



**UNIVERSITI  
MALAYA**

*Faculty of Computer Science  
and Information Technology*

**WIA1002 DATA STRUCTURES**

**ASSIGNMENT REPORT GROUP '4 DATA STRUCTURER' (OCC 5)**

**STUDENTS NAME:**

**AHMAD IRSYAD BIN AHMAD FAIZAL  
MATRICS NUMBER: U2000621**

**HANIFF BIN HASRI  
MATRICS NUMBER: U2000464**

**MUHAMMAD MUQRI QAWIEM BIN HANIZAM  
MATRICS NUMBER: U2000726**

**AIDIL AZHAR IKMAL BIN ABDUL SANI  
MATRICS NUMBER: U2000851**

## **Table of contents**

<b>1.0 Introduction of the project (Topic D: World of Titan)</b>	<b>4</b>
<b>2.0 Members and responsibilities</b>	<b>4</b>
<b>3.0 Project flow chart</b>	<b>5</b>
<b>4.0 Basic requirements</b>	<b>6</b>
<b>4.1 Eren's allies</b>	<b>6</b>
<b>4.1.1 Source code</b>	<b>11</b>
<b>4.1.1.1 Eren's Allies.txt source code</b>	<b>11</b>
<b>4.1.2 Sample output</b>	<b>12</b>
<b>4.2 Soldiers arrangement and grouping</b>	<b>15</b>
<b>4.2.1 Source code</b>	<b>15</b>
<b>4.2.1.1 DoublyLinkedList.java source code</b>	<b>15</b>
<b>4.2.1.2 Node.java source code</b>	<b>19</b>
<b>4.2.1.3 SortList.java source code</b>	<b>20</b>
<b>4.2.1.4 Characters.java source code</b>	<b>22</b>
<b>4.2.2 Sample output</b>	<b>24</b>
<b>4.3 Titan evaluation and killing priority</b>	<b>25</b>
<b>4.3.1 Source code</b>	<b>26</b>
<b>NormalTitan.java source code</b>	<b>26</b>
<b>Titan.java source code</b>	<b>27</b>
<b>PriorityQueue.java source code</b>	<b>27</b>
<b>TitanTag.java source code</b>	<b>29</b>
<b>DoublyLinkedList.java source code</b>	<b>29</b>
<b>Node.java source code</b>	<b>33</b>
<b>4.3.2 Sample output</b>	<b>35</b>
<b>4.4 Scouting mission inside the wall</b>	<b>36</b>
<b>4.4.1 Source code</b>	<b>37</b>
<b>4.4.1.1 ScoutingClass.java source code</b>	<b>37</b>
<b>4.4.2 Sample output</b>	<b>39</b>
<b>4.5 Best path to kill Titan</b>	<b>40</b>
<b>4.5.1 Source code</b>	<b>41</b>
<b>4.5.1.1 BestPathToKillTitan.java source code</b>	<b>41</b>

4.5.2 Sample output .....	44
4.6 Marley word converter .....	45
4.6.1 Source code .....	46
4.6.1.1 CustomHashMap.java source code .....	46
4.6.2 Sample output .....	50
4.7 Protecting Wall of Maria .....	51
4.7.1 Source code .....	52
4.7.1.1 WeakestWallOfMaria.java source code .....	52
4.7.2 Sample output .....	54
4.8 Main.java source code .....	55
5.0 Extra features .....	116
6.0 Conclusion .....	117
7.0 References .....	118

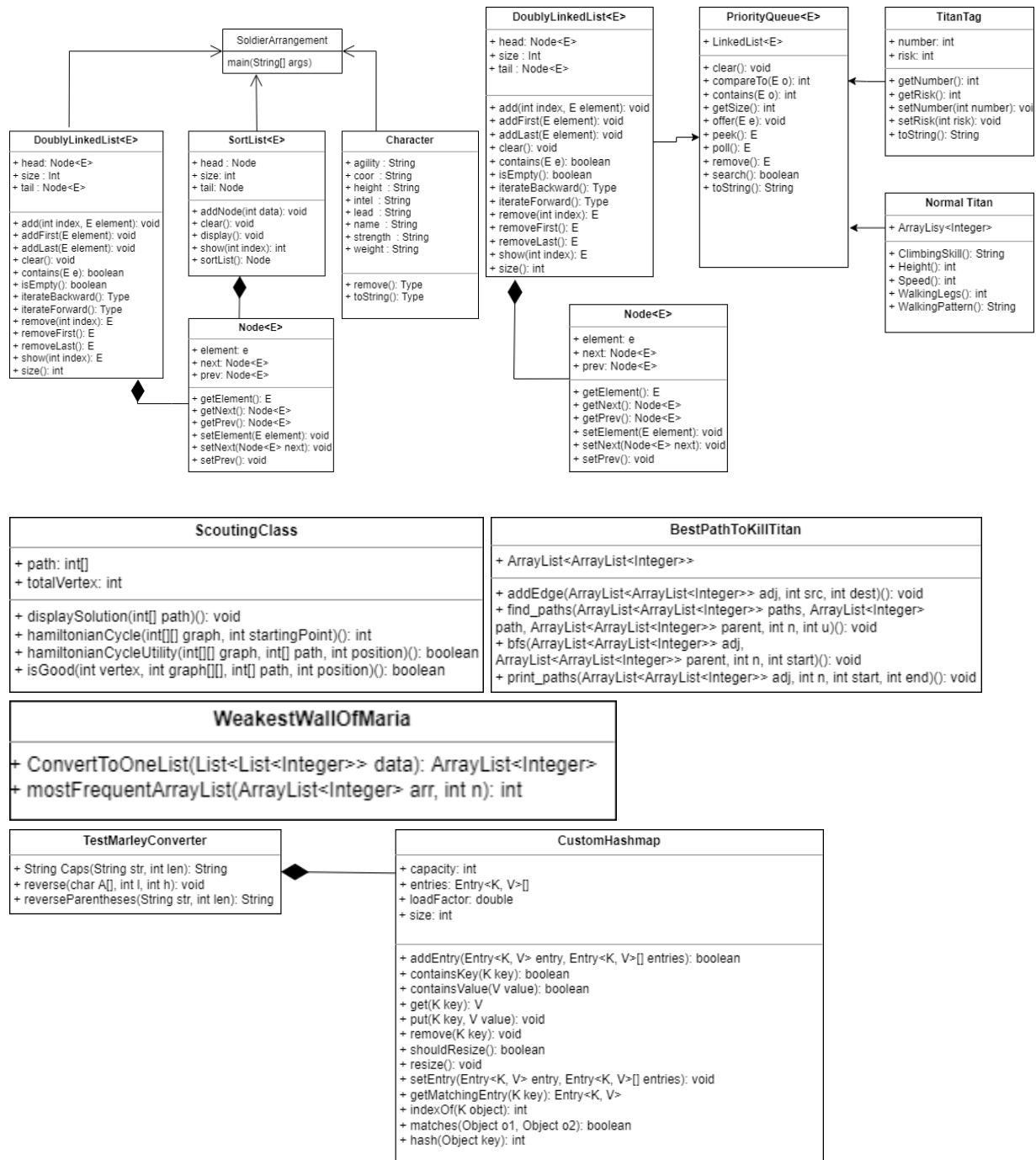
## 1.0 Introduction of the project (Topic D: World of Titan)

Several hundred years ago, humanity was driven to the brink of extinction by the humanoid giants called Titans, who apparently have no intelligence, and only attack and eat humans on sight. A small portion of humans retreated and formed a country behind extremely high walls, known as Paradis. Eren Yeager, a young man, suffered his childhood without his mother's love because she was eaten by a titan in one titan ambush incident. Since then, he decided to join the country's military forces and confront his fate to eliminate titans from the world. As Eren's friend who is familiar with the data structure knowledge, we need to help Eren solve the problems.

## 2.0 Members and responsibilities

Members	Responsibilities
AHMAD IRSYAD BIN AHMAD FAIZAL	<ul style="list-style-type: none"><li>• 2.1 Eren's Allies</li><li>• 2.2 Soldier Arrangement</li><li>• 2.4 Best Path To Kill Titan</li><li>• 2.6 Weakest Wall of Maria</li></ul>
HANIFF BIN HASRI	<ul style="list-style-type: none"><li>• 2.3 Titan Evaluation and Killing Priority</li><li>• Report writing</li></ul>
MUHAMMAD MUQRI QAWIEM BIN HANIZAM	<ul style="list-style-type: none"><li>• 2.3 Scouting Mission inside the Wall</li><li>• Compiling all source code</li><li>• Report writing</li></ul>
AIDIL AZHAR IKMAL BIN ABDUL SANI	<ul style="list-style-type: none"><li>• 2.5 Marley word converter</li><li>• Report writing</li></ul>


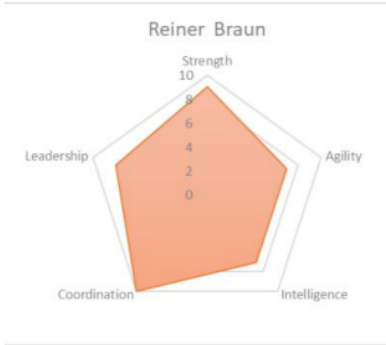

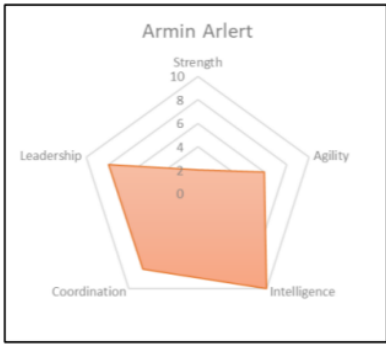
### 3.0 Project flow chart


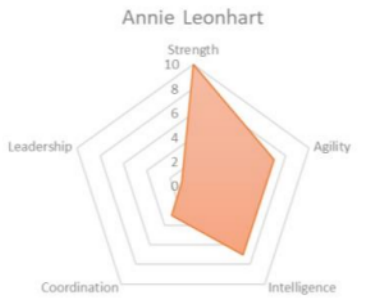

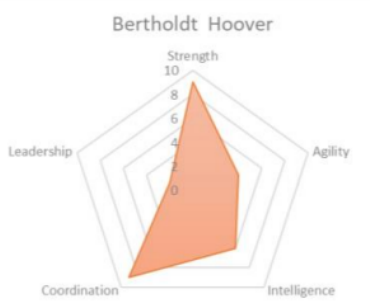

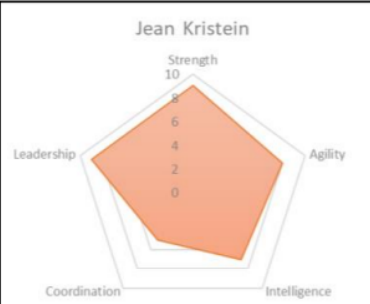




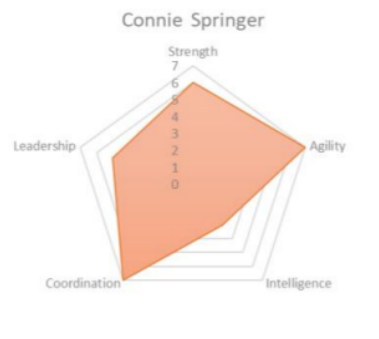


## 4.0 Basic requirements

### 4.1 Eren's allies

Eren has too many friends in the military, and Eren always forgets what the characteristics of their friends are. But Eren has provided us with a list that needs to be recorded in character class. The list below is from the note provided by Eren.

