginbotham, Age: 10

Process finished with exit code 0
Terminal 4: Run 6: TODO Event Log
All files are up-to-date (4 minutes ago) 50:1 LF UTF-8
```

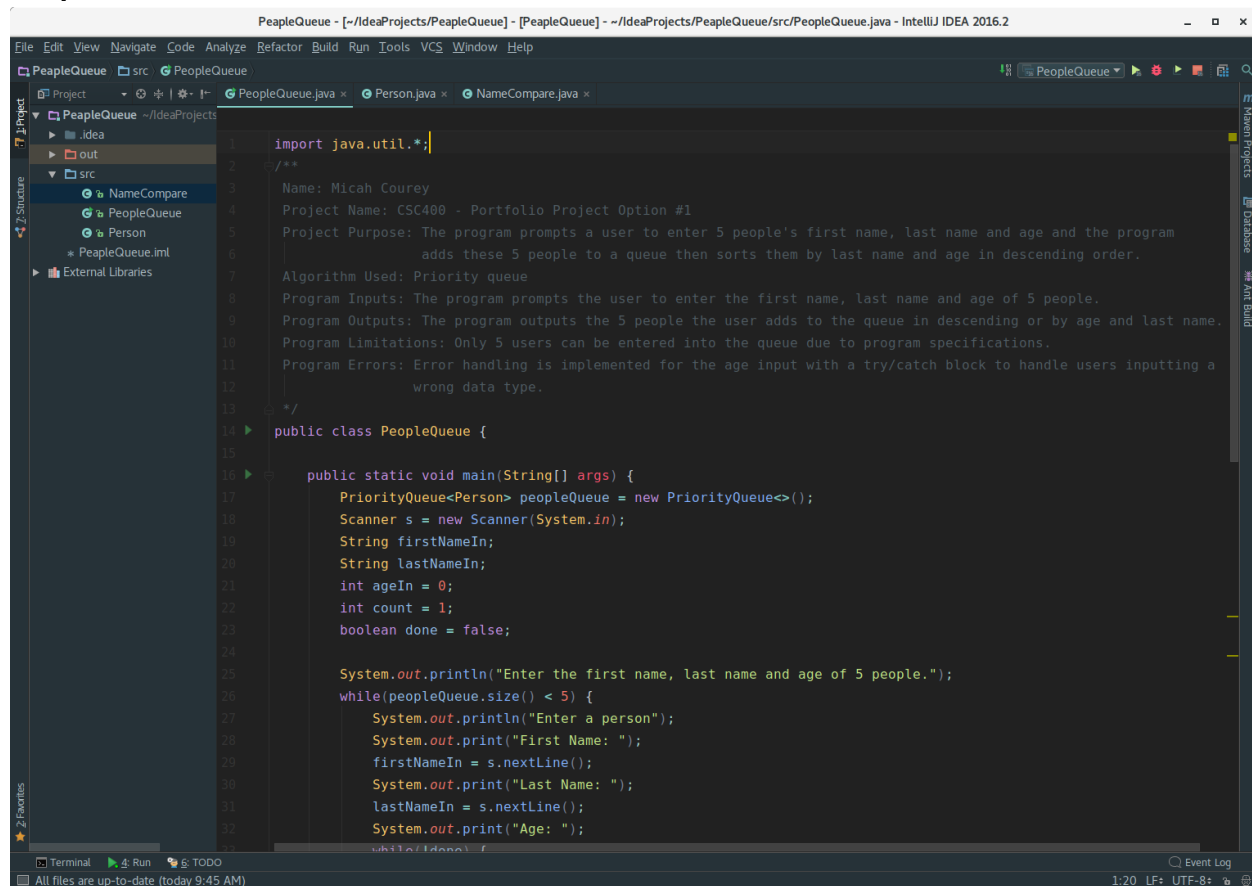
Person Class



NameCompare Class



PeopleQueue Class



Program Source Code

PeopleQueue.java

```

import java.util.*;
/**
Name: Micah Courey
Project Name: CSC400 - Portfolio Project Option #1
Project Purpose: The program prompts a user to enter 5 people's first name, last name and age and the
program adds these 5 people to a queue then sorts them by last name and age in descending order.
Algorithm Used: Priority queue
Program Inputs: The program prompts the user to enter the first name, last name and age of 5 people.
Program Outputs: The program outputs the 5 people the user adds to the queue in descending or by age and
last name.
Program Limitations: Only 5 users can be entered into the queue due to program specifications.
Program Errors: Error handling is implemented for the age input with a try/catch block to handle users
inputting a wrong data type.
*/
public class PeopleQueue {

```

