Question 1: Document Word Frequency

a) The idea of my solution

The class Element we create with two fields. The first field is a word and the second field is a frequency. In the constructor we should insert the word and the frequency assign the value 1. The word variable is final and we can only get its value, but the frequency is not final, we can increase its value using the method increaseFrequency() and get its value.

The class Document we create with a constructor that accepts the file name to analyze, method that analyze the file, methods to return words in the different orders and also list of sorted frequencies, auxiliary private methods to insert new words in the right order and to sort the whole vector of elements and the setter method to change the file name. In this solution we use binary tree sort algorithm.

The class DocumentTest we create with the static method main and all auxiliary methods to test the Document class as described in the test plan.

b) Source code

- Document.java

```
package uebung8.question1;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
import java.util.Vector;

/** The class Document accepts an input file in the constructor and handle the
 * processing of the corresponding text file.
 *
 * @author Andrii Dzhyrma */
public class Document {

  // Private Members ////////////////////////////////////////////////////////

  // Fields ------------------------------------------------------------------
  private Vector<Element> elements;
  private String fileName;

  // Methods -----------------------------------------------------------------
  /* Adds the word to the elements vector using a binary search algorithm. */
  private void binarySearchInsertion(String word) {
    if (elements.size() == 0) {
      this.elements.add(new Element(word));
      return;
    }

    int left = 0;
    int right = this.elements.size();
    while (left < right) {
      int index = (left + right) / 2;
      int comparsion = (this.elements.get(index).getWord()
          .compareToIgnoreCase(word));
      if (comparsion == 0) {
        this.elements.get(index).increaseFrequency();
        return;
      } else if (comparsion > 0)
        right = index;
      else
        left = index + 1;
    }

    this.elements.add(left, new Element(word));
  }
```

```java
/* Sorts the elements using binary tree search algorithm. */
private void ascendingBinaryTreeSort() {
  if (this.elements == null)
    return;
  for (int i = 1; i < this.elements.size(); i++) {
    Element element = this.elements.get(i);
    int left = 0;
    int right = i;
    while (left < right) {
      int index = (left + right) / 2;
      int frequency1 = this.elements.get(index).getFrequency();
      int frequency2 = element.getFrequency();
      if (frequency1 == frequency2)
        left = right = index;
      else if (frequency1 > frequency2)
        right = index;
      else
        left = index + 1;
    }
    if (i != left) {
      this.elements.remove(i);
      this.elements.add(left, element);
    }
  }
}

// Public Members ////////////////////////////////////////////////////////////

// Constructors --------------------------------------------------------------
/** Initializes a Document object for the selected file.
 *
 * @param fileName - the file path */
public Document(String fileName) {
  this.fileName = fileName;
}

// Methods -------------------------------------------------------------------
/** Analyzes the current file. */
public void analyzeFile() {
  FileReader fileReader = null;
  try {
    // Open file and read contents
    fileReader = new FileReader(fileName);
    Scanner scanner = new Scanner(fileReader);
    this.elements = new Vector<Element>();
    while (scanner.hasNext())
      binarySearchInsertion(scanner.next());
    scanner.close();
    fileReader.close();
    ascendingBinaryTreeSort();
  } catch (FileNotFoundException e) {
    System.err.println("File not found.");
  } catch (IOException e) {
    System.err.println("File not closed.");
  }
}

/** Returns unique words sorted by an alphabet.
 *
 * @return - the array of words if the list of elements is not equal to null.
 *         Otherwise null */
public String[] getWords() {
  if (this.elements == null)
    return null;
  String[] result = new String[this.elements.size()];
  for (int i = 0; i < this.elements.size(); i++)
    result[i] = this.elements.get(i).getWord();
  return result;
}

/** Returns unique words sorted by a frequency.
 *
 * @param isAscending - the order.
 * @return - the array of words if the list of elements is not equal to null.
 *         Otherwise null */
public String[] getWordList(boolean isAscending) {
  if (this.elements == null)
    return null;
```

```
    int size = this.elements.size();
    String[] result = new String[size];
    for (int i = 0; i < size; i++)
      result[isAscending ? i : (size - 1 - i)] = this.elements.get(i).getWord();
    return result;
  }

  /** Returns the sorted array of frequencies.
   *
   * @param isAscending - the order.
   * @return - the array of frequencies if the list of elements is not equal to
   *         null. Otherwise null */
  public int[] getFrequencyList(boolean isAscending) {
    if (this.elements == null)
      return null;
    int size = this.elements.size();
    int[] result = new int[size];
    for (int i = 0; i < size; i++)
      result[isAscending ? i : (size - 1 - i)] = this.elements.get(i)
          .getFrequency();
    return result;
  }

  /** Sets the file name to be processed
   *
   * @param fileName - the file name */
  public void setFileName(String fileName) {
    this.fileName = fileName;
  }
}
```