Name	Characteristic	Characteristic in graph view
<div>Reiner Braun</div> 	<ul style="list-style-type: none"><li>○ Height: 185cm</li><li>○ Weight: 95kg</li><li>○ Strength: 9</li><li>○ Agility: 7</li><li>○ Intelligence: 7</li><li>○ Coordination: 10</li><li>○ Leadership: 8</li></ul>	
<div>Armin Arlert</div> 	<ul style="list-style-type: none"><li>○ Height: 163cm</li><li>○ Weight: 55kg</li><li>○ Strength: 2</li><li>○ Agility: 6</li><li>○ Intelligence: 10</li><li>○ Coordination: 8</li><li>○ Leadership: 8</li></ul>	

<p>Annie Leonhart</p> 	<ul style="list-style-type: none"> <li>○ Height: 153cm</li> <li>○ Weight: 54kg</li> <li>○ Strength: 10</li> <li>○ Agility: 7</li> <li>○ Intelligence: 7</li> <li>○ Coordination: 3</li> <li>○ Leadership: 1</li> </ul>	<p>Annie Leonhart</p> 
<p>Bertholdt Hoover</p> 	<ul style="list-style-type: none"> <li>○ Height: 192cm</li> <li>○ Weight: 81kg</li> <li>○ Strength: 9</li> <li>○ Agility: 4</li> <li>○ Intelligence: 6</li> <li>○ Coordination: 9</li> <li>○ Leadership: 2</li> </ul>	<p>Bertholdt Hoover</p> 
<p>Jean Krstein</p> 	<ul style="list-style-type: none"> <li>○ Height: 175cm</li> <li>○ Weight: 65kg</li> <li>○ Strength: 9</li> <li>○ Agility: 8</li> <li>○ Intelligence: 7</li> <li>○ Coordination: 5</li> <li>○ Leadership: 9</li> </ul>	<p>Jean Krstein</p> 

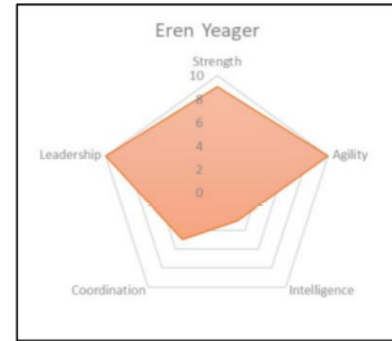
<p>Sasha Blouse</p> 	<ul style="list-style-type: none"> <li>○ Height: 165cm</li> <li>○ Weight: 55kg</li> <li>○ Strength: 6</li> <li>○ Agility: 3</li> <li>○ Intelligence: 5</li> <li>○ Coordination: 6</li> <li>○ Leadership: 7</li> </ul>	
<p>Connie Springer</p> 	<ul style="list-style-type: none"> <li>○ Height: 158cm</li> <li>○ Weight: 58kg</li> <li>○ Strength: 6</li> <li>○ Agility: 7</li> <li>○ Intelligence: 3</li> <li>○ Coordination: 7</li> <li>○ Leadership: 5</li> </ul>	
<p>Mikasa Ackerman</p> 	<ul style="list-style-type: none"> <li>○ Height: 170kg</li> <li>○ Weight: 68kg</li> <li>○ Strength: 10</li> <li>○ Agility: 9</li> <li>○ Intelligence: 8</li> <li>○ Coordination: 6</li> <li>○ Leadership: 7</li> </ul>	



Eren Yeager



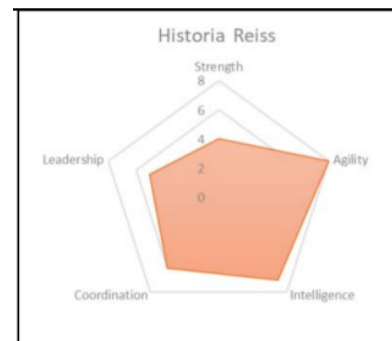
- Height: 170cm:
- Weight: 63kg
- Strength: 9
- Agility: 10
- Intelligence: 3
- Coordination: 5
- Leadership: 10



Historia Reiss



- Height: 145cm
- Weight: 42kg
- Strength: 4
- Agility: 8
- Intelligence: 7
- Coordination: 6
- Leadership: 5



Levi Ackerman



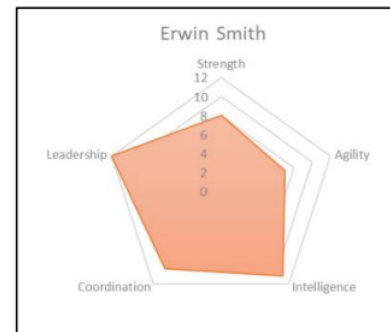
- Height: 160cm
- Weight: 65kg
- Strength: 12
- Agility: 12
- Intelligence: 7
- Coordination: 8
- Leadership: 8



Erwin Smith



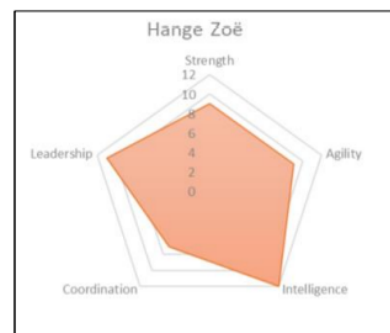
- Height: 188cm
- Weight: 92kg
- Strength: 8
- Agility: 8
- Intelligence: 11
- Coordination: 10
- Leadership: 12



Hange Zoë



- Height: 170cm
- Weight: 60kg
- Strength: 9
- Agility: 9
- Intelligence: 12
- Coordination: 7
- Leadership: 11



#### **4.1.1 Source code**

##### **4.1.1.1 Eren's Allies.txt source code**

Reiner Braun 185 95 9 7 7 10 8  
Armin Arlert 163 55 2 6 10 8 8  
Annie Leonhart 153 54 10 7 7 3 1  
Bertholdt Hoover 192 81 9 4 6 9 2  
Jean Kiste 175 65 9 8 7 5 9  
Sasha Blouse 165 55 6 3 5 6 7  
Connie Springer 158 58 6 7 3 7 5  
Mikasa Ackerman 170 68 10 9 8 6 7  
Eren Yeager 170 63 9 10 3 5 10  
Historia Reiss 145 42 4 8 7 6 5  
Levi Ackerman 160 65 12 12 7 8 8

#### 4.1.2 Sample output

```
***** Soldiers Available *****  
  
name : Reiner Braun  
height : 185cm  
weight : 95kg  
strength : 9  
agility : 7  
intelligence : 7  
coordination : 10  
leadership : 8  
  
name : Armin Arlert  
height : 163cm  
weight : 55kg  
strength : 2  
agility : 6  
intelligence : 10  
coordination : 8  
leadership : 8  
  
name : Annie Leonhart  
height : 153cm  
weight : 54kg  
strength : 10  
agility : 7  
intelligence : 7  
coordination : 3  
leadership : 1  
  
name : Bertholdt Hoover  
height : 192cm  
weight : 81kg  
strength : 9  
agility : 4  
intelligence : 6  
coordination : 9  
leadership : 2
```

name : Jean Kristein  
height : 175cm  
weight : 65kg  
strength : 9  
agility : 8  
intelligence : 7  
coordination : 5  
leadership : 9

name : Sasha Blouse  
height : 165cm  
weight : 55kg  
strength : 6  
agility : 3  
intelligence : 5  
coordination : 6  
leadership : 7

name : Connie Springer  
height : 158cm  
weight : 58kg  
strength : 6  
agility : 7  
intelligence : 3  
name : Mikasa Ackerman  
height : 170cm  
weight : 68kg  
strength : 10  
agility : 9  
intelligence : 8  
coordination : 6  
leadership : 7

name : Eren Yeager  
height : 170cm  
weight : 63kg  
strength : 9  
agility : 10  
intelligence : 3  
coordination : 5  
leadership : 10

name : Historia Reiss  
height : 145cm  
weight : 42kg  
strength : 4  
agility : 8  
intelligence : 7  
coordination : 6  
leadership : 5

name : Levi Ackerman  
height : 160cm  
weight : 65kg  
strength : 12  
agility : 12  
intelligence : 7  
coordination : 8  
leadership : 8

## 4.2 Soldiers arrangement and grouping

Captain Erwin always has a hard time arranging the soldiers. Captain Erwin wants to put the soldiers in a certain order all the time. We need to use any kind of sorting method to sort the members based on the different types of attributes of the members, such as height, weight, and ability. To carry out the investigating action outside the Wall, Captain Erwin needs to find soldiers by using ability value to make team forming easier.

### 4.2.1 Source code

#### 4.2.1.1 DoublyLinkedList.java source code

```
import java.util.NoSuchElementException;

public class DoublyLinkedList<E> {

    private Node<E> head;
    private Node<E> tail;
    private int size;

    public DoublyLinkedList() {
        size = 0;
        this.head = null;
        this.tail = null;
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void addFirst(E element) {
        // create object tmp and set pointer of the new node
        Node<E> tmp = new Node(element, head, null);
        // set head.prev of current head to be linked to the new
node
        if (head != null) {
            head.prev = tmp;
        }
        head = tmp; // now tmp become head
        // if no tail, then tmp set to be a tail
        if (tail == null) {
            tail = tmp;
        }
    }
}
```

```

        size++; // increase number of node
        // System.out.println(element);
    }

    public void addLast(E element) {
        // create object tmp and set pointer of the previous node
        Node<E> tmp = new Node(element, null, tail);
        // set tail.next point to object tmp
        if (tail != null) {
            tail.next = tmp;
        }
        // now tmp become tail
        tail = tmp;
        // if no head, then tmp set to be a head
        if (head == null) {
            head = tmp;
        }
        size++; // increase number of node
        // System.out.println(element);
    }

    public void iterateForward() {

        System.out.println("iterating forward..");
        Node<E> tmp = head;
        while (tmp != null) {
            System.out.print(tmp.element);
            System.out.print(" ");
            tmp = tmp.next;
        }
    }

    public void iterateBackward() {

        System.out.println("iterating backward..");
        Node<E> tmp = tail;
        while (tmp != null) {
            System.out.print(tmp.element);
            System.out.print(" ");
            tmp = tmp.prev;
        }
    }

    public E removeFirst() {
        if (size == 0)
            throw new NoSuchElementException();
        // copy head to node tmp
        Node<E> tmp = head;
        // head.next become a head
    }

```



```

        head = head.next;
        // set pointer of prev of new head to be null
        if (head != null)
            head.prev = null;
        // reduce number of node
        size--;
        // System.out.println(tmp.element);
        return tmp.element;
    }

    public E removeLast() {
        if (size == 0)
            throw new NoSuchElementException();
        // copy tail to node tmp
        Node<E> tmp = tail;
        // tail.prev become a tail
        tail = tail.prev;
        // set pointer of next of new tail to be null
        tail.next = null;
        // reduce number of node
        size--;
        // System.out.println(tmp.element);
        System.out.println("removed");
        return tmp.element;
    }

    public void add(int index, E element) {
        if (index == 0) {
            addFirst(element);
        } else if (index >= size) {
            addLast(element);
        } else {
            Node<E> current = head;
            for (int i = 1; i < index; i++) {
                current = current.next;
            }
            Node<E> temp = current.next;
            current.next = new Node<E>(element);
            (current.next).next = temp;
            size++;
            System.out.println("added");
        }
    }

    public E remove(int index) {
        E element = null;
        if (index < 0 || index >= size)
            throw new IndexOutOfBoundsException();
        if (index == 0)

```

```

        element = removeFirst();
    else if (index == size - 1)
        element = removeLast();
    else {
        Node<E> temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp.next;
        }
        element = temp.element;
        temp.next.prev = temp.prev;
        temp.prev.next = temp.next;
        temp.next = null;
        temp.prev = null;
        size--;
    }

    return element;
}

public void clear() {
    head = tail = null;
    size = 0;
    // System.out.println("clearing all nodes...");
}

public E show(int index) {
    if (index < 0 || index >= size) {
        throw new IndexOutOfBoundsException();
    }
    Node<E> temp = head;
    for (int i = 0; i < index; i++) {
        temp = temp.next;
    }
    E element = temp.element;
    return element;
}

public boolean contains(E e) {
    Node<E> current = head;
    while (current != null) {
        if (current.element.equals(e)) {
            return true;
        }
        current = current.next;
    }

    return false;
}

```

```
}
```

#### 4.2.1.2 Node.java source code

```
//Used Node class from tutorial
//Node is used in DoublyLinkedList class
public class Node<E> {
    E element;
    Node<E> next;
    Node<E> prev;
    // int elementInt = (int) element;

    public Node(E element, Node next, Node prev) {
        this.element = element;
        this.next = next;
        this.prev = prev;
    }

    public Node(E element) {
        this(element, null, null);
    }

    public E getElement() {
        return element;
    }

    public void setElement(E element) {
        this.element = element;
    }

    public Node<E> getNext() {
        return next;
    }

    public void setNext(Node<E> next) {
        this.next = next;
    }

    public Node<E> getPrev() {
        return prev;
    }

    public void setPrev(Node<E> prev) {
        this.prev = prev;
    }
}
```

#### 4.2.1.3 SortList.java source code

```
public class SortList<E> {
    public class Node {
        int data;
        Node next;
        private int size;

        public Node(int data) {
            this.data = data;
            this.next = null;
            size = 0;
        }
    }

    // Represent the head and tail of the singly linked list
    public Node head = null;
    public Node tail = null;
    public int size = 0;

    // addNode() will add a new node to the list
    public void addNode(int data) {
        // Create a new node
        Node newNode = new Node(data);

        // Checks if the list is empty
        if (head == null) {
            // If list is empty, both head and tail will point to
new node
            head = newNode;
            tail = newNode;
        } else {
            // newNode will be added after tail such that tail's
next will point to newNode
            tail.next = newNode;
            // newNode will become new tail of the list
            tail = newNode;
        }
        size++;
    }

    // sortList() will sort nodes of the list in ascending order
    public void sortList() {
        // Node current will point to head
        Node current = head, index = null;
        int temp;

        if (head == null) {
            return;
        }
    }
}
```

```

    } else {
        while (current != null) {
            // Node index will point to node next to current
            index = current.next;

            while (index != null) {
                // If current node's data is greater than
index's node data, swap the data
                // between them
                if (current.data < index.data) {
                    temp = current.data;
                    current.data = index.data;
                    index.data = temp;
                }
                index = index.next;
            }
            current = current.next;
        }
    }
}

public int show(int index) {
    Node temp = head;
    for (int i = 0; i < index; i++) {
        temp = temp.next;
    }
    int element = temp.data;
    return element;
}

// display() will display all the nodes present in the list
public void display() {
    // Node current will point to head
    Node current = head;
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    while (current != null) {
        // Prints each node by incrementing pointer
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

public void clear() {
    head = tail = null;
    size = 0;
}

