**Michal Bochenek  
EC1401916**

**Technical Manual**

**For Assessment on Stacks and Queues**

**Table of contents:   
  
1. Program Specification.  
2. Code Listing.  
3. Testing and program running.**

1. **Program Specification.**Program performs operations on stacks and queues in which firstly it loads the cargo to the pods from the data text file, which are stacks and unloads them into the corridor of a space station, which is a queue.   
   After that corridor sends the cargo to the specified storage which is food, technical or personal storage.  
     
   **List of operations to perform:**1. Show the state of the stack and queue before pods are loaded.  
   2. Show the state of the stack and queue when shuttle reaches space station.  
   3. Show the state of the stack and queue after half the stack has been unloaded.  
   4. Show the state of the stack and queue after the stack has been unloaded.  
   5. Show the state of the stack and queue after the loading bay is used.

**Example of a working program:**

|  |
| --- |
| This is cargo sorting program.  Choose an option by pressing number 1 or 2 and press ENTER:  1. Program test with PodData.txt or data run 1/2  2. Generate cargo data.  Exit - none or any other value  1  Testing program.  Choose data file to test 1-9:  1-7 are given text files, 8,9 are data run 1 and data run 2.  8  Stack [pod1] status pre-load:  []  Stack [pod2] status pre-load:  []  Queue status [corridor] pre-load:  []  Pod1 status on reaching the station:  [F11, F34, P22, T56, F16, T77, P12, F41, T22]  Pod2 status on reaching the station:  [P19, F39, T92, T36, P36, T75, P15, F48, F88]  Corridor status on reaching the station:  [] |

1. **Code listing.**
   1. **Main program.**

package stackandqueuesassessment;

import java.io.\*;

import java.util.\*;

import java.util.concurrent.\*;

/\*\*

\*

\* @author Michal Bochenek

\*/

public class StackAndQueuesAssessment

{

public static void main(String[] args) throws InterruptedException

{

//global variables to be used later

//pod 1 and 2 with cargo loaded

Stack<String> pod1 = new Stack<>();

Stack<String> pod2 = new Stack<>();

//this is the corridor queue at the space station max 18 items are allowed

BlockingQueue<String> corridor = new ArrayBlockingQueue<>(18);

//cargo bay

ArrayList<String> food = new ArrayList<>();

ArrayList<String> technical = new ArrayList<>();

ArrayList<String> personal = new ArrayList<>();

//other

int option;

System.out.println("This is cargo sorting program.");

System.out.println("Choose an option by pressing number 1 or 2 and press ENTER:");

System.out.println("1. Program test with PodData.txt or data run 1/2");

System.out.println("2. Generate cargo data.");

System.out.println("Exit - none or any other value");

Scanner sd = new Scanner(System.in);

option = sd.nextInt();

if(option==2)

{

//generate cargo

//this is our cargo which will be loaded to the pod 1 and 2 respectively

Set<String> cargo1 = new HashSet<>();

Set<String> cargo2 = new HashSet<>();

LoadPods genload = new LoadPods(cargo1, cargo2);

genload.LoadPods();

//load pods with data

cargo1.addAll(genload.car1);

cargo2.addAll(genload.car2);

//start sorting

Sorting generator = new Sorting();

generator.option2(cargo1, cargo2);

}

else if(option==1)

{

//testing program with txt files or run data 1/2

Sorting txtfile = new Sorting();

txtfile.option1();

}

}

}

**2.2 Sorting class. Most/all calculations and functions are put here.**

package stackandqueuesassessment;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

import java.util.Set;

import java.util.Stack;

import java.util.StringTokenizer;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

/\*\*

\*

\* @author ec1401916

\*/

public class Sorting

{

//here we will put our cargo to be loaded to the pods

ArrayList<String> stacking = new ArrayList<>();

//global variables to be used later

//pod 1 and 2 with cargo loaded

private Stack<String> pod1 = new Stack<>();

private Stack<String> pod2 = new Stack<>();

//this is the corridor queue at the space station max 18 items are allowed

private BlockingQueue<String> corridor = new ArrayBlockingQueue<>(18);

//cargo bay

private ArrayList<String> food = new ArrayList<>();

private ArrayList<String> technical = new ArrayList<>();

private ArrayList<String> personal = new ArrayList<>();

//other variables and markers

private String t ="T";

private String p ="P";

private String f ="F";

void option1() throws InterruptedException

{

try

{

System.out.println("Testing program.");

System.out.println("Choose data file to test 1-9: ");

System.out.println("1-7 are given text files, 8,9 are data run 1 and data run 2.");

Scanner sc = new Scanner(System.in);

int fnumber = sc.nextInt();

//build a file name

String nameoffile = "PodData"+Integer.toString(fnumber)+".txt" ;

//System.out.println(nameoffile);