- Element.java

```
package uebung8.question1;

/** The class Element contains two features: a word and a frequency counter of
 * the word.
 *
 * @author Andrii Dzhyrma */
public class Element {

  // Private Members ////////////////////////////////////////////////////////

  // Fields ------------------------------------------------------------------
  private final String word;
  private int frequency = 1;

  // Public Members /////////////////////////////////////////////////////////

  // Constructors ------------------------------------------------------------
  /** Initializes the element with a word.
   *
   * @param word - the word */
  public Element(String word) {
    this.word = word;
  }

  // Methods -----------------------------------------------------------------
  /** Returns the frequency of the current word.
   *
   * @return - the current frequency */
  public int getFrequency() {
    return this.frequency;
  }

  /** Increases the frequency of the word by one. */
  public void increaseFrequency() {
    this.frequency++;
  }

  /** Returns the word from the current element
   *
   * @return - the word */
  public String getWord() {
    return this.word;
  }
}
```

- DocumentTest.java

```java
package uebung8.question1;

/** The DocumentTest class tests an existed class Document
 *
 * @author Andrii Dzhyrma */
public class DocumentTest {
  public static void main(String[] args) {
    System.out.println("Test #1");
    Document document1 = new Document("some_file.txt");
    test(document1);
    System.out.println("\nTest #2");
    Document document2 = new Document("alltitles.txt");
    test(document2);
  }

  public static void test(Document document) {
    document.analyzeFile();
    System.out.println("===============printWords()======================");
    printWords(document);
    System.out.println("===============printWordList(true)==============");
    printWordList(document,true);
    System.out.println("===============printWordList(false)=============");
    printWordList(document,false);
  }

  public static void printWords(Document document) {
    String[] words = document.getWords();
    if (words != null)
      for (String word : words)
        System.out.println(word);
    else
      System.out.println("printWords error!");
  }

  public static void printWordList(Document document, boolean isAscending) {
    String[] words = document.getWordList(isAscending);
    int[] frequencies = document.getFrequencyList(isAscending);
    if (words != null || frequencies != null)
      for (int i = 0; i < words.length; i++)
        System.out.println(words[i] + " " + frequencies[i]);
    else
      System.out.println("printWordList error!");
  }
}
```

a) Test plan

#	Description	Expected output
1	Test the all methods with the file that does not exist	**An error**
2	Test the all methods with the file "alltitles.txt"	**A correspondent reaction as described in the requirements**

b) The output of the program

```
Test #1
File not found.
printWords()
printWords error!
printWordList(true)
printWordList error!
printWordList(false)
printWordList error!

Test #2
printWords()
activates
activity
aggresome
along
american
```

```
printWordList(true)
activates 1
activity 1
aggresome 1
along 1
american 1
analysis 1
aneuploidy 1
animals 1
antibodies 1
antimigratory 1
area 1
as 1
at 1
atomic 1
b 1
```

```
printWordList(false)
of 46
the 37
in 15
and 14
by 8
a 7
protein 5
c 4
on 4
regulates 4
to 4
from 4
antigen 3
between 3
binding 3
```