```

```
}  
  
}
```

#### 4.2.1.4 Characters.java source code

```
public class Characters {  
    String name, height, weight, strength, agility, intel, coor,  
    lead;  
  
    public Character(String a, String b, String c, String d, String  
e, String f, String g, String h) {  
        name = a;  
        height = b;  
        weight = c;  
        strength = d;  
        agility = e;  
        intel = f;  
        coor = g;  
        lead = h;  
        System.out.println(toString());  
    }  
  
    public void remove(String a, String b, String c, String d,  
String e, String f, String g, String h) {  
        name = null;  
        height = null;  
        weight = null;  
        strength = null;  
        agility = null;  
        intel = null;  
        coor = null;  
        lead = null;  
    }  
  
    @Override  
    public String toString() {  
        return "" + "\nname : " + name + "\nheight : " + height +  
"cm" + "\nweight : " + weight + "kg"  
            + "\nstrength : " + strength + "\nagility : " +  
agility + "\nintelligence : " + intel  
            + "\ncoordination : " + coor + "\nleadership : " +  
lead;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

```
public String getHeight() {  
    return height;  
}  
  
public String getWeight() {  
    return weight;  
}  
  
public String getStrength() {  
    return strength;  
}  
  
public String getAgility() {  
    return agility;  
}  
  
public String getIntel() {  
    return intel;  
}  
  
public String getCoor() {  
    return coor;  
}  
  
public String getLead() {  
    return lead;  
}  
}
```

#### 4.2.2 Sample output

```
Do you want to sort attribute? : y
Available attributes : height, weight, strength, agility, intelligence, coordination, leadership

Sorting attribute: weight

Sorting from highest to lowest :

Reiner Braun 95
Bertholdt Hoover 81
Mikasa Ackerman 68
Jean Kristein 65
Levi Ackerman 65
Eren Yeager 63
Connie Springer 58
Armin Arlert 55
Sasha Blouse 55
Annie LeonHart 54
Historia Reiss 42
```

Sample output for sorting weight attribute.

```
Do you want to search for ability? : y
Available ability : height, weight, strength, agility, intelligence, coordination, leadership

Finding ability : strength
Value : 10

Soldier :
Annie LeonHart, Mikasa Ackerman,
```

Sample output for searching strength ability for value 10.

```
Do you want to search for ability? : y
Available ability : height, weight, strength, agility, intelligence, coordination, leadership

Finding ability : intelligence
Value : 9

soldier :
No soldiers found :(
```

Sample output for searching intelligence ability for value 9.



### 4.3 Titan evaluation and killing priority

Eren Yeager and his team had noticed that there were many different danger levels of the Titan. Titans with some features will be more dangerous and they will make more people die. These titans also have strong ability in fighting with the soldiers. This is how we evaluate the dangers of titans:

If Titan type is normal (sum all the danger risk to get the total danger risk):

- Height
  - height > 20m - danger risk 3
  - height > 10m - danger risk 2
  - height < 10m - danger risk 1
- Walking legs
  - 4 legs - danger risk 3
  - 2 legs - danger risk 2
  - 0 legs - danger risk 1
- Running speed
  - Speed > 20ms - danger risk 3
  - Speed > 10ms - danger risk 2
  - Speed < 10ms - danger risk 1
- Walking pattern
  - Not repeated pattern - danger risk 3
  - Repeated pattern - danger risk 2
  - Normal pattern - danger risk 1
- Climbing skill
  - Can climb - danger risk 3
  - Cannot climb - danger risk 2

If Titan type is abnormal:

- Danger risk 15

If Titan type is one of the nine Titan:

- Danger risk 19

To ensure the safety of the residents, we should choose to kill Titans with the greatest danger risk first. We construct a priority queue to put the Titans' details and compute which Titan should be killed first. We calculated the distance moved in terms of the difference in index position of the titans in the input.

### 4.3.1 Source code

#### NormalTitan.java source code

```
import java.util.ArrayList;
import java.util.Random;

public class NormalTitan<T> {
    ArrayList<Integer> e = new ArrayList<>();

    public int Height() {
        Random r = new Random();
        int a = r.nextInt(35) + 1;
        if (a > 20) {
            return a;
        } else if (a > 10) {
            return a;
        } else {
            return a;
        }
    }

    public int WalkingLegs() {
        Random r = new Random();
        int a = r.nextInt(4) + 1;
        while (a != 1 && a != 3) {
            if (a == 4) {
                return 4;
            } else if (a == 2) {
                return 2;
            } else {
                return 0;
            }
        }
        return 0;
    }

    public int Speed() {
        Random r = new Random();
        int a = r.nextInt(30) + 1;
        if (a > 20) {
            return a;
        } else if (a > 10) {
            return a;
        } else {
            return a;
        }
    }
}
```

```

    public String WalkingPattern() {
        String[] s = { "Not repeated pattern", "Repeated pattern",
"Normal pattern" };
        Random r = new Random();
        int randomNumber = r.nextInt(s.length);
        if (randomNumber == 0) {
            return "Not repeated pattern";
        } else if (randomNumber == 1) {
            return "Repeated pattern";
        } else {
            return "Normal pattern";
        }
    }

    public String ClimbingSkill() {
        String[] s = { "Can climb", "Cannot climb" };
        Random r = new Random();
        int randomNumber = r.nextInt(s.length);
        if (randomNumber == 0) {
            return "Can climb";
        } else {
            return "Cannot climb";
        }
    }
}

```

### Titan.java source code

```

import java.util.*;

public class Titan extends NormalTitan {
    public int AbnormalTitan() {
        return 15;
    }

    public int NineTitan() {
        return 19;
    }
}

```

### PriorityQueue.java source code

```

public class PriorityQueue<E> implements Comparable<E> {
    private final java.util.LinkedList<E> list = new
java.util.LinkedList<>();
}

```

```

    public void offer(E e) {
        list.offer(e);
    }

    public E remove() {
        return list.remove();
    }

    public E peek() {
        return list.peek();
    }

    public E poll() {
        return list.poll();
    }

    public void clear() {
        list.clear();
    }

    public int getSize() {
        return list.size();
    }

    public boolean contains(E o) {
        return list.contains(o);
    }

    @Override
    public String toString() {
        return "Sequence to be killed : " + list.toString();
    }

    public boolean search(E o) {
        return list.contains(o);
    }

    @Override
    public int compareTo(E o) {
        throw new UnsupportedOperationException("Not supported
yet."); // To change body of generated methods, choose
// Tools | Templates.
    }
}

```

### TitanTag.java source code

```
public class TitanTag {
    int number;
    int risk;

    public TitanTag(int number, int risk) {
        this.number = number;
        this.risk = risk;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public int getRisk() {
        return risk;
    }

    public void setRisk(int risk) {
        this.risk = risk;
    }

    @Override
    public String toString() {
        return "Titan " + number; // To change body of generated
methods, choose Tools | Templates.
    }
}
```

### DoublyLinkedList.java source code

```
import java.util.NoSuchElementException;

public class DoublyLinkedList<E> {
    private Node<E> head;
    private Node<E> tail;
    private int size;

    public DoublyLinkedList() {
        size = 0;
        this.head = null;
        this.tail = null;
    }
}
```

```

    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void addFirst(E element) {
        // create object tmp and set pointer of the new node
        Node<E> tmp = new Node(element, head, null);
        // set head.prev of current head to be linked to the new
node
        if (head != null) {
            head.prev = tmp;
        }
        head = tmp; // now tmp become head
        // if no tail, then tmp set to be a tail
        if (tail == null) {
            tail = tmp;
        }
        size++; // increase number of node
        // System.out.println(element);
    }

    public void addLast(E element) {
        // create object tmp and set pointer of the previous node
        Node<E> tmp = new Node(element, null, tail);
        // set tail.next point to object tmp
        if (tail != null) {
            tail.next = tmp;
        }
        // now tmp become tail
        tail = tmp;
        // if no head, then tmp set to be a head
        if (head == null) {
            head = tmp;
        }
        size++; // increase number of node
        // System.out.println(element);
    }

    public void iterateForward() {

        System.out.println("iterating forward..");
        Node<E> tmp = head;
        while (tmp != null) {

```

```

        System.out.print(tmp.element);
        System.out.print(" ");
        tmp = tmp.next;
    }
}

public void iterateBackward() {

    System.out.println("iterating backward..");
    Node<E> tmp = tail;
    while (tmp != null) {
        System.out.print(tmp.element);
        System.out.print(" ");
        tmp = tmp.prev;
    }
}

public E removeFirst() {
    if (size == 0)
        throw new NoSuchElementException();
    // copy head to node tmp
    Node<E> tmp = head;
    // head.next become a head
    head = head.next;
    // set pointer of prev of new head to be null
    if (head != null)
        head.prev = null;
    // reduce number of node
    size--;
    // System.out.println(tmp.element);
    return tmp.element;
}

public E removeLast() {
    if (size == 0)
        throw new NoSuchElementException();
    // copy tail to node tmp
    Node<E> tmp = tail;
    // tail.prev become a tail
    tail = tail.prev;
    // set pointer of next of new tail to be null
    tail.next = null;
    // reduce number of node
    size--;
    // System.out.println(tmp.element);
    return tmp.element;
}

public void add(int index, E element) {

```

```

        if (index == 0) {
            addFirst(element);
        } else if (index >= size) {
            addLast(element);
        } else {
            Node<E> current = head;
            for (int i = 1; i < index; i++) {
                current = current.next;
            }
            Node<E> temp = current.next;
            current.next = new Node<E>(element);
            (current.next).next = temp;
            size++;
        }
    }

    public E remove(int index) {
        E element = null;
        if (index < 0 || index >= size)
            throw new IndexOutOfBoundsException();
        if (index == 0)
            element = removeFirst();
        else if (index == size - 1)
            element = removeLast();
        else {
            Node<E> temp = head;
            for (int i = 0; i < index; i++) {
                temp = temp.next;
            }
            element = temp.element;
            temp.next.prev = temp.prev;
            temp.prev.next = temp.next;
            temp.next = null;
            temp.prev = null;
            size--;
        }

        return element;
    }

    public void clear() {
        head = tail = null;
        size = 0;
        // System.out.println("clearing all nodes...");
    }

    public E show(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
    }

```



```

    }
    Node<E> temp = head;
    for (int i = 0; i < index; i++) {
        temp = temp.next;
    }
    E element = temp.element;
    return element;
}

public boolean contains(E e) {
    Node<E> current = head;
    while (current != null) {
        if (current.element.equals(e)) {
            return true;
        }
        current = current.next;
    }
    return false;
}
}

```

### Node.java source code

```

public class Node<E> {
    E element;
    Node<E> next;
    Node<E> prev;
    // int elementInt = (int) element;

    public Node(E element, Node next, Node prev) {
        this.element = element;
        this.next = next;
        this.prev = prev;
    }

    public Node(E element) {
        this(element, null, null);
    }

    public E getElement() {
        return element;
    }
}

```

```
public void setElement(E element) {  
    this.element = element;  
}  
  
public Node<E> getNext() {  
    return next;  
}  
  
public void setNext(Node<E> next) {  
    this.next = next;  
}  
  
public Node<E> getPrev() {  
    return prev;  
}  
  
public void setPrev(Node<E> prev) {  
    this.prev = prev;  
}  
}
```

### 4.3.2 Sample output

```
Number of Titans: 6

Generating 6 Titans ....
Titan 1: Normal Titan (26m, 0 legs, 15ms, Not repeated pattern, Can climb) Risk 12
Titan 2: Normal Titan (26m, 0 legs, 5ms, Normal pattern, Cannot climb) Risk 9
Titan 3: Nine Titan (War Hammer Titan) Risk 19
Titan 4: Normal Titan (19m, 4 legs, 15ms, Not repeated pattern, Cannot climb) Risk 13
Titan 5: Nine Titan (Jaw Titan) Risk 19
Titan 6: Normal Titan (32m, 4 legs, 17ms, Repeated pattern, Cannot climb) Risk 9

Sequence to be killed : Titan 2 --> Titan 6 --> Titan 1 --> Titan 4 --> Titan 3 --> Titan 5
```

Sample output for randomly generated six Titans.

```
Number of Titans: 1

Generating 1 Titans ....
Titan 1: Normal Titan (22m, 4 legs, 29ms, Normal pattern, Cannot climb) Risk 9

Sequence to be killed : Titan 1
```

Sample output for randomly generated one Titans.

