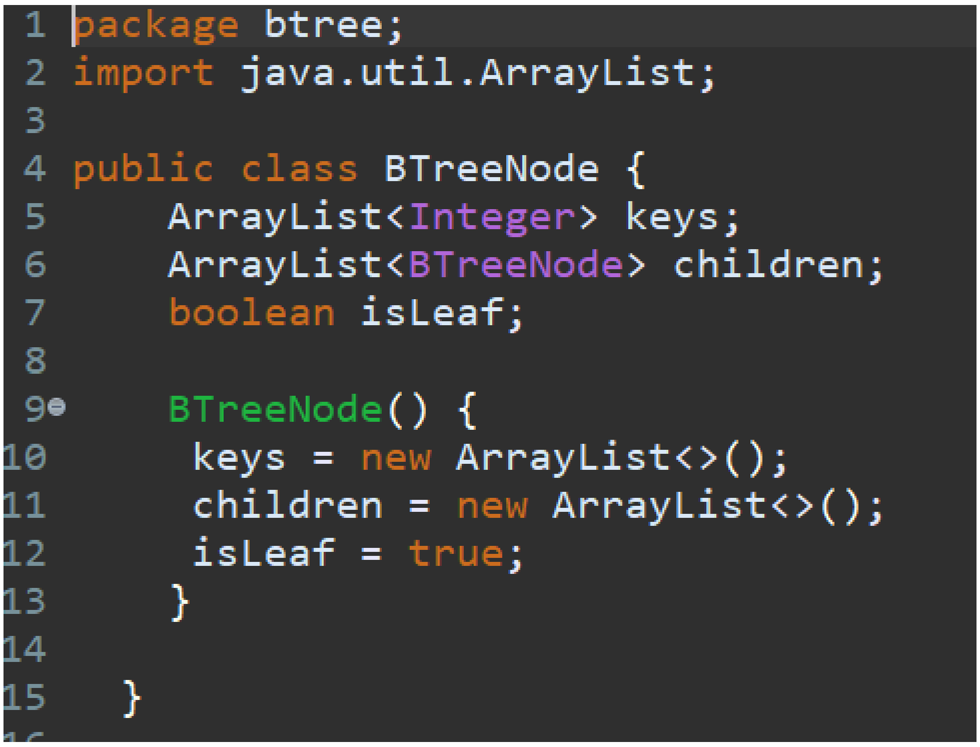
DATA STRUCTURES HOMEWORK

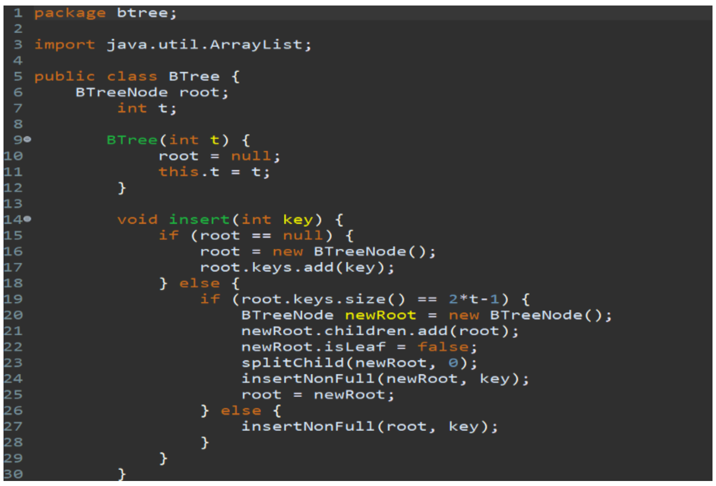
1)



```java
package btree;
import java.util.ArrayList;

public class BTreeNode {
    ArrayList<Integer> keys;
    ArrayList<BTreeNode> children;
    boolean isLeaf;

    BTreeNode() {
     keys = new ArrayList<>();
     children = new ArrayList<>();
     isLeaf = true;
    }

  }
```

In the above part, first we have created class that is called the BTreeNode to use it in the BTree class. In this class, we defined two array lists and one boolean variable. One of the array lists is created for storing the keys and the other one is created for storing the children. And the “ isLeaf” variable is created for the decide if it is leaf or not.