analysis	bacteriophage 1	receptor 3
aneuploidy	basic 1	is 3
animals	bacterial 1	interaction 3
antibodies	bioinorganic 1	for 3
antimigratory	biology 1	rna 3
area	boundary 1	archaeology 2
as	brucella 1	control 2
at	branching 1	beta 2
atomic	caenorhabditis 1	development 2
b	capsid 1	directed 2
bacteriophage	caribou 1	dorsal 2
basic	central 1	g 2
bacterial	channel 1	its 2
bioinorganic	characterization 1	lambda 2
biology	choice 1	like 2
boundary	class 1	kinase 2
brucella	colloids 1	mhc 2
branching	complex 1	helix 2
caenorhabditis	conservation 1	inhibits 2
capsid	continuously 1	neuronal 2
caribou	cooperate 1	induction 2
central	cord 1	s 2
channel	coupled 1	role 2
characterization	critique 1	specific 2
choice	cro 1	structural 2
class	daf 1	stat 2
colloids	decisions 1	target 2
complex	depletion 1	ii 2
conservation	differentiation 1	identity 2
continuously	discovery 1	eukaryote 2
cooperate	dispersion 1	deep 2
cord	dna 1	culture 2
coupled	domain 1	cell 2
critique	during 1	barbados 2
cro	early 1	assembly 2
daf	elegans 1	activated 1
decisions	engineering 1	adenovirus 1
depletion	essential 1	allow 1
differentiation	establishes 1	alpha 1
discovery	extension 1	an 1
dispersion	females 1	animal 1
dna	flap 1	anomalous 1
domain	fluctuations 1	aring 1
during	formation 1	ataxin 1
early	free 1	axis 1
elegans	gamma 1	biological 1
engineering	global 1	bisphosphate 1
essential	helicobacter 1	cadmium 1
establishes	helium 1	can 1
extension	hematopoietic 1	cells 1
females	high 1	certain 1
flap	hud 1	channels 1
fluctuations	hypothesis 1	chemistry 1
formation	human 1	chromosome 1
free	immunity 1	coli 1
gamma	indians 1	comb 1
global	induced 1	conformational 1
helicobacter	inhibit 1	consistently 1
helium	insights 1	copper 1
hematopoietic	into 1	core 1
high	intron 1	coupling 1
hud	inwardly 1	cyclooxygenase 1
hypothesis	introns 1	defection 1
human	jnk 1	disease 1
immunity	juxtaposition 1	divergence 1
indians	k 1	drosophila 1
induced	l 1	dynamical 1
inhibit	lack 1	effector 1
insights	late 1	elongation 1
into	level 1	escherichia 1
intron	ligands 1	experiments 1
inwardly	linked 1	factor 1
introns	local 1	filament 1
jnk	loop 1	forkhead 1
juxtaposition	macrophages 1	genetic 1
k	mammary 1	growth 1
l	mate 1	heliotropism 1
lack	mechanisms 1	heme 1

late
level
ligands
linked
local
loop
macrophages
mammary
mate
mechanisms
mediating
met
microcosms
model
modulate
monoubiquitination
multiple
myc
nascent
necessary
network
neurodegenerative
neutral
ninth
north
occupying
oceans
olig
papillomavirus
pattern
phenomena
phosphatidylinositol
phosphorylated
phytoplankton
plate
polymerase
postnatal
poxvirus
presentation
programming
prophage
proteins
proton
provision
pylori
radial
receptors
region
regulated
resolution
reverse
revision
richness
ribosome
romk
satellite
secretion
self
sigma
simulation
smooth
solar
species
spinal
src
stem
stimulated
structure
subunits
superfluid
synaptonemal
synthesis
system
t
term
tgf
through
toll
mediating 1
met 1
microcosms 1
model 1
modulate 1
monoubiquitination 1
multiple 1
myc 1
nascent 1
necessary 1
network 1
neurodegenerative 1
neutral 1
ninth 1
north 1
occupying 1
oceans 1
olig 1
papillomavirus 1
pattern 1
phenomena 1
phosphatidylinositol 1
phosphorylated 1
phytoplankton 1
plate 1
polymerase 1
postnatal 1
poxvirus 1
presentation 1
programming 1
prophage 1
proteins 1
proton 1
provision 1
pylori 1
radial 1
receptors 1
region 1
regulated 1
resolution 1
reverse 1
revision 1
richness 1
ribosome 1
romk 1
satellite 1
secretion 1
self 1
sigma 1
simulation 1
smooth 1
solar 1
species 1
spinal 1
src 1
stem 1
stimulated 1
structure 1
subunits 1
superfluid 1
synaptonemal 1
synthesis 1
system 1
t 1
term 1
tgf 1
through 1
toll 1
transcriptional 1
translocation 1
trichomonas 1
trunk 1
tumorigenesis 1
type 1
tyrosine 1
unit 1
vaccine 1
vascular 1
homology 1
il 1
infectivity 1
intramolecular 1
iv 1
jupiter 1
kinetic 1
laboratory 1
lethal 1
lifespan 1
living 1
long 1
m 1
major 1
manner 1
maturation 1
mediated 1
microarray 1
micrometer 1
modified 1
modulating 1
mouse 1
muscle 1
mycobacterium 1
nebulae 1
nemo 1
neural 1
neutralizing 1
normal 1
nuclear 1
other 1
parkinson 1
peptide 1
phoradendron 1
phospholipase 1
phosphorylation 1
plants 1
polyglutamine 1
pooling 1
potent 1
predictably 1
production 1
properties 1
proteolysis 1
prototypes 1
pten 1
quantitative 1
rectifying 1
relation 1
retina 1
reversible 1
samples 1
scales 1
segmental 1
ser 1
signaling 1
sized 1
smpd 1
sphingomyelinase 1
spliceosomal 1
stickleback 1
structures 1
suis 1
suppression 1
syndromes 1
synuclein 1
tbx 1
terminal 1
transcription 1
transitions 1
transverse 1
trisomy 1
tuberculosis 1
two 1
types 1
understanding 1
utility 1
vaginalis 1