```
Number of Titans: 3

Generating 3 Titans ....
Titan 1: Nine Titan (Female Titan) Risk 19
Titan 2: Abnormal Titan Risk 15
Titan 3: Normal Titan (30m, 4 legs, 3ms, Repeated pattern, Can climb) Risk 7

Sequence to be killed : Titan 3 --> Titan 2 --> Titan 1
```

Sample output for randomly generated three Titans.

#### 4.4 Scouting mission inside the wall

Captain Erwin gets the information from the garrison that there are some titans invading the wall (noticed that those titans are summoned by Beast Titan). After walking through the wall, the survey corps found that there is no difference on the Wall. But, to clear all the titans inside that area. Scouting missions should be carried out to notice all titans' positions.

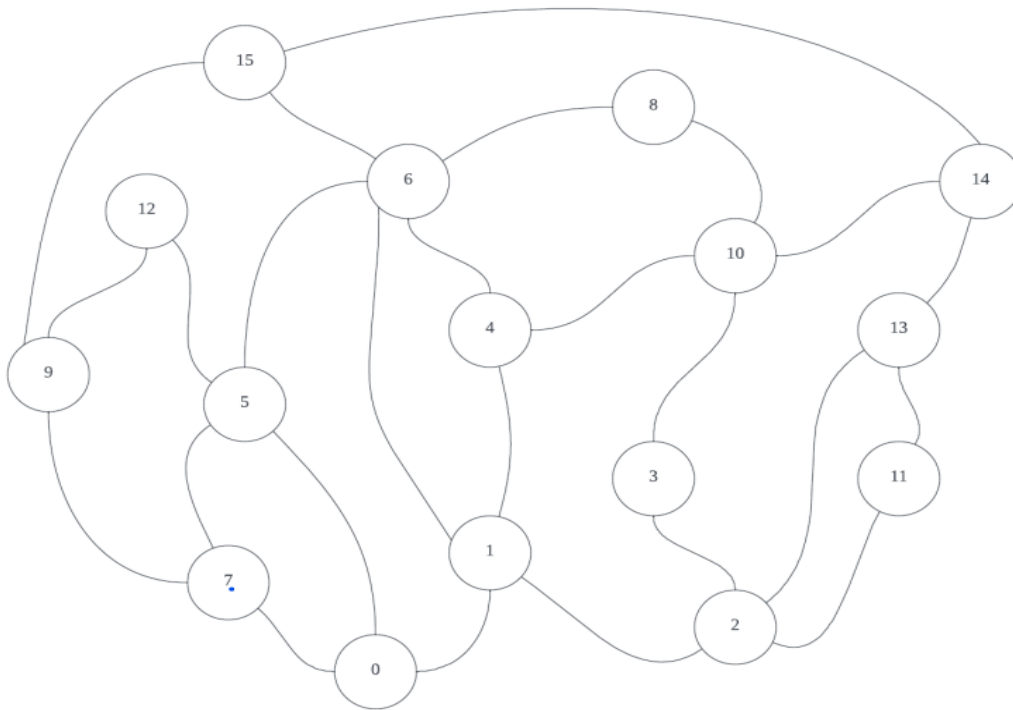


Figure 1: Map nodes in Paradis which the soldier move using moving machine

We need to find one point that can search all the points without repeating the points. Then, getting back to that starting point using the Hamiltonian cycle.

#### 4.4.1 Source code

##### 4.4.1.1 ScoutingClass.java source code

```
public class ScoutingClass {
    final int totalVertex = 16; // total number of nodes inside the
    map
    int[] path;

    /**
     * Utility function to check if the 'int vertex' can be added
     to the index
     * 'int position' that are stored in array 'int[] path'
     */
    public boolean isGood(int vertex, int graph[][], int[] path,
    int position) {
        /**
         * First criteria:
         * To check if the vertex is an adjacent vertex of the
    previous vertex
         */
        if (graph[path[position - 1]][vertex] == 0)
            return false;

        /**
         * Second criteria:
         * To check if the vertex already included
         */
        for (int i = 0; i < position; i++)
            if (path[i] == vertex)
                return false;

        return true;
    }

    /** Recursive utility function to find hamiltonian cyle */
    public boolean hamiltonianCycleUtility(int[][] graph, int[]
    path, int position) {
        // if all vertex included in the hamiltonian cycle
        if (position == totalVertex) {
            // if there is vertex from last included vertex to the
    first vertex
            if (graph[path[position - 1]][path[0]] == 1)
                return true;
            else
                return false;
        }
    }
```

```

        // trying different vertex
        for (int v = 0; v < totalVertex; v++) {
            // check if this vertex is good to be added into the
hamiltonian cycle
            if (isGood(v, graph, path, position)) {
                path[position] = v;

                // continue to construct the rest of the path
                if (hamiltonianCycleUtility(graph, path, position +
1) == true)
                    return true;

                // if adding vertex v doesn't lead to a solution
then need to remove it
                path[position] = -1;
            }
        }

        // if no vertex can be added so far
        return false;
    }

    public int hamiltonianCycle(int[][] graph, int startingPoint) {
        path = new int[totalVertex];
        for (int i = 0; i < totalVertex; i++)
            path[i] = -1;

        path[0] = startingPoint;
        if (hamiltonianCycleUtility(graph, path, 1) == false) {
            System.out.println("\nNo path found. ");
            return 0;
        }

        displaySolution(path);
        return 1;
    }

    /** function to display the solution */
    public void displaySolution(int[] path) {
        System.out.println("\nPath found! ");
        for (int i = 0; i < totalVertex; i++)
            System.out.print(path[i] + " => ");

        // print first vertex again to show a complete cycle
        System.out.println(path[0] + " ");
    }
}

```

#### 4.4.2 Sample output

```
Enter starting point : 3
Path found!
3 => 2 => 11 => 13 => 14 => 15 => 9 => 12 => 5 => 7 => 0 => 1 => 4 => 6 => 8 => 10 => 3
```

Sample output for path finding if we choose node 3 as starting point.

```
Enter starting point : 5
Path found!
5 => 7 => 0 => 1 => 4 => 6 => 8 => 10 => 3 => 2 => 11 => 13 => 14 => 15 => 9 => 12 => 5
```

Sample output for path finding if we choose node 5 as starting point.

```
Enter starting point : 7
Path found!
7 => 0 => 1 => 4 => 6 => 8 => 10 => 3 => 2 => 11 => 13 => 14 => 15 => 9 => 12 => 5 => 7
```

Sample output for path finding if we choose node 7 as starting point.

```
Enter starting point : 9
Path found!
9 => 12 => 5 => 7 => 0 => 1 => 4 => 6 => 8 => 10 => 3 => 2 => 11 => 13 => 14 => 15 => 9
```

Sample output for path finding if we choose node 9 as starting point.

```
Enter starting point : 11
Path found!
11 => 2 => 3 => 10 => 8 => 6 => 4 => 1 => 0 => 7 => 5 => 12 => 9 => 15 => 14 => 13 => 11
```

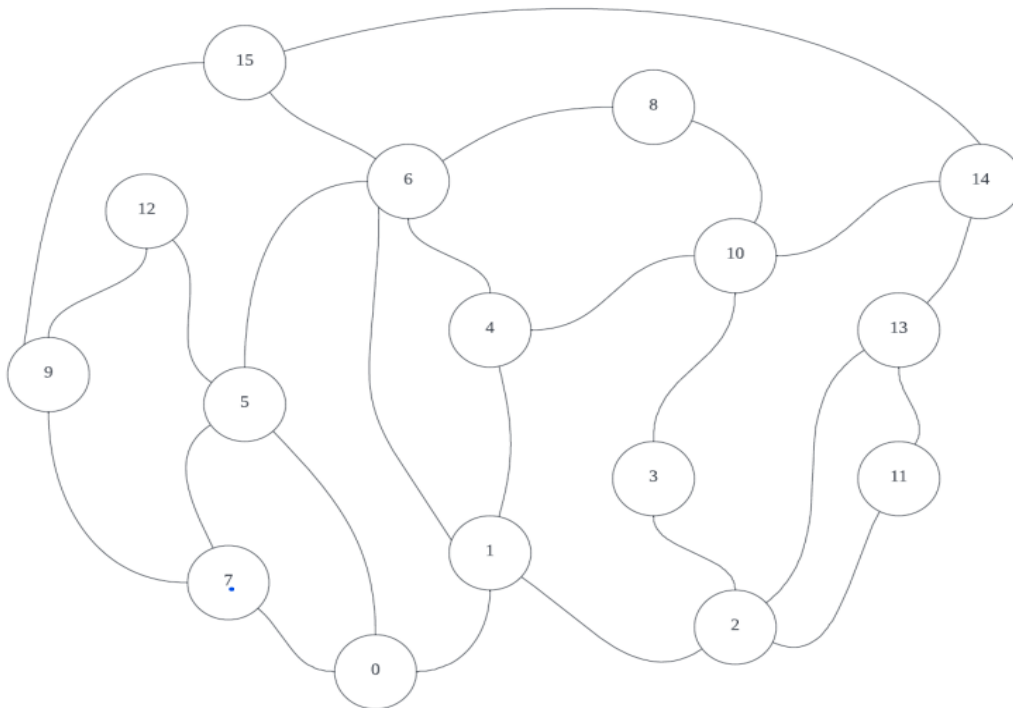
Sample output for path finding if we choose node 11 as starting point.

```
Enter starting point : 15
Path found!
15 => 9 => 12 => 5 => 7 => 0 => 1 => 4 => 6 => 8 => 10 => 3 => 2 => 11 => 13 => 14 => 15
```

Sample output for path finding if we choose node 15 as starting point.

#### 4.5 Best path to kill Titan

There are too many barriers in the town when the titan enters Maria's wall. Many of the soldiers die because they use the long path to kill the titan. This causes the gas of their movement machine to become empty and get killed by the titan. So, to increase the survival rate of the soldiers, Eren has provided the map of the barriers in the town and the position of the titan, which will randomly appear in all the corners of the city. Below is a map of the city.



The titan will be generated at random in the graph node, excluding the node start. Given that the gas of the movement machine is used constantly with the passage of time, and all the time used to go to another node using the movement machine. We need to implement an algorithm to find the shortest path to kill the Titan.



## 4.5.1 Source code

### 4.5.1.1 BestPathToKillTitan.java source code

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;
import java.util.Queue;

public class BestPathToKillTitan {
    // Function to form edge between
    // two vertices src and dest
    public void addEdge(ArrayList<ArrayList<Integer>> adj, int src,
int dest) {
        adj.get(src).add(dest);
        adj.get(dest).add(src);
    }

    // Function which finds all the paths
    // and stores it in paths array
    public static void find_paths(ArrayList<ArrayList<Integer>>
paths, ArrayList<Integer> path,
        ArrayList<ArrayList<Integer>> parent, int n, int u) {
        // Base Case
        if (u == -1) {
            paths.add(new ArrayList<>(path));
            return;
        }

        // Loop for all the parents
        // of the given vertex
        for (int par : parent.get(u)) {

            // Insert the current
            // vertex in path
            path.add(u);

            // Recursive call for its parent
            find_paths(paths, path, parent, n, par);

            // Remove the current vertex
            path.remove(path.size() - 1);
        }
    }

    // Function which performs bfs
```

```

// from the given source vertex
public static void bfs(ArrayList<ArrayList<Integer>> adj,
ArrayList<ArrayList<Integer>> parent,
int n, int start) {

    // dist will contain shortest distance
    // from start to every other vertex
    int[] dist = new int[n];
    Arrays.fill(dist, Integer.MAX_VALUE);

    Queue<Integer> q = new LinkedList<>();

    // Insert source vertex in queue and make
    // its parent -1 and distance 0
    q.offer(start);

    parent.get(start).clear();
    parent.get(start).add(-1);

    dist[start] = 0;

    // Until Queue is empty
    while (!q.isEmpty()) {
        int u = q.poll();

        for (int v : adj.get(u)) {
            if (dist[v] > dist[u] + 1) {

                // A shorter distance is found
                // So erase all the previous parents
                // and insert new parent u in parent[v]
                dist[v] = dist[u] + 1;
                q.offer(v);
                parent.get(v).clear();
                parent.get(v).add(u);
            } else if (dist[v] == dist[u] + 1) {

                // Another candidate parent for
                // shortest path found
                parent.get(v).add(u);
            }
        }
    }
}

// Function which prints all the paths
// from start to end
public static void print_paths(ArrayList<ArrayList<Integer>>
adj, int n, int start, int end) {

```

```

    ArrayList<ArrayList<Integer>> paths = new ArrayList<>();
    ArrayList<Integer> path = new ArrayList<>();
    ArrayList<ArrayList<Integer>> parent = new ArrayList<>();

    System.out.println("\nBest path: ");
    for (int i = 0; i < n; i++) {
        parent.add(new ArrayList<>());
    }

    // Function call to bfs
    bfs(adj, parent, n, start);

    // Function call to find_paths
    find_paths(paths, path, parent, n, end);

    for (ArrayList<Integer> v : paths) {

        // Since paths contain each
        // path in reverse order,
        // so reverse it
        Collections.reverse(v);

        // Print node for the current path
        for (int u : v)
            System.out.print(u + " ");

        System.out.println();
    }
}

```

#### 4.5.2 Sample output

```
Enter location of Titan: 6  
  
Best path:  
0 1 6  
0 5 6
```

Sample output for finding the best path if the Titan is located at node 6.

```
Enter location of Titan: 9  
  
Best path:  
0 7 9
```

Sample output for finding the best path if the Titan is located at node 9.

```
Enter location of Titan: 11  
  
Best path:  
0 1 2 11
```

Sample output for finding the best path if the Titan is located at node 11.

```
Enter location of Titan: 13  
  
Best path:  
0 1 2 13
```

Sample output for finding the best path if the Titan is located at node 13.

```
Enter location of Titan: 15  
  
Best path:  
0 1 6 15  
0 5 6 15  
0 7 9 15
```

Sample output for finding the best path if the Titan is located at node 13.