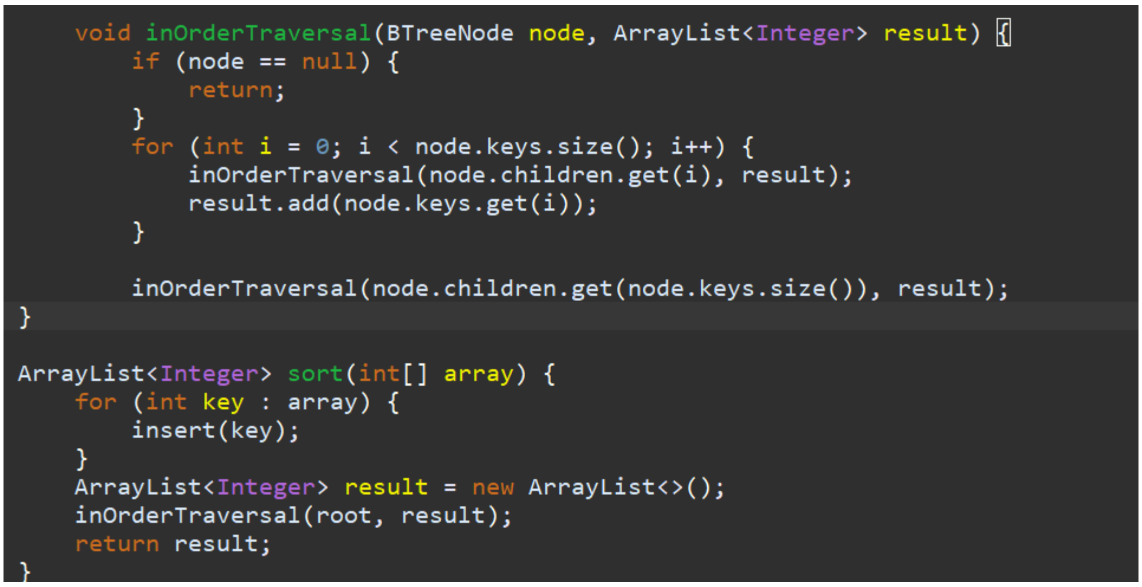
Also, there is a constructor called BTreeNode and it takes no arguments. It creates keys and children arraylists and also it assigns true value to “ isLeaf”.



```java
package btree;

import java.util.ArrayList;

public class BTree {
    BTreeNode root;
        int t;

       BTree(int t) {
            root = null;
            this.t = t;
        }

        void insert(int key) {
            if (root == null) {
                root = new BTreeNode();
                root.keys.add(key);
            } else {
                if (root.keys.size() == 2*t-1) {
                    BTreeNode newRoot = new BTreeNode();
                    newRoot.children.add(root);
                    newRoot.isLeaf = false;
                    splitChild(newRoot, 0);
                    insertNonFull(newRoot, key);
                    root = newRoot;
                } else {
                    insertNonFull(root, key);
                }
            }
        }
```

In this code part, we have created the BTree class to use BTree. First we import arraylist by using “import java.util.ArrayList;” code. And we need root variable from BTreeNode. Also wee need to use integer value to decide number of keys and “ t” variable is used for this. Also we created constructor and “insert method”.

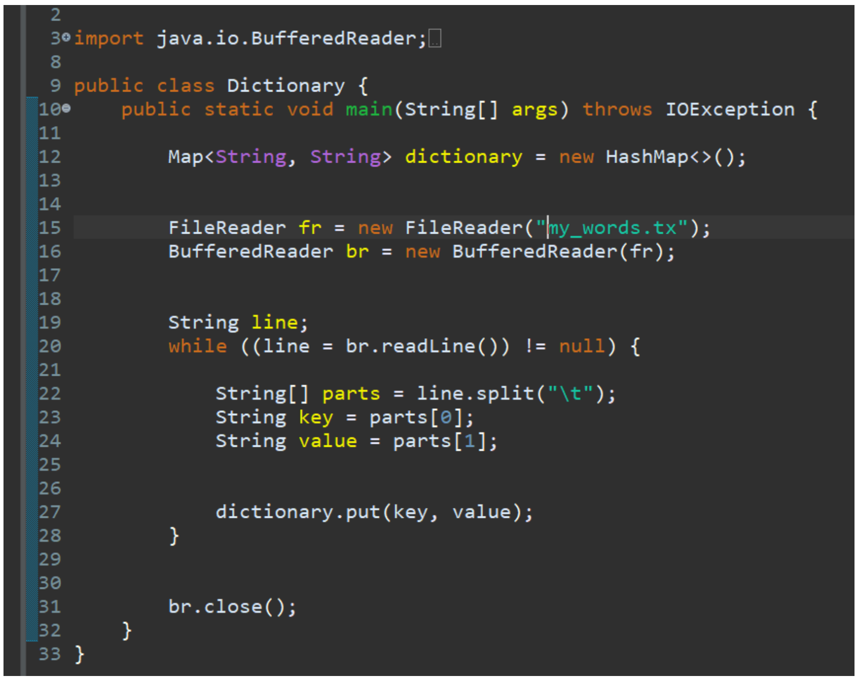
İnstert method takes a key and insert it to a BTree unless BTree is full. And it returns void. If root.keys.size() is full it creates new root.



```java
    void inOrderTraversal(BTreeNode node, ArrayList<Integer> result) {
        if (node == null) {
            return;
        }
        for (int i = 0; i < node.keys.size(); i++) {
            inOrderTraversal(node.children.get(i), result);
            result.add(node.keys.get(i));
        }

        inOrderTraversal(node.children.get(node.keys.size()), result);
}

ArrayList<Integer> sort(int[] array) {
    for (int key : array) {
        insert(key);
    }
    ArrayList<Integer> result = new ArrayList<>();
    inOrderTraversal(root, result);
    return result;
}
```