transcriptional	ventral 1	velocities 1
translocation	viral 1	viable 1
trichomonas	vivo 1	virb 1
trunk	with 1	weight 1
tumorigenesis	world 1	within 1
type	zebrafish 1	zebrafish 1
tyrosine	within 1	world 1
unit	weight 1	with 1
vaccine	virb 1	vivo 1
vascular	viable 1	viral 1
ventral	velocities 1	ventral 1
viral	vaginalis 1	vascular 1
vivo	utility 1	vaccine 1
with	understanding 1	unit 1
world	types 1	tyrosine 1
zebrafish	two 1	type 1
within	tuberculosis 1	tumorigenesis 1
weight	trisomy 1	trunk 1
virb	transverse 1	trichomonas 1
viable	transitions 1	translocation 1
velocities	transcription 1	transcriptional 1
vaginalis	terminal 1	toll 1
utility	tbx 1	through 1
understanding	synuclein 1	tgf 1
types	syndromes 1	term 1
two	suppression 1	t 1
tuberculosis	suis 1	system 1
trisomy	structures 1	synthesis 1
transverse	stickleback 1	synaptonemal 1
transitions	spliceosomal 1	superfluid 1
transcription	sphingomyelinase 1	subunits 1
terminal	smpd 1	structure 1
tbx	sized 1	stimulated 1
synuclein	signaling 1	stem 1
syndromes	ser 1	src 1
suppression	segmental 1	spinal 1
suis	scales 1	species 1
structures	samples 1	solar 1
stickleback	reversible 1	smooth 1
spliceosomal	retina 1	simulation 1
sphingomyelinase	relation 1	sigma 1
smpd	rectifying 1	self 1
sized	quantitative 1	secretion 1
signaling	pten 1	satellite 1
ser	prototypes 1	romk 1
segmental	proteolysis 1	ribosome 1
scales	properties 1	richness 1
samples	production 1	revision 1
reversible	predictably 1	reverse 1
retina	potent 1	resolution 1
relation	pooling 1	regulated 1
rectifying	polyglutamine 1	region 1
quantitative	plants 1	receptors 1
pten	phosphorylation 1	radial 1
prototypes	phospholipase 1	pylori 1
proteolysis	phoradendron 1	provision 1
properties	peptide 1	proton 1
production	parkinson 1	proteins 1
predictably	other 1	prophage 1
potent	nuclear 1	programming 1
pooling	normal 1	presentation 1
polyglutamine	neutralizing 1	poxvirus 1
plants	neural 1	postnatal 1
phosphorylation	nemo 1	polymerase 1
phospholipase	nebulae 1	plate 1
phoradendron	mycobacterium 1	phytoplankton 1
peptide	muscle 1	phosphorylated 1
parkinson	mouse 1	phosphatidylinositol 1
other	modulating 1	phenomena 1
nuclear	modified 1	pattern 1
normal	micrometer 1	papillomavirus 1
neutralizing	microarray 1	olig 1
neural	mediated 1	oceans 1
nemo	maturation 1	occupying 1
nebulae	manner 1	north 1
mycobacterium	major 1	ninth 1
muscle	m 1	neutral 1
mouse	long 1	neurodegenerative 1