#### 4.6 Marley word converter

After that, we tried hard in the war to get back Maria Wall. Eren had finally returned to his home to open the door to the basement in his house. Eren successfully gets the information from his father, Grisha. But all the information is in Marley's sentences. However, Eren found a dictionary in the book rack that is a Marley translation-based dictionary. After Eren's study, he found that there was a pattern and relationship between the Paradis language and the Marley language. Eren hopes to achieve automated translation of the word to make sure he can get the word easily. So, Figure 2 below shows the character translation provided by Eren and Figure 3 provides the grammar symbols in Marley word.

a	b	c	d	e	f	g	h	i	j	k	l	m
j	c	t	a	k	z	s	i	w	x	o	n	g

n	o	p	q	r	s	t	u	v	w	x	y	z
b	f	h	l	d	e	y	m	v	u	p	q	r

Figure 2: Marley character to Paradis character

^	Character after word will be in big letters.
\$	This will be space in the Marley character.
,	This will be “,” in Marley character.
()	The word inside parentheses will be inverted.

Figure 3: Grammar in Marley word

## 4.6.1 Source code

### 4.6.1.1 CustomHashMap.java source code

```
public class CustomHashMap<K, V> {
    private int size = 0;
    private int capacity = 25;
    private Entry<K, V>[] entries = new Entry[capacity];
    private double loadFactor = 0.75;

    private static class Entry<K, V> {
        private final K key;
        private V value;
        private Entry<K, V> next;

        public Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }

    public boolean isEmpty() {
        return this.size == 0;
    }

    public int getSize() {
        return this.size;
    }

    public boolean containsKey(K key) {
        Entry<K, V> matchingEntry = getMatchingEntry(key);

        return matchingEntry != null && matchingEntry.key == key;
    }

    public boolean containsValue(V value) {
        for (Entry<K, V> entry : this.entries) {
            while (entry != null && !matches(value, entry.value)) {
                entry = entry.next;
            }

            if (entry != null) {
                return true;
            }
        }

        return false;
    }
}
```

```

public V get(K key) {
    Entry<K, V> matchingEntry = getMatchingEntry(key);

    return matchingEntry == null ? null : matchingEntry.value;
}

public void put(K key, V value) {
    if (this.shouldResize()) {
        this.resize();
    }

    if (addEntry(new Entry<>(key, value), this.entries)) {
        this.size++;
    }
}

public void remove(K key) {
    int index = indexOf(key);
    Entry<K, V> currentEntry = this.entries[index];

    while (currentEntry != null && currentEntry.next != null &&
!matches(key, currentEntry.next.key)) {
        currentEntry = currentEntry.next;
    }

    if (currentEntry != null) {
        // this case can only occur if there is only one
non-null entry at the index
        if (matches(key, currentEntry.key)) {
            this.entries[index] = null;
            // this case can only occur because the next
entry's key matched
        } else if (currentEntry.next != null) {
            currentEntry.next = currentEntry.next.next;
        }

        this.size--;
    }
}

private boolean shouldResize() {
    return this.size > Math.ceil((double) this.capacity *
this.loadFactor);
}

private void resize() {
    this.capacity = this.size * 2;
}

```

```

        Entry<K, V>[] newEntries = new Entry[this.capacity];
        for (Entry<K, V> entry : this.entries) {
            if (entry != null) {
                this.setEntry(entry, newEntries);
            }
        }

        this.entries = newEntries;
    }

    private void setEntry(Entry<K, V> entry, Entry<K, V>[] entries)
    {
        Entry<K, V> nextEntry = entry.next;
        entry.next = null;

        this.addEntry(entry, entries);

        if (nextEntry != null) {
            this.setEntry(nextEntry, entries);
        }
    }

    private boolean addEntry(Entry<K, V> entry, Entry<K, V>[]
entries) {
        int index = indexOf(entry.key);
        Entry<K, V> existingEntry = entries[index];

        if (existingEntry == null) {
            entries[index] = entry;
            return true;
        } else {
            while (!this.matches(entry.key, existingEntry.key) &&
existingEntry.next != null) {
                existingEntry = existingEntry.next;
            }

            if (this.matches(entry.key, existingEntry.key)) {
                existingEntry.value = entry.value;
                return false;
            }

            existingEntry.next = entry;
            return true;
        }
    }

    private Entry<K, V> getMatchingEntry(K key) {
        Entry<K, V> existingEntry = this.entries[indexOf(key)];
    }

```



```

        while (existingEntry != null && !matches(key,
existingEntry.key)) {
            existingEntry = existingEntry.next;
        }

        return existingEntry;
    }

    private int indexOf(K object) {
        return object == null ? 0 : hash(object) & (this.capacity -
1);
    }

    private boolean matches(Object o1, Object o2) {
        return (o1 == null && o2 == null) ||
            (o1 != null && o2 != null && o1.equals(o2));
    }

    /**
     * Applies a supplemental hash function to a given hashCode,
which
     * defends against poor quality hash functions. This is
critical
     * because HashMap uses power-of-two length hash tables, that
     * otherwise encounter collisions for hashCodes that do not
differ
     * in lower bits. Note: Null keys always map to hash 0, thus
index 0.
     */
    private static int hash(Object key) {
        // This function ensures that hashCodes that differ only by
        // constant multiples at each bit position have a bounded
        // number of collisions (approximately 8 at default load
factor).
        int h;
        return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>>
16);
    }
}

```

#### 4.6.2 Sample output

```
Enter Marley Sentence : oh(ldchc$ebdccd$rl)
find attack titan
```

```
Enter Marley Sentence : rsgc(qqd^i$tkz)$ko$^udzhd,(rld$sgk^z$)$^gpssld
destroy Wall of Maria, Rose and Sheena
```

```
Enter Marley Sentence : ^ukgc$rd(vsq$gh$zshrqkg$gwkzsml)h$dbeszudl
Most dangerous soldier is levi ackerman
```

#### 4.7 Protecting Wall of Maria

Captain Erwin captures information that the Armored Titan and Colossal Titan will try to break the Wall of Maria again. They also knew their enemy had obtained the wall building structure of Wall of Maria, and they will break the weakest part of the wall. We need to help Captain Erwin to find the weakest part of the wall so that he can assign soldiers to guard the part of the wall. Our Wall of Maria is as shown below.

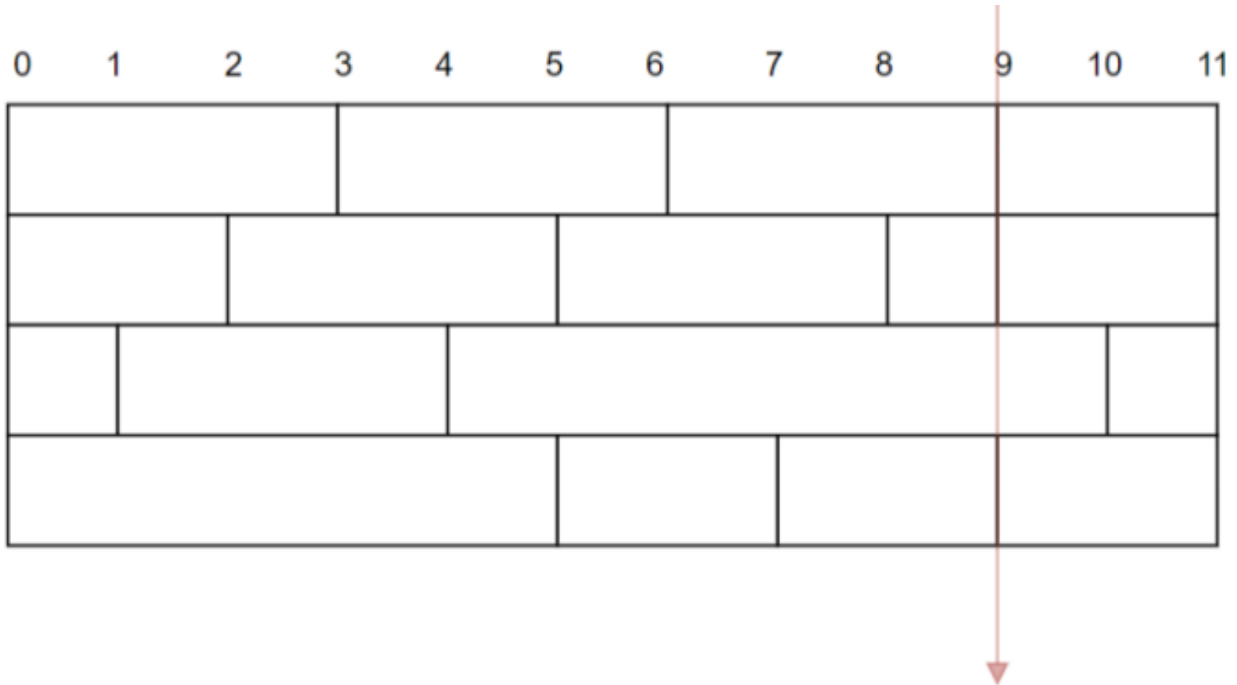


Figure 4: Wall of Maria

## 4.7.1 Source code

### 4.7.1.1 WeakestWallOfMaria.java source code

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Objects;

// Java program to find the most frequent element
// in an array

public class WeakestWallOfMaria {

    public static void main(String[] args) {
        System.out.println("\nWall of Maria has already been
generated. ");
        System.out.println("\nScanning weakest part of Wall of
Maria....\n");
        List<List<Integer>> layer = new ArrayList<>();
        List<Integer> brick1 = new ArrayList<>();
        List<Integer> brick2 = new ArrayList<>();
        List<Integer> brick3 = new ArrayList<>();
        List<Integer> brick4 = new ArrayList<>();
        ArrayList<Integer> arr = new ArrayList<>();
        brick1.add(3);
        brick1.add(6);
        brick1.add(9);
        layer.add(brick1);
        brick2.add(2);
        brick2.add(5);
        brick2.add(8);
        brick2.add(9);
        layer.add(brick2);
        brick3.add(1);
        brick3.add(4);
        brick3.add(10);
        layer.add(brick3);
        brick4.add(5);
        brick4.add(7);
        brick4.add(9);
        layer.add(brick4);
        arr = ConvertToOneList(layer);
        int test = mostFrequentArrayList(arr, arr.size());
        System.out.println("Weakest part of Wall of Maria is at
position " + test + "\n");
    }
}
```

```

        public static ArrayList<Integer>
ConvertToOneList(List<List<Integer>> data) {
    ArrayList<Integer> arr = new ArrayList<>();
    while (!data.isEmpty()) {
        List<Integer> layer = new ArrayList<>();
        layer = data.get(0);
        data.remove(0);
        while (!layer.isEmpty()) {
            arr.add(layer.get(0));
            layer.remove(0);
        }
    }
    return arr;
}

    public static int mostFrequentArrayList(ArrayList<Integer> arr,
int n) {
    // Sort the array
    Collections.sort(arr);
    // find the max frequency using linear traversal
    Integer max_count = 1;
    Integer res = arr.get(0);
    int curr_count = 1;

    for (int i = 1; i < n; i++) {
        if (Objects.equals(arr.get(i), arr.get(i - 1))) {
            curr_count++;
        } else {
            curr_count = 1;
        }

        if (curr_count > max_count) {
            max_count = curr_count;
            res = arr.get(i - 1);
        }
    }
    return res;
}
}

```

#### 4.7.2 Sample output

```
Wall of Maria has already been generated.  
Scanning weakest part of Wall of Maria....  
Weakest part of Wall of Maria is at position 9
```

Sample output for finding the weakest part for Wall of Maria.

## 4.8 Main.java source code

This source code is to run all the source code given above.

