APIs

String (Partial)

Methods

|  |  |  |
| --- | --- | --- |
| type | method | Description |
| char | charAt(int index) | The method returns the character in the given index |
| int | length() | The method returns the number of characters in this string instance. |
| String | substring(int start) | The method returns the string which starts at the index start and ends in the end of the string. |
| String | substring(int start, int end) | The method returns the string which starts at the index start and ends in the end index. (excluding it) |
| String | toLowerCase() | The method returns the same string but every capital letter is switched to lower case. |
| int | compareTo(String other) | The method returns 0 if the strings are equal, return positive number if this string should be before the other string (in lexicographic terms) returns negative if the other string string should be before this string. |

Object API (Partial)

Methods

|  |  |  |
| --- | --- | --- |
| Type | method | Description |
| String | toString() | Returns a textual presentation of the Object |
| boolean | equals(Object obj) | Compares two objects by their location in the memory |
| Class <?> | getClass() | Return runtime class of this object. |

LinkedList<E> API

Constructor:

|  |
| --- |
| method |
| LinkedList() |

Methods

|  |  |  |
| --- | --- | --- |
| type | method | Description |
| void | add(E element) | The method inserts the given element to the end of the list |
| void | add(E element,int index) | The method inserts the given element to the given index’ position. While 0 means add to the first non dummy element. And size means add to the last position. |
| E | remove(int index) | The method removes and returns element at the given index |
| E | get(int index) | Returns the element at the given index of this list. |
| int | size() | The method returns the number of elements from this list. |
| int | indexOf(E element) | The method returns the index of the element in the list if it does appear in this list instance if not returns -1. |
| boolean | contains (E element) | The method returns true if the element appears in this list instance if not returns false. |
| String | toString() | The method overrides the Object class’s toString method. Returns all the elements as consecutive string while using the toString method of every element. |
| ListIterator<E> | iterator() | The method returns a ListIterator of the elements in this list. Starting at the first position in this list. (Not dummy, actual first) |

ListIterator<E>

Methods

|  |  |  |
| --- | --- | --- |
| type | method | Description |
| boolean | hasNext() | The method returns true if the iterator has more elements. |
| E | next() | The method returns current element in the list and advances to the next one. |

Worker Class

public class Worker {

protected String name;

protected int salary;

// API

// Constructor

/\*\*

\* Constructs a new worker with the given name and salary, saves in

\* the field as lower case .

\* if the salary is negative or zero, an IllegalArgumentException is

\* thrown.

\* @param name - the name of the worker

\* @param salary - the salary of the worker

\*/

public Worker (String name, int salary){

**// Assume it is implemented and can be used**

}

// Getters

// Returns the name of the worker

public String getName (){

**// Assume it is implemented and can be used**

}

// Returns the salary of the worker

public int getSalary(){

**// Assume it is implemented and can be used**

}

/\*\*

\* returns true if the given object is a worker with the same name and salary,

\* false otherwise

\* it overrides the equals method in the Object class

\* @param obj - the object to compare to

\* @return true if the given object is a worker with the same name and salary,

\* false otherwise

\*/

public boolean equals(Object obj){

**// Assume it is implemented and can be used**

}

// Returns true if the worker is a manager, false otherwise

public boolean isManager(){

**// Assume it is implemented and can be used**

}

// Returns a string representation of the worker

public String toString(){

**// Assume it is implemented and can be used**

}

// Q4

public void raiseSalary(int amount){

**// TODO**

}

}

Manager Class

public class Manager extends Worker {

private LinkedList<Worker> workers;

/\*\*

\* Constructs a new manager with the given name and salary.

\* @param name - the name of the manager

\* @param salary - the salary of the manager

\*/

public Manager (String name, int salary){

**// Assume it is implemented and can be used**

}

/\*\*

\* Returns the list of workers of this manager.

\* @return the list of workers of this manager.

\*/

public LinkedList<Worker> getWorkers(){

**// Assume it is implemented and can be used**

}

/\*\*

\* Returns the array of workers of this manager.

\* @return the array of workers of this manager.

\*/

public Worker[] toArray(){

**// Assume it is implemented and can be used**

}

/\*\*

\* Returns true if this manager is a manager, false otherwise.

