**Joshua Ramos**

**Fernando Aguilar**

**Gender Mag 2**

**Student 1**

**1.Write two small paragraphs: one about GenderMag and the other about Abi (her assets,**

**how Abi is different from Pat and Tim, etc). Please be precise with your writing.**

**Create a custom persona of Abi based on the task she needs to perform:**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

Gendermag is research designed to find and fix so-called "gendermag bugs". Created based on the personal thoughts of the designer or developer. Some find the generated application harder to use and understand than others, while others find it very intuitive and easy to use, depending on their particular personality type and learning style. The "user interface" used by the user is explicitly mentioned here. There are three basic persona types for her in this situation: Abi, Pat, and Tim.

Abi should be able to understand his teammates' code. This can be difficult if the code is very different from what she is used to. If the code is well written and documented, Abi will be very easy to maintain. She is very methodical, trying to get a clear picture of how things work. So she can understand and maintain well-written and well-documented code. If the code is more complex or not well written or documented, she may have more trouble understanding it.

**2. Create a .txt file with 100 rows of various movies data of your choice in the following**

**table format:**

**Movie ID number of viewers rating release year Movie name**

**0000012211 174 8.4 2017 “Star Wars: The Last Jedi”**

**0000122110 369 7.9 2017 “Thor: Ragnarok”**

Graphical user interface, table

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Table

Description automatically generated

****

**ID Number of views Rating Year Title**

0000000001 10002 8.5 2022 "All Quiet on the Western Front"

0000000002 65532345 8.0 1965 "The Sound of Music"

0000000003 345684 5.0 2016 "Ben-Hur2"

0000000004 97654612 8.8 1959 "Ben-Hur1"

0000000005 5430021 7.5 2017 "Dunkirk"

0000000006 1200327 8.9 2019 "1917"

0000000007 9000324 7.9 2015 "Mad Max: Fury Road"

0000000008 6043212 6.2 1985 "Mad Max: Beyond Thunderdome"

0000000009 4003213 9.0 1981 "Mad Max 2"

0000000010 9003238 9.0 1979 "Mad Max"

0000000011 15003235 6.9 1979 "Star Trek: The Motion Picture"

0000000012 15024567 9.2 1982 "Star Trek II: The Wrath of Khan"

0000000013 15234568 8.2 1984 "Star Trek III: The Search for Spock"

0000000014 20345671 8.9 1986 "Star Trek IV: The Voyage Home"

0000000015 10000078 6.0 1989 "Star Trek V: The Final Frontier"

0000000016 13045783 9.0 1991 "Star Trek VI: The Undiscovered Country"

0000000017 10003246 6.2 1994 "Star Trek: Generations"

0000000018 17643456 8.8 1996 "Star Trek: First Contact"

0000000019 10003211 6.0 1998 "Star Trek: Insurrection"

0000000020 5003212 6.0 2002 "Star Trek: Nemesis"

0000000021 16534897 7.5 2009 "Star Trek"

0000000022 19023456 7.7 2013 "Star Trek Into Darkness"

0000000023 20003432 7.0 2016 "Star Trek Beyond"

0000000024 60434521 5.2 1999 "Star Wars: Episode I: The Phantom Menace"

0000000025 50000003 9.3 1979 "Alien"

0000000026 45325790 7.3 1986 "Aliens"

0000000027 35679532 6.2 1993 "Alien 3"

0000000028 23694021 5.9 1997 "Alien: Resurrection"

0000000029 35678491 6.3 2004 "Alien Vs. Predator"

0000000030 3556670 4.9 2007 "Aliens Vs. Predator: Requiem"

0000000031 48976046 6.6 2012 "Prometheus"

0000000032 30095864 6.6 2017 "Alien: Covenant"

0000000033 40596223 6.2 2001 "Planet of the Apes"

0000000034 94689932 9.0 1968 "Planet of the Apes"

0000000035 54823456 8.7 2001 "Shrek"

0000000036 96428351 9.0 2004 "Shrek 2"

0000000037 70496743 7.0 2007 "Shrek the Third"

0000000038 90475721 7.7 2010 "Shrek Forever After"

0000000039 50000062 7.5 2011 "Puss in Boots"

0000000040 100002342 9.0 1978 "Superman"

0000000041 100023411 9.4 1980 "Superman II"

0000000042 92345676 6.9 2006 "Superman Returns"

0000000043 95664312 9.9 2015 "The Martian"

0000000044 4096853 6.1 2005 "Doom"

0000000045 6784936 9.7 1982 "First Blood"