```
import SoldierArrangement.Characters;
import SoldierArrangement.DoublyLinkedList;
//import SoldierArrangement.Node;
import SoldierArrangement.SortList;
import Titan.NormalTitan;
//import Titan.PriorityQueue;
import Titan.Titan;
import Titan.TitanTag;
import ScoutingMissionInsideWall.ScoutingClass;
import BestPathToKillTitan.BestPathToKillTitan;
import MarleyWordConverter.CustomHashMap;
//import WallOfMaria.WeakestWallOfMaria;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Objects;
import java.util.Random;
import java.util.Scanner;
//import java.util.Stack;

public class Main {

    public static void main(String[] args) throws IOException {
        DoublyLinkedList<String> tmp = new
DoublyLinkedList<String>();
        DoublyLinkedList<Characters> Allies = new
DoublyLinkedList<>();
        SortList<Characters> Sort = new SortList<>();
        DoublyLinkedList<Characters> CheckList = new
DoublyLinkedList<>();
        System.out.println();
        System.out.println("\033[3m- - - - - TOPIC D : WORLD OF
TITAN - - - - - \033[0m");
        System.out.println("\n\n\033[3mWelcome\033[0m");
        System.out.println("\n\n\033[3mEntering the first part
in...\033[0m");
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
```

```

        e.printStackTrace();
    }
    System.out.println("3..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("2..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("1..");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

//
System.out.println("\n\n\n*****");
*****");
    System.out.println("\n");
    FileWriter out = new FileWriter("Eren's Allies.txt", true);
    PrintWriter outFile = new PrintWriter(out);
    try (Scanner sc = new Scanner(System.in)) {
        System.out.println(
            "\033[3m~ ~ ~ ~ ~ PART 1 : Eren's Allies,
Soldier Arrangement & Grouping ~ ~ ~ ~ ~ \033[0m\n");
        while (true) {
            System.out.println("\nDo you want to enter Eren's
allies? [ yes / no ] ");
            String input = sc.nextLine();

            if (input.equalsIgnoreCase("yes") ||
input.equalsIgnoreCase("y")) {
                System.out.println("\nName (name and surname):
");
                String name = sc.nextLine();

                System.out.println(
                    "Enter Character (height, weight,
strength, agility, intelligence, coordination, leadership) : ");
                String input1 = sc.nextLine();
                int i = 0;
                // to split the Character and store into
listchar

```



```

        for (String s : input1.split(" ")) {

            i++;
            tmp.add(i, s);

        }

        // listchar.iterateForward();
        System.out.println("");
        // remove the Character in listchar and assign
to the string Character
        String height = tmp.removeFirst();
        String weight = tmp.removeFirst();
        String strength = tmp.removeFirst();
        String agility = tmp.removeFirst();
        String intel = tmp.removeFirst();
        String coor = tmp.removeFirst();
        String lead = tmp.removeFirst();

        outFile.print(
            name + " " + height + " " + weight + "
" + strength + " " + agility + " " + intel + " "
            + coor + " " + lead);
        outFile.println();
        outFile.close();
        continue;
    } else if (input.equalsIgnoreCase("no") ||
input.equalsIgnoreCase("n")) {
        break;
    } else {
        System.out.println("Incorrect input");
    }

}
// list.clear();

System.out.println("");
System.out.println("\n\033[3m- - - - - List Of
Soldiers Available - - - - -\033[0m");
try {

    // File myObj = new File("filename.txt");
    FileReader fr = new FileReader("Eren's
Allies.txt");

    Scanner myReader = new Scanner(fr);
    while (myReader.hasNextLine()) {
        String data = null;
        data = myReader.nextLine();
        int i = 0;

```

```

        tmp.clear();
        // System.out.println(data);

        for (String s : data.split(" ")) {

            tmp.add(i, s);
            i++;

        }

        String name = tmp.removeFirst() + " " +
tmp.removeFirst();

        String height = tmp.removeFirst();
        String weight = tmp.removeFirst();
        String strength = tmp.removeFirst();
        String agility = tmp.removeFirst();
        String intel = tmp.removeFirst();
        String coor = tmp.removeFirst();
        String lead = tmp.removeFirst();
        Characters character = new Characters(name,
height, weight, strength, agility, intel, coor, lead);
        Allies.addLast(character);

    }
    myReader.close();
} catch (FileNotFoundException e) {
    System.out.println("An error occurred. ");
    e.printStackTrace();
}

while (true) {

    CheckList.clear();
    Sort.clear();

    System.out.println("");
    System.out.print("Do you want to sort attribute ? :
");

    String input1 = sc.nextLine();
        if (input1.equalsIgnoreCase("yes") ||
input1.equalsIgnoreCase("y")) {
        System.out.println(
            "Available attributes : height, weight,
strength, agility, intelligence, coordination, leadership");
        System.out.print("\nSorting attribute: ");
        String attribute = sc.nextLine();
        System.out.println("\nSorting from highest to
lowest : \n");

```

```

        switch (attribute) {
            case "height":
                for (int k = 0; k < Allies.size(); k++)
                {
                    int height =
Integer.parseInt(Allies.show(k).getHeight());
                    Sort.addNode(height);
                }
                Sort.sortList();
                // CList.display();
                for (int i = 0; i < Allies.size(); i++)
                {
                    for (int j = 0; j < Allies.size();
j++) {
                        if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getHeight())) {
                            if
(!CheckList.contains(Allies.show(j))) {
                                System.out.println(Allies.show(j).getName() + " " + Sort.show(i));
                                CheckList.addFirst(Allies.show(j));
                            }
                        }
                    }
                }
                break;
            case "weight":
                for (int k = 0; k < Allies.size(); k++)
                {
                    int height =
Integer.parseInt(Allies.show(k).getWeight());
                    Sort.addNode(height);
                }
                Sort.sortList();
                // CList.display();
                for (int i = 0; i < Allies.size(); i++)
                {
                    for (int j = 0; j < Allies.size();
j++) {
                        if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getWeight())) {
                            // String printed = "";
                            //
System.out.println(list.show(j));

```

```

                                                                    if
(!CheckList.contains(Allies.show(j))) {

System.out.println(Allies.show(j).getName() + " " + Sort.show(i));

CheckList.addFirst(Allies.show(j));

    }

    }

    }

    }

        break;
        case "strength":
            for (int k = 0; k < Allies.size(); k++)
{
                                                                    int height =
Integer.parseInt(Allies.show(k).getStrength());
                Sort.addNode(height);
            }
            Sort.sortList();
            // CList.display();
            for (int i = 0; i < Allies.size(); i++)
{
                for (int j = 0; j < Allies.size();
j++) {
                                                                    if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getStrength())) {
                                                                    // String printed = "";
                                                                    //
System.out.println(list.show(j));
                                                                    if
(!CheckList.contains(Allies.show(j))) {

System.out.println(Allies.show(j).getName() + " " + Sort.show(i));

CheckList.addFirst(Allies.show(j));

    }

    }

    }

    }

        break;
        case "agility":
            for (int k = 0; k < Allies.size(); k++)

```

```

{
    int height =
Integer.parseInt(Allies.show(k).getAgility());
    Sort.addNode(height);
}
Sort.sortList();
// CList.display();
for (int i = 0; i < Allies.size(); i++)
{
    for (int j = 0; j < Allies.size();
j++) {
        if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getAgility())) {
            // String printed = "";
            //
System.out.println(list.show(j));
            if
(!CheckList.contains(Allies.show(j))) {
System.out.println(Allies.show(j).getName() + " " + Sort.show(i));
CheckList.addFirst(Allies.show(j));
            }
        }
    }
}
break;
case "intelligence":
    for (int k = 0; k < Allies.size(); k++)
{
    int height =
Integer.parseInt(Allies.show(k).getIntel());
    Sort.addNode(height);
}
Sort.sortList();
// CList.display();
for (int i = 0; i < Allies.size(); i++)
{
    for (int j = 0; j < Allies.size();
j++) {
        if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getIntel())) {
            // String printed = "";
            //
System.out.println(list.show(j));

```

```

                                                                    if
(!CheckList.contains(Allies.show(j))) {

System.out.println(Allies.show(j).getName() + " " + Sort.show(i));

CheckList.addFirst(Allies.show(j));

    }

    }

    }

    }

        break;
        case "coordination":
            for (int k = 0; k < Allies.size(); k++)
{
                                                                    int height =
Integer.parseInt(Allies.show(k).getCoor());
                Sort.addNode(height);
            }
            Sort.sortList();
            // CList.display();
            for (int i = 0; i < Allies.size(); i++)
{
                for (int j = 0; j < Allies.size();
j++) {
                                                                    if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getCoor())) {
                                                                    // String printed = "";
                                                                    //
System.out.println(list.show(j));
                                                                    if
(!CheckList.contains(Allies.show(j))) {

System.out.println(Allies.show(j).getName() + " " + Sort.show(i));

CheckList.addFirst(Allies.show(j));

    }

    }

    }

    }

        break;
        case "leadership":

```

```

        for (int k = 0; k < Allies.size(); k++)
        {
            int height =
Integer.parseInt(Allies.show(k).getLead());
            Sort.addNode(height);
        }
        Sort.sortList();
        // CList.display();
        for (int i = 0; i < Allies.size(); i++)
        {
            for (int j = 0; j < Allies.size();
j++) {
                if (Sort.show(i) ==
Integer.parseInt(Allies.show(j).getLead())) {
                    // String printed = "";
                    //
System.out.println(list.show(j));
                    if
(!CheckList.contains(Allies.show(j))) {
                        System.out.println(Allies.show(j).getName() + " " + Sort.show(i));
                        CheckList.addFirst(Allies.show(j));
                    }
                }
            }
        }
        break;
        default:
            System.out.print("Attribute not
found.");
    }

    // System.out.println("");
    continue;
    } else if (input1.equalsIgnoreCase("no") ||
input1.equalsIgnoreCase("n")) {
        break;
    } else {
        System.out.println("Incorrect input");
    }
}

// System.out.print("Do you want to search for ability?

```

```

: ");
    // String input1 = sc.nextLine();
    while (true) {
        CheckList.clear();
        System.out.println("");
        System.out.print("Do you want to search for ability
? : ");

        String input1 = sc.nextLine();
        if (input1.equalsIgnoreCase("yes") ||
input1.equalsIgnoreCase("y")) {
            System.out.println(
                "Available ability : height, weight,
strength, agility, intelligence, coordination, leadership");
            System.out.print("\nFinding ability : ");
            String ability = sc.nextLine();
            System.out.print("Value : ");
            String valueAbility = sc.nextLine();
            System.out.println("");

            switch (ability) {
                case "height":
                    System.out.println("Soldier : ");
                    for (int i = 0; i < Allies.size(); i++)
{
                                                                if
(Allies.show(i).getHeight().equalsIgnoreCase(valueAbility)) {
System.out.print(Allies.show(i).getName() + ", ");
CheckList.addFirst(Allies.show(i));
                    }
                }
                if (CheckList.isEmpty()) {
                    System.out.println("No soldiers
found :( ");
                }
                break;
                case "weight":
                    System.out.println("Soldier : ");
                    for (int i = 0; i < Allies.size(); i++)
{
                                                                if
(Allies.show(i).getWeight().equalsIgnoreCase(valueAbility)) {
System.out.print(Allies.show(i).getName() + ", ");
CheckList.addFirst(Allies.show(i));
                    }
                }
            }
        }
    }
}

```



```

        if (CheckList.isEmpty()) {
            System.out.println("No soldiers
found :( ");
        }
        break;
        case "strength":
            System.out.println("Soldier : ");
            for (int i = 0; i < Allies.size(); i++)
            {
                if
                (Allies.show(i).getStrength().equalsIgnoreCase(valueAbility)) {
                    System.out.print(Allies.show(i).getName() + ", ");
                    CheckList.addFirst(Allies.show(i));
                }
            }
            if (CheckList.isEmpty()) {
                System.out.println("No soldiers
found :( ");
            }
            break;
            case "agility":
                System.out.println("soldier : ");
                for (int i = 0; i < Allies.size(); i++)
                {
                    if
                    (Allies.show(i).getAgility().equalsIgnoreCase(valueAbility)) {
                        System.out.print(Allies.show(i).getName() + ", ");
                        CheckList.addFirst(Allies.show(i));
                    }
                }
                if (CheckList.isEmpty()) {
                    System.out.println("No soldiers
found :( ");
                }
                break;
                case "intelligence":
                    System.out.println("soldier : ");
                    for (int i = 0; i < Allies.size(); i++)
                    {
                        if
                        (Allies.show(i).getIntel().equalsIgnoreCase(valueAbility)) {
                            System.out.print(Allies.show(i).getName() + ", ");
                            CheckList.addFirst(Allies.show(i));
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    if (CheckList.isEmpty()) {
        System.out.println("No soldiers
found :( ");
    }
    break;
    case "coordination":
        System.out.println("soldier : ");
        for (int i = 0; i < Allies.size(); i++)
        {
            if
            (Allies.show(i).getCoor().equalsIgnoreCase(valueAbility)) {

                System.out.print(Allies.show(i).getName() + ", ");

                CheckList.addFirst(Allies.show(i));
            }
        }
        if (CheckList.isEmpty()) {
            System.out.println("No soldiers
found :( ");
        }
        break;
        case "leadership":
            System.out.println("soldier : ");
            for (int i = 0; i < Allies.size(); i++)
            {
                if
                (Allies.show(i).getLead().equalsIgnoreCase(valueAbility)) {

                    System.out.print(Allies.show(i).getName() + ", ");