\* @return true if this manager is a manager, false otherwise.

\*/

public boolean isManager(){

**// Assume it is implemented and can be used**

}

/\*\*

\* Adds the given worker to the list of workers of this manager.   
 \* The worker should be added to the list of workers in the correct   
 \* lexicographic ordered location (by their name)

\* @param worker - the worker to add

\*/

public void addWorker(Worker worker){

**//TODO – related to Q9**

}

/\*\*

\* Removes the given worker from the list of workers of this manager.

\* @param worker - the worker to remove

\*/

public void removeWorker(Worker worker){

**//TODO – related to Q10**

}

/\*\*

\* Checks if the given worker is a worker of this manager or one of his sub-

\* managers.

\* @param worker - the worker to check

\* @return true if the given worker is a worker of this manager or one of his sub-

\* managers, false otherwise.

\*/

public boolean hasWorker (Worker worker){

**// Assume it is implemented and can be used**

}

// Q5

/\*\*

\* Constructs a new manager with the given name, salary and workers.

\* The workers are added to the list of workers of this manager.

\* @param name - the name of the manager

\* @param salary - the salary of the manager

\* @param workers - the workers to add to the list of workers of this manager

\*/

public Manager (String name, int salary, Worker[] workers){

**// TODO**

}

// Q6

/\*\*

\* Returns the total salaries of this manager and his workers.

\* If a worker is a manager, his total salaries are also included.

\* @return the total salaries of this manager and his workers.

\*/

public long getTotalSalaries(){

**// TODO**

}

// Q7

/\*\*

\* Returns true if the given object is a manager with the same name, salary and

\* workers as this manager

\* @param obj - the object to compare

\* @return true if the given object is a manager with the same name, salary and

\* workers as this manager.

\* @Override

\*/

public boolean equals(Object obj){

**//TODO**

}

}

Office Class

public class Office {

private Manager manager;

private String name;

private String location;

/\*\*

\* Constructs a new office with the given name, location and manager.

\* @param name - the name of the office

\* @param location - the location of the office

\* @param manager - the manager of the office

\*/

public Office(String name, String location, Manager manager) {

**// Assume it is implemented and can be used**

}

// Getters

/\*\*

\* Returns the name of the office.

\* @return - the name of the office.

\*/

public String getName() {

**// Assume it is implemented and can be used**

}

/\*\*

\* Returns the location of the office.

\* @return - the location of the office.

\*/

public String getLocation() {

**// Assume it is implemented and can be used**

}

/\*\*

\* Returns the manager of the office.

\* @return - the manager of the office.

\*/

public Manager getManager() {

**// Assume it is implemented and can be used**

}

// Q8

/\*\*

\* Returns the direct manager of the given worker.

\* @param worker - the worker to find his manager

\* @return the direct manager of the given worker.

\*/

public Manager getWorkersManager(Worker worker) {

**// TODO**

}

// Q9

/\*\*

\* Adds the given worker to the given manager.

\* @param worker - the worker to add

\* @param manager - the manager to add the worker to (the workers new

\* direct manager)

\* if the worker is already a worker of the office, do nothing.

\* if the manager is not a manager of the office, do nothing.

\*/

public void addWorkerToManager(Worker worker, Manager manager) {

**// TODO**

}

// Q10

/\*\*

\* Removes the given worker from the office.

\* if the worker is not a worker of the office, do nothing.

\* @param worker - the worker to remove

\*/

public void removeWorkerFromOffice (Worker worker) {

**// TODO**

}

// Q11

/\*\*

\* Returns the total salaries of all the workers in the office.

\* @return the total salaries of all the workers in the office.

\*/

public long getTotalSalaries() {

**// TODO**

}

// Q12

/\*\*

\* moves the given worker from his current manager to the given manager.