0000000046 67849024 6.9 1985 "Rambo: First Blood Part II"

0000000047 79492045 8.8 1988 "Rambo III"

0000000048 3172842 8.2 1989 "UHF"

0000000049 58392567 8.9 2008 "Rambo"

0000000050 98764023 9.0 2019 "Rambo: Last Blood"

0000000051 15356565 8.5 1942 "Casablanca";

0000000052 8974785 7.2 2009 "Night at the Museum: Battle of the Smithsonian"

0000000053 7395834 7.2 2014 "Night at the Museum: Secret of the Tomb"

0000000054 9909472 9.0 2018 "A Quiet Place"

0000000055 10838421 9.0 2020 "A Quiet Place Part II"

0000000056 20984772 8.6 1986 "The Fly"

0000000057 19757382 6.7 1958 "The Fly"

0000000058 1098374 6.2 1989 "The Fly II"

0000000059 19384721 8.0 1985 "Out of Africa"

0000000060 20937471 8.0 2018 "Black Panther"

0000000061 20937471 8.0 2008 "Iron Man"

0000000062 20394753 7.2 2010 "Iron Man II"

0000000063 300058323 9.8 2018 "Logan"

0000000064 539284752 2.7 2009 "Transformers: Revenge of the Fallen"

0000000065 300000214 9.4 2018 "Deadpool II"

0000000066 323857393 9.5 2016 "Deadpool"

0000000067 38294212 9.2 1988 "The Dead Pool"

0000000068 59687943 7.0 1996 "Mission: Impossible"

0000000069 100002341 6.0 2000 "Mission: Impossible II"

0000000070 348572123 6.7 1986 "Top Gun"

0000000071 907482812 9.9 2022 "Top Gun: Maverick"

0000000072 39237421 8.4 1993 "Dragon: The Bruce Lee Story"

0000000073 50693845 8.3 1994 "The Crow"

0000000074 984756212 8.5 2005 "King Kong"

0000000075 10937421 1.1 2015 "50 Shades of Grey"

0000000076 29473458 5.3 2001 "Pearl Harbor"

0000000077 48392712 7.7 2019 "Midway"

0000000078 78504021 9.0 1933 "King Kong"

0000000079 391234 9.3 2016 "The Exception"

0000000080 1230482 9.1 2004 "Downfall (Der Untergang)"

0000000081 5093721 9.7 1981 "Das Boot"

0000000082 989684 8.4 2019 "Parasite"

0000000083 9846211 8.5 2010 "Inception"

0000000084 8763621 9.3 2003 "The Last Samurai"

0000000085 9385621 6.8 2004 "Meet the Fockers"

0000000086 9758692 7.4 2004 "Troy"

0000000087 23948312 7.9 2007 "300"

0000000088 27347582 9.6 2006 "Casino Royale"

0000000089; 40037337; 7.5 2016 "BeautyandBeast"

0000000090; 25023422 8.1 2015 "Chicago";2015

0000000091 15899999 6.2 2017 "Downsizing"

0000000092 62777777 8.8 2010 "Inception

0000000093 88833333 8.7 1990 "Goodfellas"

0000000094 64544444 6.8 2017 "Ferdinand"

0000000095 35353535 6.2 2017 "Downsizing

0000000096; 23333333 8.7 1999 "TheMatrix

0000000097; 28633333 6.7 2002 "ResidentEvil"

0000000098; 35344444 6.8 2017 "TheFateoftheFurious"

0000000099; 38855555 7.9 2002 "TheBourneIdentity"

0000000100; 21144444 7.8 2009 "Avatar"

3. **Create a class Movie which instantiates variables corresponds to the table columns.**

**Implement getters, setters, constructors and toString. Test your class.**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**4.** I**mplement 2 readData() methods : the first one will return an array of Movies and**

**the second one will return a List of movies . Test your methods.**

Graphical user interface, text, application, email

Description automatically generated

**5. Implement two search methods to search Movies by their rating from unsorted array**

**and by year (first) and a rating (second) from unsorted List. Test your methods and explain their time complexity.**

Graphical user interface, text, application, email

Description automatically generated

**6** **Implement two MovieComparators to compare movies by their rating from unsorted**

**array and by year (first) and a rating (second) from unsorted List. Test your methods**

**and explain their time complexity.**

Graphical user interface, text, application, email

Description automatically generated

**7. Github Link:**

**Student 2**

1.