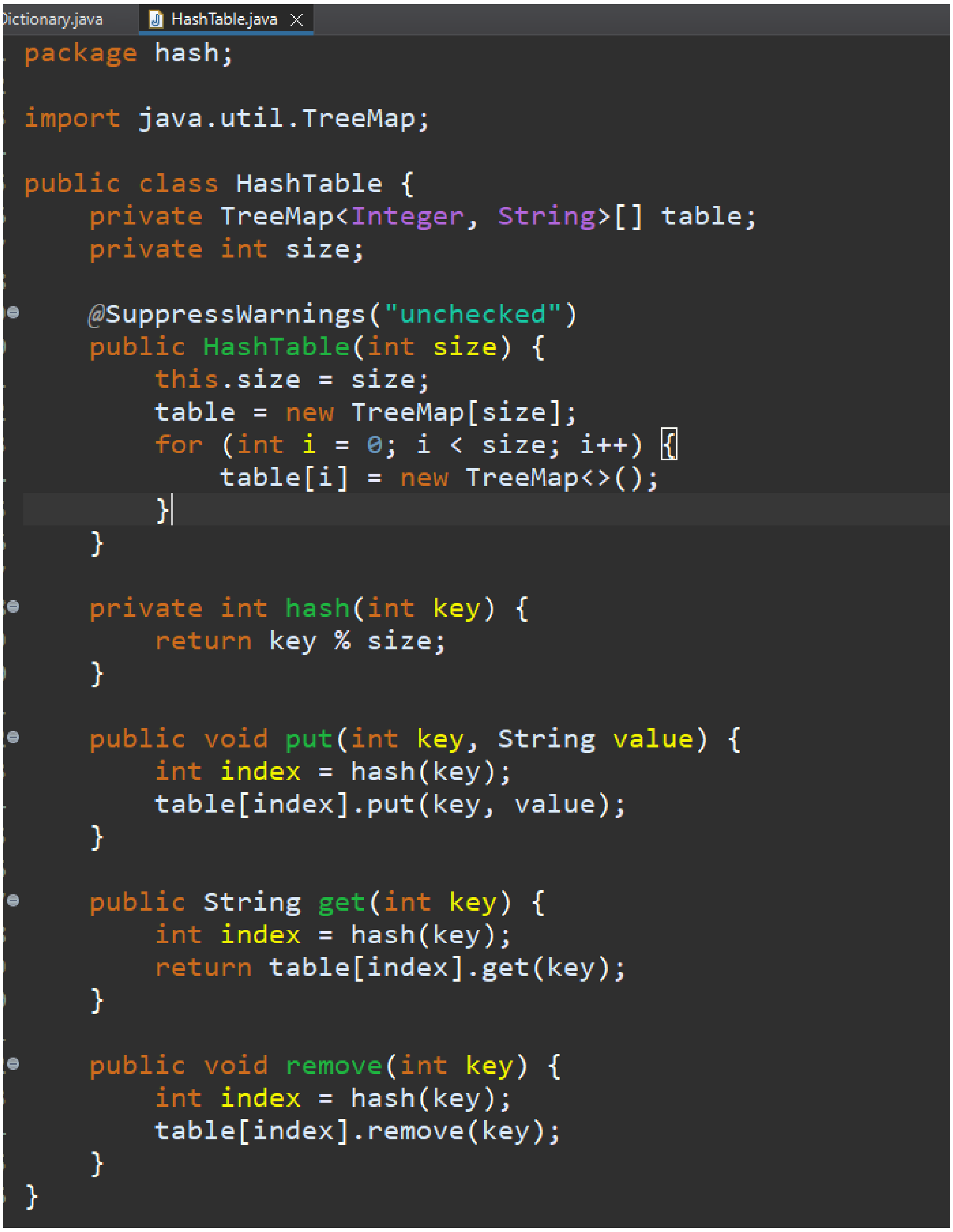
Here, we created inOrderTraversal method to traverse the BTree. Also there is a sort method. It takes an arraylist and returns sorted arraylist.

2)



```java

import java.io.BufferedReader;

public class Dictionary {
    public static void main(String[] args) throws IOException {

        Map<String, String> dictionary = new HashMap<>();


        FileReader fr = new FileReader("my_words.tx");
        BufferedReader br = new BufferedReader(fr);


        String line;
        while ((line = br.readLine()) != null) {

            String[] parts = line.split("\t");
            String key = parts[0];
            String value = parts[1];


            dictionary.put(key, value);
        }


        br.close();
    }
}
```