\* @param worker - the worker to move

\* @param newManager - the new manager of the worker

\*/

public void moveWorkerBetweenManagers(Worker worker, Manager newManager){

**// TODO**

}

}

EXAMPLE

public class TheOffice {

public static void main(String[] args) {

Manager michael = new Manager("Michael Scott", 150000);

Worker dwight = new Worker("Dwight Schrute", 80000);

Manager jim = new Manager("Jim Halpert", 80000);

Worker pam = new Worker("Pam Beesly", 60000);

Manager angela = new Manager("Angela Martin", 100000);

Worker kevin = new Worker("Kevin Malone", 60000);

Worker stanley = new Worker("Stanley Hudson", 60000);

Worker creed = new Worker("Creed Bratton", 60000);

Worker oscar = new Worker("Oscar Martinez", 60000);

Worker phyllis = new Worker("Phyllis Lapin", 60000);

Worker andy = new Worker("Andy Bernard", 60000);

Worker erin = new Worker("Erin Hannon", 60000);

Manager kelly = new Manager("Kelly Kapoor", 60000);

Manager darryl = new Manager("Darryl Philbin", 60000);

Worker toby = new Worker("Toby Flenderson", 60000);

Worker holly = new Worker("Holly Flax", 60000);

Manager jan = new Manager("Jan Levinson", 60000);

Worker roy = new Worker("Roy Anderson", 60000);

Worker ryan = new Worker("Ryan Howard", 60000);

Manager karen = new Manager("Karen Filippelli", 60000);

Worker intern = new Worker("Intern", 1);

Worker intern2 = new Worker("Intern2", 1);

michael.addWorker(dwight);

michael.addWorker(jim);

michael.addWorker(pam);

michael.addWorker(angela);

michael.addWorker(stanley);

michael.addWorker(creed);

michael.addWorker(phyllis);

michael.addWorker(andy);

michael.addWorker(erin);

michael.addWorker(kelly);

michael.addWorker(darryl);

michael.addWorker(toby);

michael.addWorker(holly);

michael.addWorker(ryan);

angela.addWorker(kevin);

darryl.addWorker(roy);

jim.addWorker(karen);

karen.addWorker(intern);

Office office = new Office("Dunder Mifflin", "Scranton, PA", michael);

System.out.println("\nPrints Q8\n");

System.out.println(office.getWorkersManager(ryan));

System.out.println(office.getWorkersManager(roy));

System.out.println(office.getWorkersManager(kevin));

System.out.println(office.getWorkersManager(intern));

System.out.println(office.getWorkersManager(jan));

System.out.println("\nPrints Q9\n");

System.out.println(angela.hasWorker(oscar));

office.addWorkerToManager(oscar, angela);

System.out.println(angela.hasWorker(oscar));

System.out.println(karen.hasWorker(intern2));

System.out.println(jim.hasWorker(intern2));

office.addWorkerToManager(intern2, karen);

System.out.println(karen.hasWorker(intern2));

System.out.println(jim.hasWorker(intern2));

System.out.println("\nPrints Q10\n");

System.out.println(karen.hasWorker(intern2));

System.out.println(jim.hasWorker(intern2));

office.removeWorkerFromManager(intern2, karen);

System.out.println(karen.hasWorker(intern2));

System.out.println(jim.hasWorker(intern2));

System.out.println("\nPrints Q11\n");

System.out.println(office.getTotalSalaries());

System.out.println("\nPrints Q12\n");

System.out.println(karen.hasWorker(intern));

System.out.println(darryl.hasWorker(intern));

office.moveWorkerBetweenManagers(intern, darryl);

System.out.println(karen.hasWorker(intern));

System.out.println(darryl.hasWorker(intern));

System.out.println(michael.hasWorker(ryan));

System.out.println(michael.getWorkers().contains(ryan));

System.out.println(kelly.hasWorker(ryan));

office.moveWorkerBetweenManagers(ryan, kelly);

System.out.println(michael.hasWorker(ryan));// should be true since

// kelly is still a worker of michael and ryan under kelly

System.out.println(michael.getWorkers().contains(ryan));// should be

// false since ryan is not directly under michael

System.out.println(kelly.hasWorker(ryan));

}

}

TheOffice Output

Prints Q8

Name: michael scott ,Salary:150000

Name: darryl philbin ,Salary:60000

Name: angela martin ,Salary:100000

Name: karen filippelli ,Salary:60000

null

Prints Q9

false

true

false

false

true

true

Prints Q10

true

true

false

false

Prints Q11

1250001

Prints Q12

true

false

false

true

true

true

false

true

false

true