```

    public static void main(String[] args) {
        PriorityQueue<Person> peopleQueue = new PriorityQueue<>();
        Scanner s = new Scanner(System.in);
        String firstNameIn;
        String lastNameIn;
        int ageIn = 0;

```



```

int count = 1;
boolean done = false;

System.out.println("Enter the first name, last name and age of 5 people.");
while(peopleQueue.size() < 5) {
    System.out.println("Enter a person");
    System.out.print("First Name: ");
    firstNameIn = s.nextLine();
    System.out.print("Last Name: ");
    lastNameIn = s.nextLine();
    System.out.print("Age: ");
    while(!done) {
        try {
            ageIn = Integer.parseInt(s.nextLine());
            done = true;
        } catch (NumberFormatException e) {
            System.out.println("Error: Please enter a number for age");
            System.out.print("Age: ");
        }
    }
    Person person = new Person(firstNameIn, lastNameIn, ageIn);
    peopleQueue.add(person);
    System.out.println("Added to queue: " + firstNameIn + " " + lastNameIn + ", Age: " + ageIn + "\n");
    done = false;
}

NameCompare nameCompare = new NameCompare();
List<Person> people = new ArrayList(peopleQueue);
Collections.sort(people, nameCompare);

System.out.println("Sorted by Last Name\n");
for (int i = 0; i < people.size(); i++) {
    System.out.println("Name: " + people.get(i).getFirstName() + " "
        + people.get(i).getLastName() + ", Age: "
        + people.get(i).getAge());
}

System.out.println("\nSorted by Age and removed from queue\n");
while (!peopleQueue.isEmpty()) {
    System.out.println("Removed Person " + count + " -" + peopleQueue.remove());
    count++;
}
}
}

```

Person.java

```
/**
```

Name: Micah Courey

Project Name: CSC400 - Portfolio Project Option #1

Project Purpose: The program prompts a user to enter 5 people's first name, last name and age and the program adds these 5 people to a queue then sorts them by last name and age in descending order.

Algorithm Used: Priority queue

Program Inputs: The program prompts the user to enter the first name, last name and age of 5 people.

Program Outputs: The program outputs the 5 people the user adds to the queue in descending or by age and last name.

Program Limitations: Only 5 users can be entered into the queue due to program specifications.

Program Errors: Error handling is implemented for the age input with a try/catch block to handle users inputting a wrong data type.

```
*/
```

```
public class Person implements Comparable<Person> {
```

```
    private int age;
```

```

private String firstName;
private String lastName;

public Person(String first, String last, int theirAge) {
    this.firstName = first;
    this.lastName = last;
    this.age = theirAge;
}

public String getFirstName() {
    return this.firstName;
}

public String getLastName() {
    return this.lastName;
}

public void setLastName(String last) {
    this.lastName = last;
}

public int getAge() {
    return this.age;
}

@Override
public int compareTo(Person person) {
    if(this.equals(person))
        return 0;
    else if(getAge() < person.getAge())
        return 1;
    else
        return -1;
}

public String toString() {
    return "Name: " + getFirstName() + " " + getLastName() + ", Age: " + getAge();
}
}

```

NameCompare.java

```

import java.util.Comparator;
/**

```

Name: Micah Courey

Project Name: CSC400 - Portfolio Project Option #1

Project Purpose: The program prompts a user to enter 5 people's first name, last name and age and the program adds these 5 people to a queue then sorts them by last name and age in descending order.

Algorithm Used: Priority queue

Program Inputs: The program prompts the user to enter the first name, last name and age of 5 people.

Program Outputs: The program outputs the 5 people the user adds to the queue in descending or by age and last name.

Program Limitations: Only 5 users can be entered into the queue due to program specifications.

Program Errors: Error handling is implemented for the age input with a try/catch block to handle users inputting a wrong data type.

```

*/

```

```

public class NameCompare implements Comparator<Person> {

    public int compare(Person p1, Person p2) {
        int i = p1.getLastName().compareTo(p2.getLastName());
        if(i != 0) return -i;
        return i;
    }
}

```