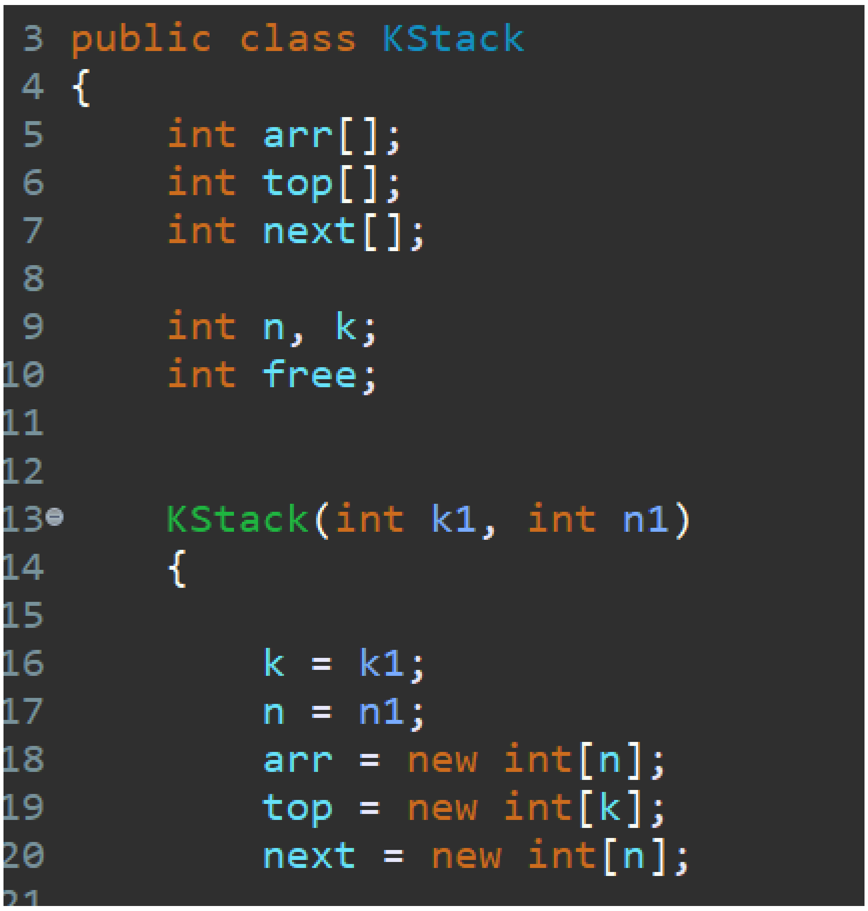
In the 2. Question first, we need to create a class to read from file that is called "my_words.txt" to do that we have to import BufferedReader. In this class we created HashMap to store keys and values. and by using the while part we read the words that are in the "my_words.txt". when it reachs the "\" it stops reading and closes the file.



```java
package hash;

import java.util.TreeMap;

public class HashTable {
    private TreeMap<Integer, String>[] table;
    private int size;

    @SuppressWarnings("unchecked")
    public HashTable(int size) {
        this.size = size;
        table = new TreeMap[size];
        for (int i = 0; i < size; i++) {
            table[i] = new TreeMap<>();
        }
    }

    private int hash(int key) {
        return key % size;
    }

    public void put(int key, String value) {
        int index = hash(key);
        table[index].put(key, value);
    }

    public String get(int key) {
        int index = hash(key);
        return table[index].get(key);
    }

    public void remove(int key) {
        int index = hash(key);
        table[index].remove(key);
    }
}
```

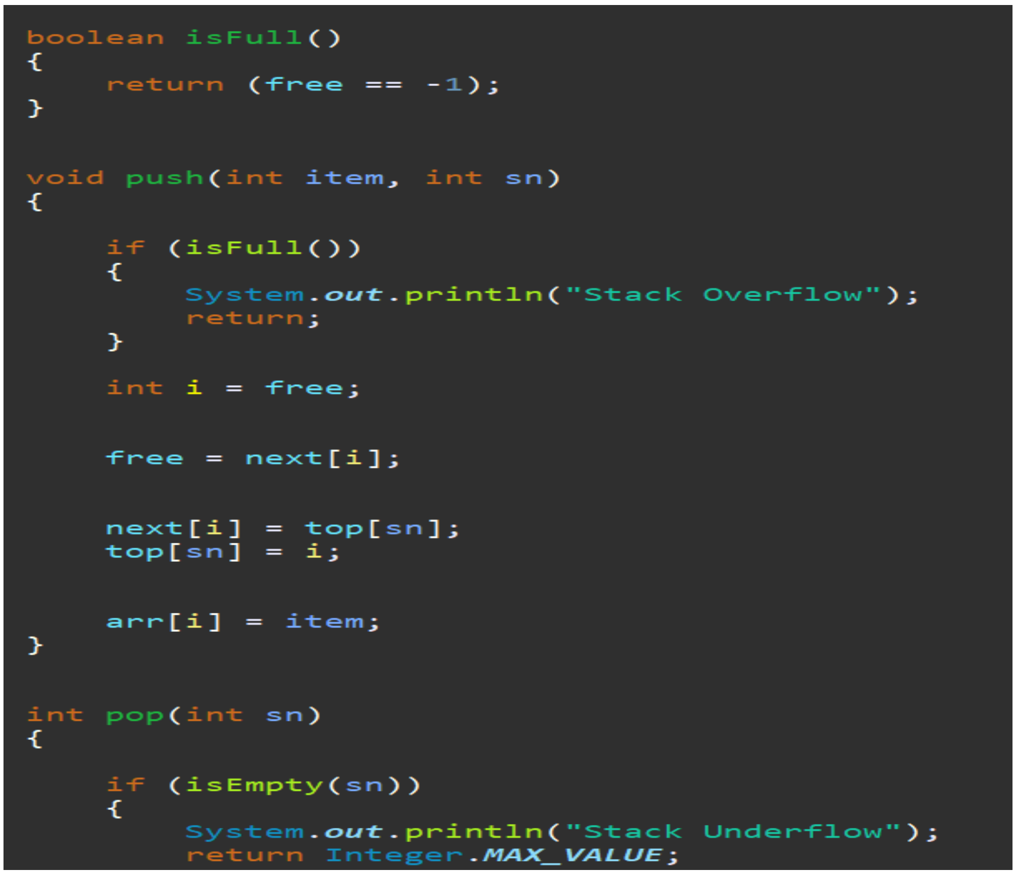
In hash table, we have used TreeMap to resolve collisions that is a balanced binary search tree that stores key - value pairs in ascending order. This is an afficient way for insertion, deletion and searching operations. It is also the best way for the time complexity. By using hash method we return an integer then we use it as a index in the put method. Put method places the value to the index. Get method takes an key and returns it from the hash table. By using the remove method, we can remove the value that is matched with the key.