//show stack and queue status before pods are loaded

System.out.println("Stack [pod1] status pre-load: ");

System.out.println(pod1);

System.out.println("Stack [pod2] status pre-load: ");

System.out.println(pod2);

System.out.println("Queue status [corridor] pre-load: ");

System.out.println(corridor);

File myFile = new File(nameoffile);

FileReader fr = new FileReader(myFile);

BufferedReader br = new BufferedReader(fr);

String line = null;

//variable to show half the way state of stack and queue

int halfway=0;

while((line=br.readLine())!=null)

{

StringTokenizer st = new StringTokenizer(line, ",");

int tokens = st.countTokens();

halfway=tokens/2;

while(st.hasMoreTokens())

{

stacking.add(st.nextToken());

}

//in case there is not enough items to take:

if(tokens<9)

{

System.out.println("There is not enough cargo to load pod1 size: "+tokens);

System.out.println("Program will run incorrectly.");

for(int i=0; i<9; i++)

{

pod1.push(stacking.get(i));

}

for(int j=9; j<tokens; j++)

{

pod2.push(stacking.get(j));

}

}

//in case there is too many items to take:

else if(tokens>18)

{

System.out.println("There is too much cargo to load, an error might occur (overwritten stack or queue): "+tokens);

//for testing purpose

for(int i=0; i<9; i++)

{

pod1.push(stacking.get(i));

}

for(int j=9; j<tokens; j++)

{

pod2.push(stacking.get(j));

}

}

//in case there is just enough items to take:

else if(tokens>0&&tokens<=18)

{

for(int i=0; i<9; i++)

{

pod1.push(stacking.get(i));

}

for(int j=9; j<tokens; j++)

{

pod2.push(stacking.get(j));

}

//shuttle reaches the station:

System.out.println("Pod1 status on reaching the station: ");

System.out.println(pod1);

System.out.println("Pod2 status on reaching the station: ");

System.out.println(pod2);

System.out.println("Corridor status on reaching the station: ");

System.out.println(corridor);

System.out.println("");

}

//start queueing items:

System.out.println("Unloading cargo at the corridor: ");

for(int k=0; k<9; k++)

{

if(halfway==k)

{

System.out.println("Stack 1 [pod1] status on half way unload: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status on half way unload: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status on half way unload: ");

System.out.println(corridor);

System.out.println("");

}

corridor.put(pod1.pop());

}

for(int l=9; l<tokens; l++)

{

if(halfway==l)

{

System.out.println("Stack 1 [pod1] status on half way unload: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status on half way unload: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status on half way unload: ");

System.out.println(corridor);

System.out.println("");

}

corridor.put(pod2.pop());

}

System.out.println("Stack 1 [pod1] status after unloading: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status after unloading: ");

System.out.println(pod2);

System.out.println("Corridor status after unloading: ");

System.out.println(corridor);

System.out.println("");

//start sorting cargo from queue:

System.out.println("Sorting cargo to the storage.");

while(!corridor.isEmpty())

{

if(corridor.peek().startsWith(t))

{

technical.add(corridor.remove());

}

else if(corridor.peek().startsWith(p))

{

personal.add(corridor.remove());

}

else if(corridor.peek().startsWith(f))

{

food.add(corridor.remove());

}

else

{

System.out.println("unknown cargo detected! ID: "+corridor.take()+" it will be cast into space.");

}

}

System.out.println("Food storage status:");

System.out.println(food);

System.out.println("Personal storage status:");

System.out.println(personal);

System.out.println("Technical storage status:");

System.out.println(technical);

System.out.println("Stack 1 [pod1] status after being used: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status after being used: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status after being used: ");

System.out.println(corridor);

}

br.close();

}

catch(IOException e)

{

System.out.println("An unknown IO Error has occured");

}

}

void option2(Set<String> cargo1, Set<String> cargo2) throws InterruptedException

{

stacking.addAll(cargo1);

stacking.addAll(cargo2);

int halfway = stacking.size()/2;

//show stack and queue status before pods are loaded

System.out.println("Stack [pod1] status pre-load: ");

System.out.println(pod1);

System.out.println("Stack [pod2] status pre-load: ");

System.out.println(pod2);

System.out.println("Queue status [corridor] pre-load: ");

System.out.println(corridor);

//in this case our pod1 will always be full and total number of elements never larger than 18

for(int i=0; i<9;i++)

{

pod1.push(stacking.get(i));

}

if(stacking.size()>=9)

{

for(int j=9; j<stacking.size(); j++)

{

pod2.push(stacking.get(j));

}

}

else if(stacking.contains("000"))

{

System.out.println("Pod2 is empty.");

}

else

{

System.out.println("An unknown error has occured.");

}

System.out.println("Pod1 status when reaching space station: ");

System.out.println(pod1);