                    CheckList.addFirst(Allies.show(i));
                }
            }
            if (CheckList.isEmpty()) {
                System.out.println("No soldiers
found :( ");
            }
            break;
            default:
                System.out.print("Ability not found.");
        }
        System.out.println("");
        continue;
    } else if (input1.equalsIgnoreCase("no") ||
input1.equalsIgnoreCase("n")) {
        break;
    }
}

```

```

        } else {
            System.out.println("Incorrect input. ");
        }
    }

    System.out.println("\n\n~ ~ ~ ~ ~ END OF PART 1 ~ ~
~ ~ ~ ~ ~");
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("\n");

    System.out.println("\033[3mMoving to part
2.\033[0m\n\n");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Press [ Enter ] key to continue.");
    sc.nextLine();
    System.out.print("[ Enter ] key pressed. \n");
    System.out.println("\nMoving to the next part in...");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("3..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("2..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("1..");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

```

```

        System.out.println(
            "\n\n\033[3m- - - - - PART 2 : Titans
Evaluation & Killing Priority - - - - -\033[0m");
        System.out.println("\n");

        DoublyLinkedList<TitanTag> CheckList2 = new
DoublyLinkedList<>();
        NormalTitan a = new NormalTitan();
        Random t = new Random();
        int num = t.nextInt(10) + 1;
        System.out.println("Number of Titans: " + num);
        System.out.println();
        System.out.println("Generating " + num + " Titans
....");
        int risk = 0;
        java.util.PriorityQueue<Integer> q = new
java.util.PriorityQueue<>();
        for (int d = 1; d <= num; d++) {
            Titan s = new Titan();
            String[] o = { "Normal Titan", "Abnormal Titan",
"Nine Titan" };
            Random r = new Random();
            int randomNumber = r.nextInt(o.length);
            // to find normal Titan's risk with specific
characteristic
            if (randomNumber == 0) {
                if (a.Height() > 20) { // 3
                    if (a.WalkingLegs() == 4) { // 3
                        if (a.Speed() > 20) { // 3
                            if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                    risk = 15; // 3+3+3+3+3
                                    q.offer(risk);
                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                    risk = 13; // 3+3+3+3+1
                                    q.offer(risk);
                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +

```

```

a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 14; // 3+3+3+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 12; // 3+3+3+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 13; // 3+3+3+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 11; // 3+3+3+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
} else if (a.Speed() > 10) { // 2
    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3

```

```

                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 14; // 3+3+2+3+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 12; // 3+3+2+3+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 13; // 3+3+2+2+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 11; // 3+3+2+2+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    } else { // 1
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 12; // 3+3+2+1+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "

```

```

+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 10; // 3+3+2+1+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
}
} else { // 1
    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 13; // 3+3+1+3+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 11; // 3+3+1+3+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 12; // 3+3+1+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +

```

```

a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 10; // 3+3+1+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
        } else { // 1
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 3+3+1+1+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 3+3+1+1+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    }
    } else if (a.WalkingLegs() == 2) { // 2
        if (a.Speed() > 20) { // 3
            if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                    risk = 14; // 3+2+3+3+3
                    q.offer(risk);
                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                } else {

```



```

risk = 12; // 3+2+3+3+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 13; // 3+2+3+2+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 11; // 3+2+3+2+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 12; // 3+2+3+1+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 10; // 3+2+3+1+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()

```

```

+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if (a.Speed() > 10) { // 2
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 13; // 3+2+2+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 11; // 3+2+2+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 3+2+2+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 3+2+2+2+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    }
}

```

```

        } else { // 1
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 3+2+2+1+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 3+2+2+1+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    } else { // 1
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 3+2+1+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 3+2+1+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {

```

```

risk = 11; // 3+2+1+2+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 9; // 3+2+1+2+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 10; // 3+2+1+1+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 9; // 3+2+1+1+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
}
} else { // 1
if (a.Speed() > 20) { // 3
if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 13; // 3+1+3+3+3
q.offer(risk);

```

```

        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 11; // 3+1+3+3+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 12; // 3+1+3+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 10; // 3+1+3+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 11; // 3+1+3+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);

```

```

        } else {
            risk = 9; // 3+1+3+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
    } else if (a.Speed() > 10) { // 2
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 3+1+2+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 3+1+2+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 3+1+2+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 3+1+2+2+1
                q.offer(risk);
            }
        }
    }
}

```

```

        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 3+1+2+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
            + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
            + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 8; // 3+1+2+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
            + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
            + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
    } else { // 1
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 3+1+1+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 3+1+1+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()

```

```

+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 3+1+1+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 8; // 3+1+1+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 9; // 3+1+1+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 7; // 3+1+1+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
}
}
}

```



```

        } else if (a.Height() > 10) { // 2
            if (a.WalkingLegs() == 4) { // 3
                if (a.Speed() > 20) { // 3
                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                            risk = 14; // 2+3+3+3+3
                            q.offer(risk);
                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                        } else {
                            risk = 12; // 2+3+3+3+1
                            q.offer(risk);
                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                        }
                    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                            risk = 13; // 2+3+3+2+3
                            q.offer(risk);
                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                        } else {
                            risk = 11; // 2+3+3+2+1
                            q.offer(risk);
                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                        }
                    } else { // 1

```

```

                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 12; // 2+3+3+1+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 10; // 2+3+3+1+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    }
                                                                    } else if (a.Speed() > 10) { // 2
                                                                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 13; // 2+3+2+3+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 11; // 2+3+2+3+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 12; // 2+3+2+2+3

```

```

                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                    risk = 10; // 2+3+2+2+1
                                    q.offer(risk);
                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else { // 1
                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                        risk = 11; // 2+3+2+1+3
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                    } else {
                                        risk = 9; // 2+3+2+1+1
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                    }
                                }
                                } else { // 1
                                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 12; // 2+3+1+3+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                            + a.WalkingLegs() +

```

```

" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                    risk = 10; // 2+3+1+3+1
                                    q.offer(risk);
                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                        risk = 11; // 2+3+1+2+3
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 9; // 2+3+1+2+1
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        }
                                    } else { // 1
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 10; // 2+3+1+1+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 8; // 2+3+1+1+1
                                            q.offer(risk);

```

```

        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
}
}
} else if (a.WalkingLegs() == 2) { // 2
    if (a.Speed() > 20) { // 3
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 13; // 2+2+3+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 11; // 2+2+3+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 2+2+3+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 2+2+3+2+1
                q.offer(risk);
                System.out.println("Titan "

```

```

+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else { // 1
                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                        risk = 11; // 2+2+3+1+3
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                    } else {
                                        risk = 9; // 2+2+3+1+1
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                    }
                                }
                                } else if (a.Speed() > 10) { // 2
                                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 12; // 2+2+2+3+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 10; // 2+2+2+3+1
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +

```

```

a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 11; // 2+2+2+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 9; // 2+2+2+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 2+2+2+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 8; // 2+2+2+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
} else { // 1
    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3

```

```

                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 11; // 2+2+1+3+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 9; // 2+2+1+3+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 10; // 2+2+1+2+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    } else {
                                                                    risk = 8; // 2+2+1+2+1
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                                    }
                                                                    } else { // 1
                                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                                    risk = 9; // 2+2+1+1+3
                                                                    q.offer(risk);
                                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "

```



```

+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 7; // 2+2+1+1+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
}
}
} else { // 1
    if (a.Speed() > 20) { // 3
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 2+1+3+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 2+1+3+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 2+1+3+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +

```

```

" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                    risk = 9; // 2+1+3+2+1
                                    q.offer(risk);
                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else { // 1
                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                        risk = 10; // 2+1+3+1+3
                                        q.offer(risk);
                                        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 8; // 2+1+3+1+1
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        }
                                    }
                                } else if (a.Speed() > 10) { // 2
                                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 11; // 2+1+2+3+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {

```

```

risk = 9; // 2+1+2+3+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 10; // 2+1+2+2+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 8; // 2+1+2+2+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 9; // 2+1+2+1+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 7; // 2+1+2+1+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()

```

```

+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else { // 1
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 10; // 2+1+1+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 8; // 2+1+1+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 9; // 2+1+1+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 7; // 2+1+1+2+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    }
}

```

```

        } else { // 1
                                                    if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                                    risk = 8; // 2+1+1+1+3
                                                    q.offer(risk);
                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                    } else {
                                                    risk = 6; // 2+1+1+1+1
                                                    q.offer(risk);
                                                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                                    }
        }
    }
} else { // 1
    if (a.WalkingLegs() == 4) { // 3
        if (a.Speed() > 20) { // 3
            if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                    risk = 13; // 1+3+3+3+3
                    q.offer(risk);
                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                } else {
                    risk = 11; // 1+3+3+3+1
                    q.offer(risk);
                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                }
            }
        }
    }
}

```

```

        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 12; // 1+3+3+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 10; // 1+3+3+2+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else { // 1
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 1+3+3+1+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 1+3+3+1+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    } else if (a.Speed() > 10) { // 2
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {

```

```

risk = 12; // 1+3+2+3+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 10; // 1+3+2+3+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 11; // 1+3+2+2+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 9; // 1+3+2+2+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 10; // 1+3+2+1+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()

```

```

+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 8; // 1+3+2+1+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
}
} else { // 1
    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 11; // 1+3+1+3+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 9; // 1+3+1+3+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 1+3+1+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {

```



```

risk = 8; // 1+3+1+2+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 9; // 1+3+1+1+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 7; // 1+3+1+1+1
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
}
} else if (a.WalkingLegs() == 2) { // 2
if (a.Speed() > 20) { // 3
if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
risk = 12; // 1+2+3+3+3
q.offer(risk);
System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
risk = 10; // 1+2+3+3+1
q.offer(risk);

```

```

                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                risk = 11; // 1+2+3+2+3
                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                risk = 9; // 1+2+3+2+1
                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                } else { // 1
                                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                risk = 10; // 1+2+3+1+3
                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                } else {
                                risk = 8; // 1+2+3+1+1
                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);

```

```

    }
    }
    } else if (a.Speed() > 10) { // 2
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 11; // 1+2+2+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 9; // 1+2+2+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 10; // 1+2+2+2+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 8; // 1+2+2+2+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        } else { // 1
            if ("Can

```

```

climb".equalsIgnoreCase(a.ClimbingSkill())) {
    risk = 9; // 1+2+2+1+3
    q.offer(risk);
    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
} else {
    risk = 7; // 1+2+2+1+1
    q.offer(risk);
    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
}
} else { // 1
    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 1+2+1+3+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 8; // 1+2+1+3+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
    + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
    + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 9; // 1+2+1+2+3
            q.offer(risk);

```

```

        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 7; // 1+2+1+2+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
        + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
        + ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 8; // 1+2+1+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
            + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
            + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 6; // 1+2+1+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
            + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
            + ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
    }
    } else { // 1
        if (a.Speed() > 20) { // 3
            if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                    risk = 11; // 1+1+3+3+3
                    q.offer(risk);
                    System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "

```

```

+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    } else {
        risk = 9; // 1+1+3+3+1
        q.offer(risk);
        System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 10; // 1+1+3+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 8; // 1+1+3+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 9; // 1+1+3+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 7; // 1+1+3+1+1

```

```

                                q.offer(risk);
                                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                }
                                }
                                } else if (a.Speed() > 10) { // 2
                                    if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 10; // 1+1+2+3+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 8; // 1+1+2+3+1
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        }
                                    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
                                        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                                            risk = 9; // 1+1+2+2+3
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
                                                + a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
                                                + ", " +
a.ClimbingSkill() + ") Risk " + risk);
                                        } else {
                                            risk = 7; // 1+1+2+2+1
                                            q.offer(risk);
                                            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "

```

```

+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
    }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 8; // 1+1+2+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 6; // 1+1+2+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
    } else { // 1
        if ("Not repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 3
            if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
                risk = 9; // 1+1+1+3+3
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            } else {
                risk = 7; // 1+1+1+3+1
                q.offer(risk);
                System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
            }
        }
    }
}

```



```

    }
    } else if ("Repeated
pattern".equalsIgnoreCase(a.WalkingPattern())) { // 2
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 8; // 1+1+1+2+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 6; // 1+1+1+2+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    } else { // 1
        if ("Can
climb".equalsIgnoreCase(a.ClimbingSkill())) {
            risk = 7; // 1+1+1+1+3
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        } else {
            risk = 5; // 1+1+1+1+1
            q.offer(risk);
            System.out.println("Titan "
+ d + ": Normal Titan (" + a.Height() + "m, "
+ a.WalkingLegs() +
" legs, " + a.Speed() + "ms, " + a.WalkingPattern()
+ ", " +
a.ClimbingSkill() + ") Risk " + risk);
        }
    }
}
}
}
}
}

```