3)



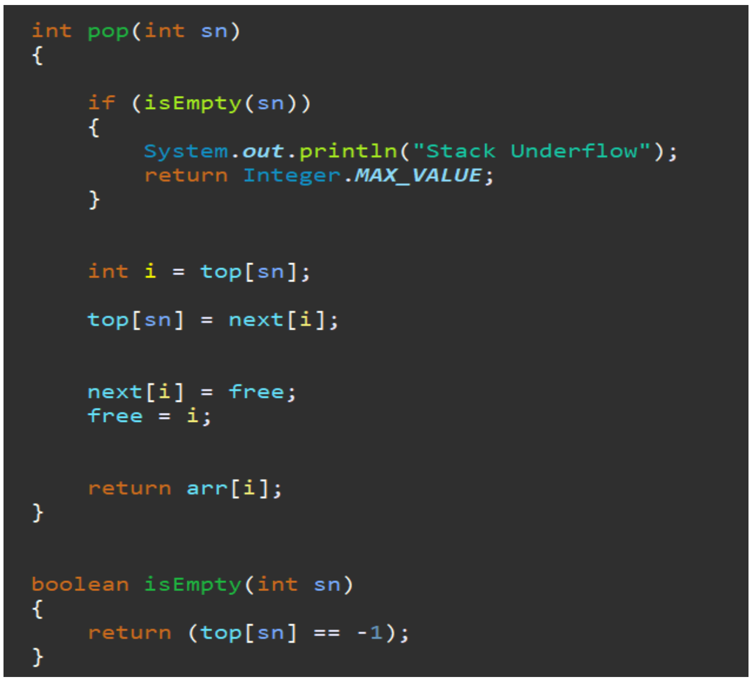
```
public class KStack
{
    int arr[];
    int top[];
    int next[];

    int n, k;
    int free;


    KStack(int k1, int n1)
    {

        k = k1;
        n = n1;
        arr = new int[n];
        top = new int[k];
        next = new int[n];
```

First we created a class that is called KStack and defined arrays that are called arr, top and next. Also defined integer variables n,k and free. Then we created a constructor and this constructor takes two integer values as integer.