System.out.println("Pod2 status when reaching space station: ");

System.out.println(pod2);

System.out.println("Corridor status when reaching space station: ");

System.out.println(corridor);

System.out.println("");

System.out.println("");

System.out.println("Unloading cargo at the corridor: ");

//start unloading cargo to the corridor

//pod1 unload

for(int k=0; k<9; k++)

{

if(halfway==k)

{

System.out.println("Stack 1 [pod1] status on half way unload: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status on half way unload: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status on half way unload: ");

System.out.println(corridor);

System.out.println("");

}

corridor.put(pod1.pop());

}

//pod2 unload

if(!stacking.contains("000"))

{

for(int l=9; l<stacking.size(); l++)

{

if(halfway==l)

{

System.out.println("Stack 1 [pod1] status on half way unload: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status on half way unload: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status on half way unload: ");

System.out.println(corridor);

System.out.println("");

}

corridor.put(pod2.pop());

}

}

//show status after unloading to the corridor

System.out.println("Stack 1 [pod1] status after unloading: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status after unloading: ");

System.out.println(pod2);

System.out.println("Corridor status after unloading: ");

System.out.println(corridor);

System.out.println("");

System.out.println("Sorting cargo to the storage.");

while(!corridor.isEmpty())

{

if(corridor.peek().startsWith(t))

{

technical.add(corridor.remove());

}

else if(corridor.peek().startsWith(p))

{

personal.add(corridor.remove());

}

else if(corridor.peek().startsWith(f))

{

food.add(corridor.remove());

}

else

{

System.out.println("unknown cargo detected! ID: "+corridor.take()+" it will be cast into space.");

}

}

System.out.println("Food storage status:");

System.out.println(food);

System.out.println("Personal storage status:");

System.out.println(personal);

System.out.println("Technical storage status:");

System.out.println(technical);

System.out.println("Stack 1 [pod1] status after being used: ");

System.out.println(pod1);

System.out.println("Stack 2 [pod2] status after being used: ");

System.out.println(pod2);

System.out.println("Queue [corridor] status after being used: ");

System.out.println(corridor);

}

}

**2.3 LoadPods Class – generates random cargo between 9-18 items.**  
  
package stackandqueuesassessment;

import java.util.\*;

public class LoadPods

{

Set<String> car1 = new HashSet<>();

Set<String> car2 = new HashSet<>();

LoadPods(Set<String> cargo1, Set<String> cargo2)

{

cargo1 = car1;

cargo2 = car2;

}

public String GenerateCargo()

{

//generates pseudorandom number

Random rgen = new Random();

//use type and nbr to build cargo ID later on and return cargo

String type;

String nbr;

String cargoID;

//type of cargo T - technology, F - food, P - personal

String[] typeofcargo = {"T", "F", "P"};

String[] cargonumber = new String[89];

//filling cargonumber with string values

for(int i=0; i<89; i++)

{

int j=10+i;

cargonumber[i] = Integer.toString(j);

}

//pointer for searching the respective arrays

int t;

int c;

t = rgen.nextInt(2);

c = rgen.nextInt(88);

//use generated numbers to build a cargoID string

type = typeofcargo[t];

nbr = cargonumber[c];

cargoID = type + nbr;

return cargoID;

}

//method for loading pod1 [always full]

public Set<String> LoadPods()

{

//this will generate a set of 9 different! types of cargo, since there can be only one entry of the same type in a set

while(car1.size()<9)

{

car1.add(GenerateCargo());

}

//System.out.println(car1);

//generates pseudorandom number

Random rgen = new Random();

//generate cargo2 size 0-9 elements

int pod2size = rgen.nextInt(9);

if(pod2size>0)

{

while(car2.size()<=pod2size)

{

car2.add(GenerateCargo());

//no such element can be in both sets, prevent this by this statement

if(car2.contains(car1))

{

car2.remove(car1);

}

}

//check if car2 contains car1 element, returns true if yes

//System.out.println(car1.contains(car2));

}

return new carload(car1, car2);

}

}

1. **Testing (white box) and program running.  
     
   Show the state of the stack and queue before the pods are loaded:**

|  |
| --- |
| Stack [pod1] status pre-load:  []  Stack [pod2] status pre-load:  []  Queue status [corridor] pre-load:  [] |

**Show the state of the stack and queue when shuttle reaches space station:**

|  |
| --- |
| Pod1 status on reaching the station:  [F11, F34, P22, T56, F16, T77, P12, F41, T22]  Pod2 status on reaching the station:  [P19, F39, T92, T36, P36, T75, P15, F48, F88]  Corridor status on reaching the station:  [] |

**Show the state of the stack and queue after half the stack has been unloaded:**

|  |
| --- |
| Stack 1 [pod1] status on half way unload:  []  Stack 2 [pod2] status on half way unload:  [P19, F39, T92, T36, P36, T75, P15, F48, F88]  Queue [corridor] status on half way unload:  [T22, F41, P12, T77, F16, T56, P22, F34, F11] |

