Reti e Laboratorio
Modulo Laboratorio 3
a.a. 2025-2026

docente: Laura Ricci

laura.ricci@unipi.it

Correzione 2
Esercizio di ripasso

02/10/2025

CORREZIONE ESERCIZIO RIPASSO 2

- in un centro di calcolo le singole unità di calcolo che eseguono tasks sono inserite in slot, che vengono identificati univocamente in base a tre numeri
  - la fila
  - la posizione relativa della colonna di unità nella fila
  - la posizione relativa dell'unità all'interno di tale colonna (e.g. terza fila della stanza, quarta colonna della fila, dodicesima unità nella colonna).
- il coordinatore del centro deve mantenere una struttura dati efficiente per l'assegnazione e rimozione di task di ogni unità, con gestione `FIFO` in cui il task più vecchio viene completato e rimosso per primo.

CORREZIONE ESERCIZIO RIPASSO 2

- si definisca una classe generica per la gestione di tuple ordinabili di lunghezza arbitraria e contenenti tipi qualsiasi ma che supportano un ordinamento.
- oltre a supportare l'ordinamento la classe offre getters (getters di tipo non primitivo) e setters per accedere o modificare l'elemento i-esimo.
- si utilizzi la classe delle tuple per individuare univocamente le unità.
- si rappresenti ogni task attraverso una stringa di lunghezza fissata di 16 caratteri.
- si scelga poi un'opportuna combinazione di collections per l'assegnamento e rimozione efficiente di task ad ogni unità.
- l'utente invia una richiesta (aggiunta, rimozione o visione del task corrente) specificando l'unità attraverso al propria tupla e il task stesso come stringa (solo per l'inserimento).
- si definisca un nuovo tipo di eccezione per gestire il caso in cui si tenti di creare una nuova unità con la stessa tripla identificativa di un'unità già esistente.

LA CLASSE TUPLE

```
package Assignment2;
import java.util.ArrayList;
public class Tuple<T extends Comparable<T>> implements Comparable<Tuple<T>>
  { private final int length;
    private ArrayList<T> elements;
    public Tuple(){
        length = 1;
        elements = new ArrayList<T>();
        elements.add(null); }
public Tuple(int l){
        length = l;
        elements = new ArrayList<T>(l);
        for(int i=0; i<l; i++)
            elements.add(null); }
```

LA CLASSE TUPLE

```
public T getElement(int i){
        return elements.get(i); }
public void setElement(int i, T elem){
        elements.set(i, elem);}
public int compareTo(Tuple<T> o) {
        int tempRes = 0;
        for(int i=0; i<length; i++){
             tempRes = this.getElement(i).compareTo(o.getElement(i));
             If (tempRes != 0)
                  return tempRes;
            }
         //arrivati alla fine dell'ArrayList e tutti gli elementi sono uguali
         return 0; } }
```

IL CENTRO DI CALCOLO

```
package Assignment2;
import java.util.LinkedList; import java.util.Queue; import java.util.TreeMap;
public class ComputingCenter {
    private final TreeMap<Tuple<Integer>,Queue<String>> tasksMap;
    public ComputingCenter(){
        tasksMap = new TreeMap<>();
    }
    public void addNewUnit(Tuple<Integer> unit) throws DuplicateException{
        if(tasksMap.containsKey(unit))
            throw(new DuplicateException("Attempting to add an already present
                                          unit."));
        else
            tasksMap.put(unit, new LinkedList<String>());
        }
```

IL CENTRO DI CALCOLO

```
public void addNewTask(String task, Tuple<Integer> unit) throws
                                        IncosistencyException{
        if(!tasksMap.containsKey(unit))
            try{
                addNewUnit(unit);
            } catch (DuplicateException ex) {
            throw(new IncosistencyException("INCONSISTENCY: same unit both
                                        present and not present as key!"));
        }
        tasksMap.get(unit).add(task);
    }
```

IL CENTRO DI CALCOLO

```
public String getFirstTask(Tuple<Integer> unit) throws IncosistencyException{
        if(!tasksMap.containsKey(unit))
            throw(new IncosistencyException("Attempting to remove a task from
                                        a non existing unit."));
        else
            return tasksMap.get(unit).peek();
    }
public String removeCompletedTask(Tuple<Integer> unit) throws
                                                    IncosistencyException{
        if(!tasksMap.containsKey(unit))
            throw(new IncosistencyException("Attempting to remove a task from
                                                a non existing unit."));

        else
            return tasksMap.get(unit).poll();} }
```

LE ECCEZIONI

```
package Assignment2;


public class DuplicateException extends Exception{
    public DuplicateException()
       {super();}
    public DuplicateException(String s)
       {super(s);}
}
```

LE ECCEZIONI

```
package Assignment2;


public class IncosistencyException extends Exception{
    public IncosistencyException()
        {super();}
    public IncosistencyException(String s)
        {super(s);}
}
```

IL MAIN

```
package Assignment2;
public class Assignment2Collections {
    public static void main(String[] args) {
        ComputingCenter cc = new ComputingCenter();
        Tuple<Integer> unit1 = new Tuple<Integer>(3);
        unit1.setElement(0,10);
        unit1.setElement(1,23);
        unit1.setElement(2,1);
        try {
            cc.addNewUnit(unit1);
        } catch (DuplicateException ex) {
            System.err.println("ERROR: duplicate insertion attempt");
        }
```

IL MAIN

```
Tuple<Integer> unit2 = new Tuple<Integer>(3);
unit2.setElement(0,5); unit2.setElement(1,40);  unit2.setElement(2,8);
try {
    cc.addNewUnit(unit2);
} catch (DuplicateException ex) {
    System.err.println("ERROR: duplicate insertion attempt");
    }
Tuple<Integer> unit1bis = new Tuple<Integer>(3);
unit1bis.setElement(0,10); unit1bis.setElement(1,23); unit1bis.setElement(2,1);
try {
    cc.addNewUnit(unit1bis);
} catch (DuplicateException ex) {
    System.err.println("ERROR: duplicate insertion attempt");
    }
```

IL MAIN

```
String [] tasks1 = {"0123456789ABCDEF", "1123456789ABCDEF", "2123456789ABCDEF",
                    "3123456789ABCDEF", "4123456789ABCDEF","5123456789ABCDEF",
                     "6123456789ABCDEF" };
String [] tasks2 = {"7123456789ABCDEF", "8123456789ABCDEF", "9123456789ABCDEF" };
for(String task:tasks1){
        try {
            System.out.println("Adding task "+task+" to unit 1.");
            cc.addNewTask(task, unit1);
        } catch (IncosistencyException ex) {System.err.println("ERROR: task insertion
                                            failed."); }  }
for(String task:tasks2){
        try {
            System.out.println("Adding task "+task+" to unit 2.");
            cc.addNewTask(task, unit2);
        } catch (IncosistencyException ex) {
            System.err.println("ERROR: task insertion failed.");} }
```

IL MAIN

```
    for(int i=0; i<4; i++){
        try {
            System.out.println("Removing task "+cc.getFirstTask(unit1)+" from unit 1.");
            cc.removeCompletedTask(unit1);
        } catch (IncosistencyException ex) {
            System.err.println("ERROR: no tasks left.");
        } }
    for(int i=0; i<6; i++){
        try {
            System.out.println("Removing task "+cc.getFirstTask(unit2)+" from unit 2.");
            cc.removeCompletedTask(unit2);
        } catch (IncosistencyException ex) {
            System.err.println("ERROR: no tasks left.");
        }
    }
} }
```