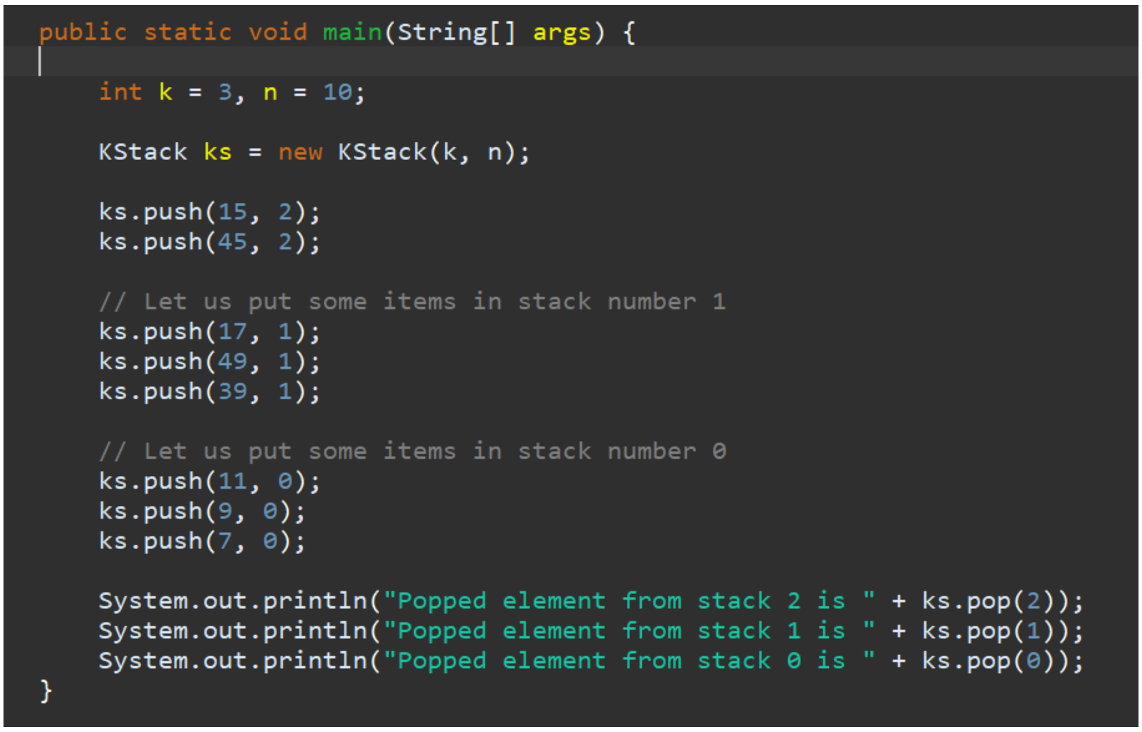
```
boolean isFull()
{
     return (free == -1);
}

void push(int item, int sn)
{

     if (isFull())
     {
          System.out.println("Stack Overflow");
          return;
     }

     int i = free;


     free = next[i];


     next[i] = top[sn];
     top[sn] = i;


     arr[i] = item;
}


int pop(int sn)
{

     if (isEmpty(sn))
     {
          System.out.println("Stack Underflow");
          return Integer.MAX_VALUE;
```

Here, isFull method is used to know if stack is full or not. Push method takes two arguments and if stack is full it will print “Stack Overflow”. İf not then it will place the item in the stack.



```
int pop(int sn)
{

     if (isEmpty(sn))
     {
          System.out.println("Stack Underflow");
          return Integer.MAX_VALUE;
     }


     int i = top[sn];

     top[sn] = next[i];


     next[i] = free;
     free = i;


     return arr[i];
}


boolean isEmpty(int sn)
{
     return (top[sn] == -1);
}
```

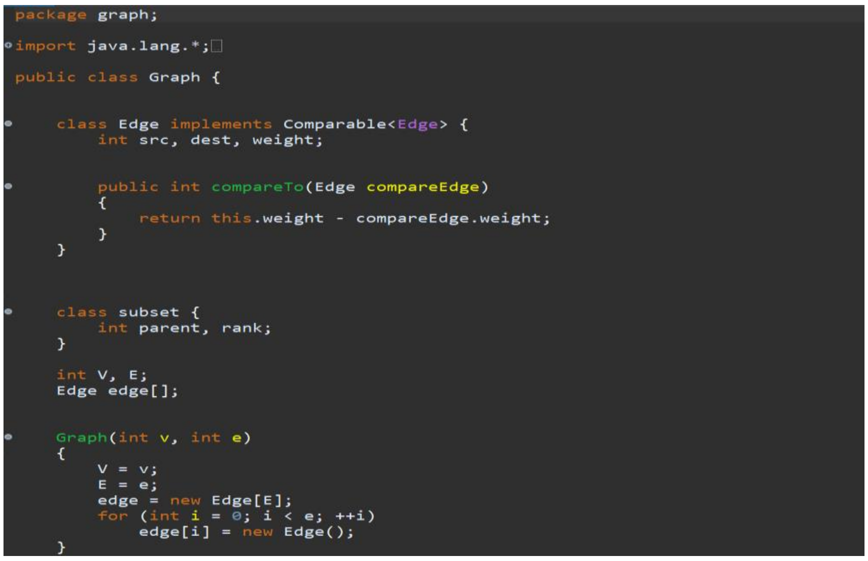
In this part, pop method used the pop from the stack, if it is not empty. Also isEmpty method used to know if stack is empty.



```java
public static void main(String[] args) {

    int k = 3, n = 10;

    KStack ks = new KStack(k, n);

    ks.push(15, 2);
    ks.push(45, 2);

    // Let us put some items in stack number 1
    ks.push(17, 1);
    ks.push(49, 1);
    ks.push(39, 1);

    // Let us put some items in stack number 0
    ks.push(11, 0);
    ks.push(9, 0);
    ks.push(7, 0);

    System.out.println("Popped element from stack 2 is " + ks.pop(2));
    System.out.println("Popped element from stack 1 is " + ks.pop(1));
    System.out.println("Popped element from stack 0 is " + ks.pop(0));
}
```

In the main class, we created variables that are called are k and n. Then we assigned their values. After this part, we created object of KStack class. Then pushed some values into them by using stack number. Finally, we popped from these stacks by using pop method.