```

        } else if (randomNumber == 1) {
            System.out.println("Titan " + d + ": Abnormal
Titan Risk 15");
            risk = 15;
            q.offer(s.AbnormalTitan());
        } else {
            Random k = new Random();
            int i = k.nextInt(7);
            s.NineTitan();
            q.offer(s.NineTitan());
            switch (i) {
                case 0:
                    System.out.println("Titan " + d + ":
Nine Titan (Beast Titan) Risk 19");
                    break;
                case 1:
                    System.out.println("Titan " + d + ":
Nine Titan (Jaw Titan) Risk 19");
                    break;
                case 2:
                    System.out.println("Titan " + d + ":
Nine Titan (Collosal Titan) Risk 19");
                    break;
                case 3:
                    System.out.println("Titan " + d + ":
Nine Titan (Armoured Titan) Risk 19");
                    break;
                case 4:
                    System.out.println("Titan " + d + ":
Nine Titan (War Hammer Titan) Risk 19");
                    break;
                case 5:
                    System.out.println("Titan " + d + ":
Nine Titan (Cart Titan) Risk 19");
                    break;
                case 6:
                    System.out.println("Titan " + d + ":
Nine Titan (Female Titan) Risk 19");
                    break;
            }
            risk = 19;
        }
        TitanTag ti = new TitanTag(d, risk);
        CheckList2.addLast(ti);
    }
    System.out.print("\nSequence to be killed : ");
    // while(!CheckList.isEmpty())
    //
    System.out.println(CheckList.removeFirst().toString());

```

```

while (!q.isEmpty()) {
    Integer r = q.poll();
    for (int i = 0; i < CheckList2.size(); i++) {
        // Integer r =q.poll();
        if (r.equals(CheckList2.show(i).getRisk())) {
            System.out.print(CheckList2.remove(i));
            if (!CheckList2.isEmpty()) {
                System.out.print(" --> ");
            } else {
                System.out.println("");
            }
            // continue;
        }
    }
}

// while(!q.isEmpty())
// System.out.println(q.poll());
System.out.println("\n\n~ ~ ~ ~ ~ End Of Part 2 ~ ~
~ ~ ~ ~ ~\n");
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("\n");

System.out.println("\033[3mMoving to part
3.\033[0m\n\n");
try {
    Thread.sleep(2000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("Press [ Enter ] key to continue.");
sc.nextLine();
System.out.print("[ Enter ] key pressed. \n");
System.out.println("\nMoving to the next part in...");
try {
    Thread.sleep(1500);
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("3..");
try {
    Thread.sleep(1500);
} catch (InterruptedException e) {
    e.printStackTrace();
}

```

```

        System.out.println("2..");
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("1..");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out
            .println("\n\n\033[3m- - - - - PART 3 :
Scouting Mission Inside The Wall - - - - -\033[0m");
        System.out.println("\n");

        System.out.print("Enter starting point : ");

        int startingPoint = sc.nextInt();

        ScoutingClass scoutClass = new ScoutingClass();

        int[][] paradisMap = {
0 },
            { 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0 },
            { 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0 },
            { 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0 },
            { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0 },
            { 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0 },
            { 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
0 },
            { 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1 },
            { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0 },
            { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0 },
            { 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1 },
            { 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0 },
            { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0 },

```

```

        { 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0 },
        { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
1 },
        { 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0 }

    };

    scoutClass.hamiltonianCycle(paradisMap, startingPoint);

    System.out.println("\n\n~ ~ ~ ~ ~ End Of Part 3 ~ ~
~ ~ ~ ~ ~\n");
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("\n");

    System.out.println("\033[3mMoving to part
4.\033[0m\n\n");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Press [ Enter ] key to continue.");
    sc.nextLine();
    System.out.print("[ Enter ] key pressed. \n");
    System.out.println("\nMoving to the next part in...");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("3..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("2..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

```

        System.out.println("1..");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\n\n\033[3m- - - - - PART 4 :
Best Path To Kill Titan - - - - - \033[0m");
        System.out.println("\n");

        System.out.print("Enter location of Titan: ");
        int dest = sc.nextInt();

        int n = 16;

        ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            adj.add(new ArrayList<>());
        }

        BestPathToKillTitan bestPath = new
BestPathToKillTitan();

        bestPath.addEdge(adj, 0, 1);
        bestPath.addEdge(adj, 0, 5);
        bestPath.addEdge(adj, 0, 7);
        bestPath.addEdge(adj, 1, 2);
        bestPath.addEdge(adj, 1, 4);
        bestPath.addEdge(adj, 1, 6);
        bestPath.addEdge(adj, 2, 3);
        bestPath.addEdge(adj, 2, 11);
        bestPath.addEdge(adj, 2, 13);
        bestPath.addEdge(adj, 3, 10);
        bestPath.addEdge(adj, 4, 6);
        bestPath.addEdge(adj, 4, 10);
        bestPath.addEdge(adj, 5, 6);
        bestPath.addEdge(adj, 5, 7);
        bestPath.addEdge(adj, 5, 12);
        bestPath.addEdge(adj, 6, 8);
        bestPath.addEdge(adj, 6, 15);
        bestPath.addEdge(adj, 7, 9);
        bestPath.addEdge(adj, 8, 10);
        bestPath.addEdge(adj, 9, 12);
        bestPath.addEdge(adj, 9, 15);
        bestPath.addEdge(adj, 10, 14);
        bestPath.addEdge(adj, 11, 13);
        bestPath.addEdge(adj, 13, 14);
        bestPath.addEdge(adj, 14, 15);

```

```

        int src = 0;

        bestPath.print_paths(adj, n, src, dest);

        System.out.println("\n\n~ ~ ~ ~ ~ End Of Part 4 ~ ~
~ ~ ~ ~ ~\n");
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("\n");

        System.out.println("\033[3mMoving to part
5.\033[0m\n\n");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Press [ Enter ] key to continue.");
        sc.nextLine();
        System.out.print("[ Enter ] key pressed. \n");
        System.out.println("\nMoving to the next part in...");
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("3..");
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("2..");
        try {
            Thread.sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("1..");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

```

```

        System.out.println("\n\n\033[3m- - - - - PART 5 :
Marley Word Converter - - - - - \033[0m");
        System.out.println("\n");

        System.out.print("Enter Marley sentence: \t");
        String marleySc = sc.next();

        String temp = " ";

        CustomHashMap<Character, Character> marley = new
CustomHashMap<>();

        marley.put('a', 'j');
        marley.put('b', 'c');
        marley.put('c', 't');
        marley.put('d', 'a');
        marley.put('e', 'k');
        marley.put('f', 'z');
        marley.put('g', 's');
        marley.put('h', 'i');
        marley.put('i', 'w');
        marley.put('j', 'x');
        marley.put('k', 'o');
        marley.put('l', 'n');
        marley.put('m', 'g');
        marley.put('n', 'b');
        marley.put('o', 'f');
        marley.put('p', 'h');
        marley.put('q', 'l');
        marley.put('r', 'd');
        marley.put('s', 'e');
        marley.put('t', 'y');
        marley.put('u', 'm');
        marley.put('v', 'v');
        marley.put('w', 'u');
        marley.put('x', 'p');
        marley.put('y', 'q');
        marley.put('z', 'r');
        marley.put('$', ' ');
        marley.put(',', ',');
        marley.put('^', '^');
        marley.put(')', ')');
        marley.put('(', '(');
        for (int j = 0; j < marleySc.length(); j++) {
            char translated = marley.get(marleySc.charAt(j));
            temp += translated;
        }
        String jadi = marley.Caps(temp, temp.length());
        String out1 = marley.reverseParentheses(jadi,

```



```

jadi.length());
    System.out.println();
    System.out.println(out1);

    System.out.println("\n\n~ ~ ~ ~ ~ End Of Part 5 ~ ~
~ ~ ~ ~ ~\n");
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("\n");

        System.out.println("\033[3mMoving to part
6.\033[0m\n\n");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Press [ Enter ] key to continue.");
    sc.nextLine();
    System.out.print("[ Enter ] key pressed. \n");
    System.out.println("\nMoving to the next part in...");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("3..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("2..");
    try {
        Thread.sleep(1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("1..");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("\n\n\033[3m- - - - - PART 6 :

```

```

Protecting Wall Of Maria - - - - -\033[0m");
    System.out.println("\n");

    System.out.println("\nWall of Maria has already been
generated. ");
    System.out.println("\nScanning weakest part of Wall of
Maria....\n");
    List<List<Integer>> layer = new ArrayList<>();
    List<Integer> brick1 = new ArrayList<>();
    List<Integer> brick2 = new ArrayList<>();
    List<Integer> brick3 = new ArrayList<>();
    List<Integer> brick4 = new ArrayList<>();
    ArrayList<Integer> arr = new ArrayList<>();
    brick1.add(3);
    brick1.add(6);
    brick1.add(9);
    layer.add(brick1);
    brick2.add(2);
    brick2.add(5);
    brick2.add(8);
    brick2.add(9);
    layer.add(brick2);
    brick3.add(1);
    brick3.add(4);
    brick3.add(10);
    layer.add(brick3);
    brick4.add(5);
    brick4.add(7);
    brick4.add(9);
    layer.add(brick4);
    arr = ConvertToOneList(layer);
    int test = mostFrequentArrayList(arr, arr.size());
    System.out.println("Weakest part of Wall of Maria is at
position " + test + "\n");
    System.out.println("\n\n~ ~ ~ ~ ~ End Of Part 6 ~ ~
~ ~ ~ ~ ~\n");
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("\n");

    System.out.println("You have reached the end of the
game.");
    System.out.println("Thank you for playing. :) \n\n");
} catch (NumberFormatException e) {
    e.printStackTrace();
}

```

```

    }

    public static ArrayList<Integer>
ConvertToOneList(List<List<Integer>> data) {
    ArrayList<Integer> arr = new ArrayList<>();
    while (!data.isEmpty()) {
        List<Integer> layer = new ArrayList<>();
        layer = data.get(0);
        data.remove(0);
        while (!layer.isEmpty()) {
            arr.add(layer.get(0));
            layer.remove(0);
        }
    }
    return arr;
}

public static int mostFrequentArrayList(ArrayList<Integer> arr,
int n) {
    // Sort the array
    Collections.sort(arr);
    // find the max frequency using linear traversal
    Integer max_count = 1;
    Integer res = arr.get(0);
    int curr_count = 1;

    for (int i = 1; i < n; i++) {
        if (Objects.equals(arr.get(i), arr.get(i - 1))) {
            curr_count++;
        } else {
            curr_count = 1;
        }

        if (curr_count > max_count) {
            max_count = curr_count;
            res = arr.get(i - 1);
        }
    }
    return res;
}
}

```

## **5.0 Extra features**

### **5.1 Implement breadth-first search algorithm**

We use breadth-first search to find the best known output path for the Titan killing process. This will help Eren's allies to kill the Titan easily by using the best path to reach the Titan.

### **5.2 Implement loading screen to the output**

We use ' *Thread.sleep(time in milliseconds);* ' to implement the loading screen in the output. This will give a delay time for Java to display the output so, it will give users a bit of experience of a real game environment.

### **5.3 Implement *italic* word to display the output**

We use "*\033[3m Enter any word here \033[0m*"; to display italic word in the output.

## 6.0 Conclusion

We as Eren's friends have done this program to help him survive in the world of titans. This programme allows him to add his friends to the list of available soldiers and arrange them by their characteristics. Aside from that, this application can search among his friends for specified characteristic values. This is how we arrange Eren's friends' characteristics.

The next crucial part of the program is it has the ability to sort titans according to their risk. To find the titans, the scout regiment needs to scout the area outside of the wall. With the map provided, this program can find the most optimal way to scout all of the areas without having to recheck the same area. If there are titans in the region after scouting, this software can assist in determining the optimal path for the soldier to take in order to eliminate the titans from the area.

The last part is more to gain information where the program allows the soldiers to translate any Marleyan sentences into English. This program can also check for the weakest part in the Wall of Maria. That is the program that we have created.

## 7.0 References

Hamiltonian cycle: Backtracking-6. GeeksforGeeks. (2022, January 6). Retrieved June 22, 2022, from <https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/>

TaranfxTaranfx 4111 gold badge11 silver badge33 bronze badges, Kannan Ekanat Kannan Ekanath 15.7k2121 gold badges7171 silver badges9898 bronze badges, Mark PopeMark Pope 11k1010 gold badges4747 silver badges5858 bronze badges, Rajesh DixitRajesh Dixit 18111 silver badge33 bronze badges, Tom NeylandTom Neyland 6, & GauravRatnawatGauravRatnawat 49911 gold badge66 silver badges1515 bronze badges. (1957, October 1). *How to create your own HashMap in Java?* Stack Overflow. Retrieved June 22, 2022, from <https://stackoverflow.com/questions/2397188/how-to-create-own-hashmap-in-java>

*Most frequent element in an array.* GeeksforGeeks. (2022, June 20). Retrieved June 22, 2022, from <https://www.geeksforgeeks.org/frequent-element-array/>

YouTube. (2020, February 6). *Java GUI tutorial - make a GUI in 13 minutes.* YouTube. Retrieved June 22, 2022, from <https://www.youtube.com/watch?v=5o3fMLPY7qY&t=580s>

YouTube. (2021, December 22). *Create your first java frame using visual studio code | create java GUI forms using vs code.* YouTube. Retrieved June 22, 2022, from <https://www.youtube.com/watch?v=5G2XM1nIX5Q&t=374s>