Gendermag is a study that was developed to discover and correct what are referred to as "Gendermag bugs," which are instances where something is built based on the designer or developer's personal thoughts. Some people may find the generated application to be difficult to use and harder to grasp than others, while others may find it to be highly intuitive and easy to use, depending on particular personality types and learning styles. The "user interface" that users will employ is expressly mentioned here. There are three basic persona types in this situation: Abi, Pat, and Tim.

Abi would need to be able to comprehend the teammate's code. This could be challenging if the code is much different from what she is used to. The code would be considerably simpler for Abi to maintain if it were well-written and documented. She is quite methodical in her thoughts and tries to clearly grasp how things operate. She would therefore be able to comprehend and maintain well-written, well-documented code. She might have more trouble comprehending the code if it were more complicated or if it wasn't well-written or well-documented.

2. **Create your own generic interface MyQueue with methods addMovie(), clearAll(),**

**isEmpty(), peekMovie(), pushMovie(), removeMovie(), size() that will accept any type of**

**elements including Movies**

**Code:**

package GenderMag;

import java.io.\*;

import java.util.\*;

interface MyQueue{

void addMovie();

void clearAll();

void isEmpty();

void peekMovie();

void pushMovie();

void removeMovie();

void size();

}

public class MyHeap implements MyQueue {

public static void main(String[] args) throws Exception {

}

private static void HeapSort(){

String[] arr = {};

}

@Override

public void addMovie() {

// TODO Auto-generated method stub

}

@Override

public void clearAll() {

// TODO Auto-generated method stub

}

@Override

public void isEmpty() {

// TODO Auto-generated method stub

}

@Override

public void peekMovie() {

// TODO Auto-generated method stub

}

@Override

public void pushMovie() {

// TODO Auto-generated method stub

}

@Override

public void removeMovie() {

// TODO Auto-generated method stub

}

@Override

public void size() {

// TODO Auto-generated method stub

}

}

**3. Create your own generic MyHeap class that will implement your MyQueue interface.**

public class MyHeap implements MyQueue {

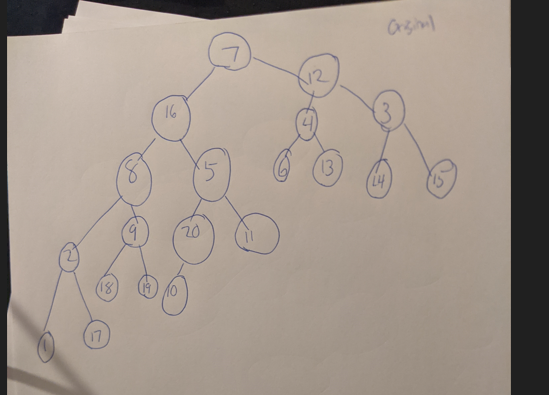
public static void main(String[] args) throws Exception {

}

**4. Draw a max Heap consisting of first 20 movies in the order they are stored in the file.**

1. 0000000001 10002 8.5 2022 "All Quiet on the Western Front"
2. 0000000002 65532345 8.0 1965 "The Sound ofMusic"
3. 0000000003 345684 5.0 2016 "Ben-Hur2"
4. 0000000004 97654612 8.8 1959 "Ben-Hur1"
5. 0000000005 5430021 7.5 2017 "Dunkirk"
6. 0000000006 1200327 8.9 2019 "1917"
7. 0000000007 9000324 7.9 2015 "Mad Max: Fury Road"
8. 0000000008 6043212 6.2 1985 "Mad Max: Beyond Thunderdome"
9. 0000000009 4003213 9.0 1981 "Mad Max 2"
10. 0000000010 9003238 9.0 1979 "Mad Max"
11. 0000000011 15003235 6.9 1979 "Star Trek: The Motion Picture"
12. 0000000012 15024567 9.2 1982 "Star Trek II: The Wrath of Khan"
13. 0000000013 15234568 8.2 1984 "Star Trek III: The Search for Spock"
14. 0000000014 20345671 8.9 1986 "Star Trek IV: The Voyage Home"
15. 0000000015 10000078 6.0 1989 "Star Trek V: The Final Frontier"
16. 0000000016 13045783 9.0 1991 "Star Trek VI: The Undiscovered Country"
17. 0000000017 10003246 6.2 1994 "Star Trek: Generations"
18. 0000000018 17643456 8.8 1996 "Star Trek: First Contact"
19. 0000000019 10003211 6.0 1998 "Star Trek: Insurrection"
20. 0000000020 5003212 6.0 2002 "Star Trek: Nemesis"