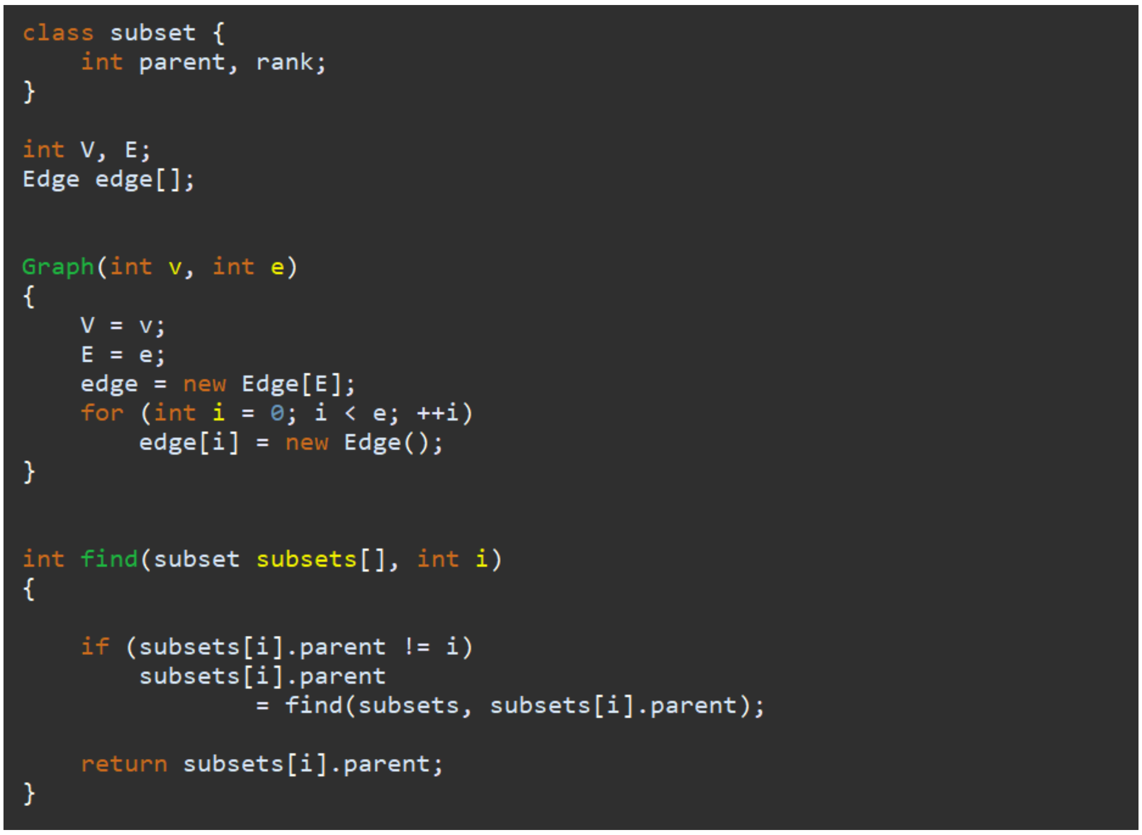
4)



```java
package graph;

import java.lang.*;

public class Graph {


    class Edge implements Comparable<Edge> {
        int src, dest, weight;


        public int compareTo(Edge compareEdge)
        {
            return this.weight - compareEdge.weight;
        }
    }



    class subset {
        int parent, rank;
    }

    int V, E;
    Edge edge[];


    Graph(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }
```

To solve this problem, we used Kruskal’s algorithm to find minimum spanning tree (MST) of given graph. This algorithm is best way to solve this problem because it can be implemented easly and has lower time complexity than other algorithms.

First we need to define a class that is called Graph and this class contains severeal inner method and classes. Class Edge is used for representing the edges and it has three fields that are weight, dest and src.

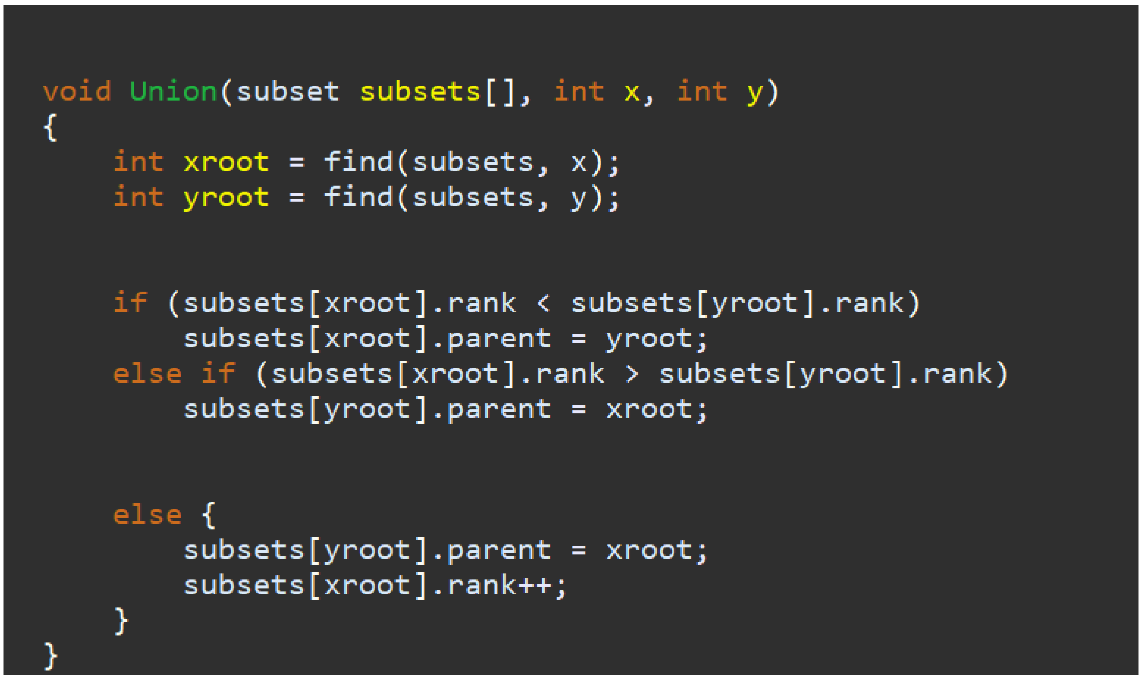


```
class subset {
    int parent, rank;
}

int V, E;
Edge edge[];


Graph(int v, int e)
{
    V = v;
    E = e;
    edge = new Edge[E];
    for (int i = 0; i < e; ++i)
        edge[i] = new Edge();
}


int find(subset subsets[], int i)
{

    if (subsets[i].parent != i)
        subsets[i].parent
                = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}
```