modulating	living 1	network 1
modified	lifespan 1	necessary 1
micrometer	lethal 1	nascent 1
microarray	laboratory 1	myc 1
mediated	kinetic 1	multiple 1
maturation	jupiter 1	monoubiquitination 1
manner	iv 1	modulate 1
major	intramolecular 1	model 1
m	infectivity 1	microcosms 1
long	il 1	met 1
living	homology 1	mediating 1
lifespan	heme 1	mechanisms 1
lethal	heliotropism 1	mate 1
laboratory	growth 1	mammary 1
kinetic	genetic 1	macrophages 1
jupiter	forkhead 1	loop 1
iv	filament 1	local 1
intramolecular	factor 1	linked 1
infectivity	experiments 1	ligands 1
il	escherichia 1	level 1
homology	elongation 1	late 1
heme	effector 1	lack 1
heliotropism	dynamical 1	l 1
growth	drosophila 1	k 1
genetic	divergence 1	juxtaposition 1
forkhead	disease 1	jnk 1
filament	defection 1	introns 1
factor	cyclooxygenase 1	inwardly 1
experiments	coupling 1	intron 1
escherichia	core 1	into 1
elongation	copper 1	insights 1
effector	consistently 1	inhibit 1
dynamical	conformational 1	induced 1
drosophila	comb 1	indians 1
divergence	coli 1	immunity 1
disease	chromosome 1	human 1
defection	chemistry 1	hypothesis 1
cyclooxygenase	channels 1	hud 1
coupling	certain 1	high 1
core	cells 1	hematopoietic 1
copper	can 1	helium 1
consistently	cadmium 1	helicobacter 1
conformational	bisphosphate 1	global 1
comb	biological 1	gamma 1
coli	axis 1	free 1
chromosome	ataxin 1	formation 1
chemistry	aring 1	fluctuations 1
channels	anomalous 1	flap 1
certain	animal 1	females 1
cells	an 1	extension 1
can	alpha 1	establishes 1
cadmium	allow 1	essential 1
bisphosphate	adenovirus 1	engineering 1
biological	activated 1	elegans 1
axis	assembly 2	early 1
ataxin	barbados 2	during 1
aring	cell 2	domain 1
anomalous	culture 2	dna 1
animal	deep 2	dispersion 1
an	eukaryote 2	discovery 1
alpha	identity 2	differentiation 1
allow	ii 2	depletion 1
adenovirus	target 2	decisions 1
activated	stat 2	daf 1
assembly	structural 2	cro 1
barbados	specific 2	critique 1
cell	role 2	coupled 1
culture	s 2	cord 1
deep	induction 2	cooperate 1
eukaryote	neuronal 2	continuously 1
identity	inhibits 2	conservation 1
ii	helix 2	complex 1
target	mhc 2	colloids 1
stat	kinase 2	class 1
structural	like 2	choice 1
specific	lambda 2	characterization 1
role	its 2	channel 1
s	g 2	central 1

induction neuronal inhibits helix mhc kinase like lambda its g dorsal directed development beta control archaeology rna for interaction is receptor binding between antigen from to regulates on c protein a by and in the of	dorsal 2 directed 2 development 2 beta 2 control 2 archaeology 2 rna 3 for 3 interaction 3 is 3 receptor 3 binding 3 between 3 antigen 3 from 4 to 4 regulates 4 on 4 c 4 protein 5 a 7 by 8 and 14 in 15 the 37 of 46	caribou 1 capsid 1 caenorhabditis 1 branching 1 brucella 1 boundary 1 biology 1 bioinorganic 1 bacterial 1 basic 1 bacteriophage 1 b 1 atomic 1 at 1 as 1 area 1 antimigratory 1 antibodies 1 animals 1 aneuploidy 1 analysis 1 american 1 along 1 aggresome 1 activity 1 activates 1

Question 2a: Company Employee List - Encapsulation

a) The idea of my solution

The class Employee we create with four private fields (an employee number, first name, last name and a salary), a constructor that generates the employee number and assigns the first name, last name and the salary, getters and setters for each field except setter for the employee number and toString() method as described in the requirements.

The class EmployeeNode we create with a one final field of the type Employee (data of the node) and two default fields (next and previous nodes). The constructor accepts only the employee.

The class EmployeeList we create just with two default fields of the type EmployeeNode: head and tail.

Now we have double linked list and we can use it in the class Company. So in this class we create a one private field list, a constructor that initializes this list, and next methods:

- addEmployee(Employee employee) – adds a new employee to the list of employees if the employee is correct (not null, the salary is not less than zero, first and last names are not nulls and the list does not contain this employee already). To find the right place in the list we go starting from the head until we reach the tail or the employee with the salary

which greater. Check for four cases: the list is empty, have to put in the head, have to put in the tail and have to put in somewhere inside.