**Show the state of the stack and queue after the stack has been unloaded:**

|  |
| --- |
| Stack 1 [pod1] status after unloading:  []  Stack 2 [pod2] status after unloading:  []  Corridor status after unloading:  [T22, F41, P12, T77, F16, T56, P22, F34, F11, F88, F48, P15, T75, P36, T36, T92, F39, P19] |

**Show the state of the stack and queue after the loading bay is used:**

|  |
| --- |
| Stack 1 [pod1] status after being used:  []  Stack 2 [pod2] status after being used:  []  Queue [corridor] status after being used:  [] |

**Testing:**Due to length of the program I’ve decided to shorten test data part to class name and what is tested.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test nr:** | **Test data:** | **Expected Result:** | **Actual Result:** |
| 1. | StackAndQueuesAssessment  Class  – main menu displaying | Displays menu and prompts for choosing an option. | This is cargo sorting program.  Choose an option by pressing number 1 or 2 and press ENTER:  1. Program test with PodData.txt or data run 1/2  2. Generate cargo data.  Exit - none or any other value  1 |
| 2. | Sorting class  – prompt for choosing data file to test by pressing number from 1-9 | Displays a prompt for choosing an option | Testing program.  Choose data file to test 1-9:  1-7 are given text files, 8,9 are data run 1 and data run 2.  8 |
| 3. | Sorting class – cargo from corridor is being processed to the storage  Corridor status after unloading:  [T22, F41, P12, T77, F16, T56, P22, F34, F11, F88, F48, P15, T75, P36, T36, T92, F39, P19] | Sorts and assigns cargo to the specified storage [food, personal, technical] | Sorting cargo to the storage.  Food storage status:  [F41, F16, F34, F11, F88, F48, F39]  Personal storage status:  [P12, P22, P15, P36, P19]  Technical storage status:  [T22, T77, T56, T75, T36, T92] |
| 4. | Sorting class – not enough cargo is in pod1 [PodData9.txt] – option 9 | Displays an error message. | There is not enough cargo to load pod1 size: 8  Program will run incorrectly.  Exception in thread "main" java.lang.IndexOutOfBoundsException: |
| 5. | Sorting class – pod2 is empty [PodData3.txt] – option 3  Pod1 status on reaching the station:  [T76, T77, T99, T14, T24, T88, T63, T53, T02]  Pod2 status on reaching the station:  [] | Program runs correctly. | Sorting cargo to the storage.  Food storage status:  []  Personal storage status:  []  Technical storage status:  [T02, T53, T63, T88, T24, T14, T99, T77, T76]  Stack 1 [pod1] status after being used:  []  Stack 2 [pod2] status after being used:  []  Queue [corridor] status after being used:  [] |
| 6. | Sorting class – pod contains unknown cargo [PodData6.txt] – option 6  Pod1 status on reaching the station:  [T16, T17, F99, F14, P34, P88, T63, E58, P02]  Pod2 status on reaching the station:  [P76, E77, T99, R14, P24, T88, F63, F53, T02] | Program runs correctly, but destroys unknown cargo | Corridor status after unloading:  [P02, E58, T63, P88, P34, F14, F99, T17, T16, T02, F53, F63, T88, P24, R14, T99, E77, P76]  Sorting cargo to the storage.  unknown cargo detected! ID: E58 it will be cast into space.  unknown cargo detected! ID: R14 it will be cast into space.  unknown cargo detected! ID: E77 it will be cast into space.  Food storage status:  [F14, F99, F53, F63]  Personal storage status:  [P02, P88, P34, P24, P76]  Technical storage status:  [T63, T17, T16, T02, T88, T99]  Stack 1 [pod1] status after being used:  []  Stack 2 [pod2] status after being used:  []  Queue [corridor] status after being used:  [] |
| 7. | LoadPods – generate new cargo and load it into the pods. | Program runs correctly. | Pod1 status when reaching space station:  [F94, F82, T97, T35, F68, F24, F46, T81, T71]  Pod2 status when reaching space station:  [T63, F12, T16, F38] |
| 8. | Sorting class – pod contains too much cargo [PodData5.txt] – option 5 | Program stops running, blocking queue is full so it waits for it to unload first. | There is too much cargo to load, an error might occur (overwritten stack or queue blocked and waiting): 19  Unloading cargo at the corridor:  Stack 1 [pod1] status on half way unload:  []  Stack 2 [pod2] status on half way unload:  [P76, F77, T99, P14, P24, T88, F63, F53, T02, T11]  Queue [corridor] status on half way unload:  [P02, F58, T63, P88, P34, F14, F99, T17, T16] |