The inner subset class is used for a set for the union-find data structure that is used in Kruskal’s algorithm do specify cycles. Subset class has only two fields and there are parent and rank. Also graph class has three fields and there are named as V, E and edge.

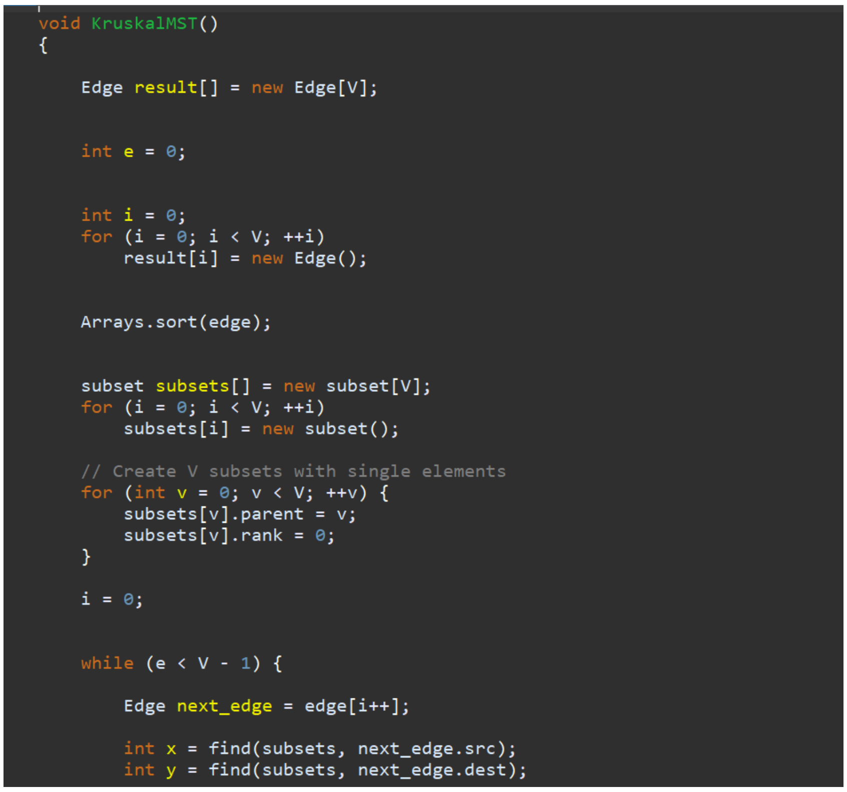
V indicates the number of vertices in the graph, E indicates the number of edges in the graph and the edge is an array that is used for storing the all edges of the graph.

Find method is used to find the set of an element i using the path compression technique. It returns the root of the set that containg element i.



```
void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);


    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;


    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

Union method is used for doing the union of two sets and using the union by rank technique. It combines two sets into a single set and assigns a rank to the resulting set according to the ranks of the original sets.



```
void KruskalMST()
{

    Edge result[] = new Edge[V];


    int e = 0;


    int i = 0;
    for (i = 0; i < V; ++i)
        result[i] = new Edge();


    Arrays.sort(edge);


    subset subsets[] = new subset[V];
    for (i = 0; i < V; ++i)
        subsets[i] = new subset();

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    i = 0;


    while (e < V - 1) {

        Edge next_edge = edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
```

KruskalMST method is the main method that implements the Kruskal’s algorithm to find the MST of the graph. Fırst it sorts the edges of the graph in a increasing order of their weight using the Arrays.sort() method. Finally, it creates V subsets, each containing a single element, using the subset class.

Baran Çakmak Demir / 210315002