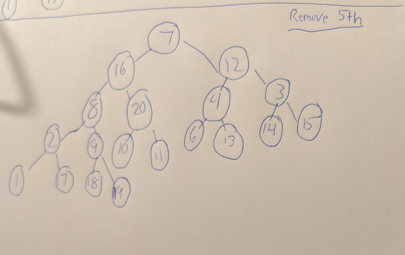
Original:



5. **Update your drawing after removing 5th, 10th, and 15th movie one by one from the**

**heap (include all three removal results).**

Remove 5th:

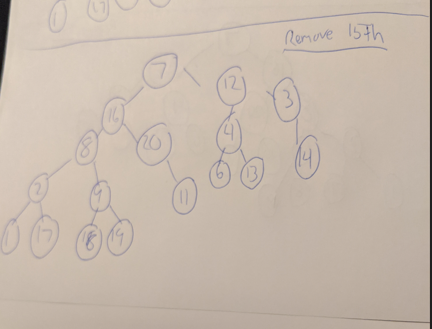


Remove 10th:

A white paper with writing on it

Description automatically generated with low confidence

Remove 15th:



**6. Implement HeapSort, sort your movie data using it and explain its time complexity.**

**package** GenderMag;

**public** **class** MyHeap

{

// Used for index in heap

**static** **int** *x* = -1;

// Predefining the heap array

**static** String []*heap* = **new** String[1000];

// Defining formation of the heap

**static** **void** heapForm(String k)

{

*x*++;

*heap*[*x*] = k;

**int** child = *x*;

String tmp;

**int** index = *x* / 2;

// Iterative heapiFy

**while** (index >= 0)

{

// Just swapping if the element

// is smaller than already

// stored element

**if** (*heap*[index].compareTo(*heap*[child]) > 0)

{

// Swapping the current index

// with its child

tmp = *heap*[index];

*heap*[index] = *heap*[child];

*heap*[child] = tmp;

child = index;

// Moving upward in the

// heap

index = index / 2;

}

**else**

{

**break**;

}

}

}

// Defining heap sort

**static** **void** heapSort()

{

**int** left1, right1;

**while** (*x* >= 0)

{

String k;

k = *heap*[0];

// Taking output of

// the minimum element

System.***out***.print(k + " ");

// Set first element

// as a last one

*heap*[0] = *heap*[*x*];

// Decrement of the

// size of the string

*x* = *x* - 1;

String tmp;

**int** index = 0;

**int** length = *x*;

// Initializing the left

// and right index

left1 = 1;

right1 = left1 + 1;

**while** (left1 <= length)

{

// Process of heap sort

// If root element is

// minimum than its both

// of the child then break

**if** (*heap*[index].compareTo(*heap*[left1]) <= 0 &&

*heap*[index].compareTo(*heap*[right1]) <= 0)

{

**break**;

}

// Otherwise checking that

// the child which one is

// smaller, swap them with

// parent element

**else**

{

// Swapping

**if** (*heap*[left1].compareTo(*heap*[right1])< 0)

{

tmp = *heap*[index];

*heap*[index] = *heap*[left1];

*heap*[left1] = tmp;

index = left1;

}

**else**

{

tmp = *heap*[index];

*heap*[index] = *heap*[right1];

*heap*[right1] = tmp;

index = right1;

}

}

// Changing the left index

// and right index

left1 = 2 \* left1;

right1 = left1 + 1;

}

}

}

// Utility function

**static** **void** sort(String k[], **int** n)

{

// To heapiFy

**for** (**int** i = 0; i < n; i++)

{

*heapForm*(k[i]);

}

// Calling heap sort function

*heapSort*();

}

// Driver Code

**public** **static** **void** main(String[] args)

{

String arr[] = {"All Quiet on the Western Front","The Sound of Music","Ben-Hur2","Ben-Hur1"

,"Dunkirk","1917","Mad Max: Fury Road","Mad Max: Beyond Thunderdome","Mad Max 2"

,"Mad Max","Star Trek: The Motion Picture","Star Trek II: The Wrath of Khan","Star Trek III: The Search for Spock"

,"Star Trek IV: The Voyage Home","Star Trek V: The Final Frontier","Star Trek VI: The Undiscovered Country"

,"Star Trek: Generations","Star Trek: First Contact","Star Trek: Insurrection","Star Trek: Nemesis"};

//System.out.println(Arrays.toString(arr));

**int** n = arr.length;

*sort*(arr, n);

}

}

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Output:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The program sucessfully performs the sort operation.

Heap Sort has O(nlog n) time complexities for all the cases ( best case, average case, and worst case).

**7. GitHub link:**