- containsEmployee(Employee employee) – returns true if the list of employees contains the given employee. This method is auxiliary for the addEmployee.
- removeEmployee(Employee employee) – removes the employee from the list if it was found in the list. Check for three cases: the employee is in the head of the list, in the tail or somewhere inside.
- getEmployeeListWithSalaryGreaterThan(double salary, boolean isAscendingOrder) – goes through the list of employees and adds to the result all employees with a salary which greater than input one. Use the isAscendingOrder variable to know from the head or from the tail we should start to search.
- toString(), toDescendingString() - returns the list of employees sorted by the ascending and the descending order correspondently. Returns "The list with employees is empty." String if the list is empty.

b) Source code

- Company.java

```
package uebung8.question2a;

import java.util.Vector;

/** The class Company gives methods to add, remove and to search employees or to
 * print list of added employees.
 *
 * @author Andrii Dzhyrma */
public class Company {

  // Private Members /////////////////////////////////////////////////////////

  // Fields ------------------------------------------------------------------
  // double linked list of the employees
  private final EmployeeList list;

  // Public Members //////////////////////////////////////////////////////////

  // Constructors ------------------------------------------------------------
  /** Initializes a Company object. */
  public Company() {
    list = new EmployeeList();
  }

  // Methods -----------------------------------------------------------------
  /** Adds a new employee to the list of employees in the company.
   *
   * @param employee - the employee to be added
   * @return true if the employee was added successfully */
  public boolean addEmployee(Employee employee) {
    // check whether the new employee is correct and not consists in the list of
    // all employees
    if (employee == null || employee.getSalary() < 0
        || employee.getFirstName() == null || employee.getLastName() == null
        || containsEmployee(employee))
      return false;
    // create a new employee node for the list
    EmployeeNode employeeNode = new EmployeeNode(employee);
    // if the list is empty
    if (this.list.head == null) {
      this.list.head = this.list.tail = employeeNode;
      return true;
    }
    // search for the right place in the list
    EmployeeNode currentEmployeeNode = this.list.head;
    while (currentEmployeeNode != null
        && currentEmployeeNode.employee.getEmployeeNum() < employee
            .getEmployeeNum())
      currentEmployeeNode = currentEmployeeNode.next;
```

```
    // we reached the tail position
    if (currentEmployeeNode == null) {
      this.list.tail.next = employeeNode;
      employeeNode.next = null;
      employeeNode.prev = this.list.tail;
      this.list.tail = employeeNode;
      // we stopped on the head position
    } else if (currentEmployeeNode.prev == null) {
      this.list.head = employeeNode;
      employeeNode.next = currentEmployeeNode;
      employeeNode.prev = null;
      currentEmployeeNode.prev = employeeNode;
      // somewhere in the middle of the list
    } else {
      employeeNode.prev = currentEmployeeNode.prev;
      employeeNode.next = currentEmployeeNode;
      currentEmployeeNode.prev = currentEmployeeNode.prev.next = employeeNode;
    }
    return true;
  }

  /** Checks if the given employee is in the list.
   *
   * @param employee - the employee
   * @return true if the list contains the given employee */
  public boolean containsEmployee(Employee employee) {
    // check for the correct employee
    if (employee == null)
      return false;
    // search for it
    EmployeeNode currentEmployeeNode = this.list.head;
    while (currentEmployeeNode != null
        && currentEmployeeNode.employee != employee)
      currentEmployeeNode = currentEmployeeNode.next;
    // return true if we did not reach the tail
    return currentEmployeeNode != null;
  }

  /** Returns a list of employees with the salary greater than given in the
   * represented order.
   *
   * @param salary - the minimal salary
   * @param isAscendingOrder - the given order (true for an ascending, false for
   *          a descending)
   * @return the vector of employees found in the given order */
  public Vector<Employee> getEmployeeListWithSalaryGreaterThan(double salary,
      boolean isAscendingOrder) {
    // create the resulted variable
    Vector<Employee> employees = new Vector<Employee>();
    // search for the employees
    EmployeeNode currentEmployeeNode = isAscendingOrder ? this.list.head
        : this.list.tail;
    while (currentEmployeeNode != null) {
      if (currentEmployeeNode.employee.getSalary() > salary)
        // add a found employee to the resulted vector
        employees.add(currentEmployeeNode.employee);
      currentEmployeeNode = isAscendingOrder ? currentEmployeeNode.next
          : currentEmployeeNode.prev;
    }
    return employees;
  }

  /** Returns a list of employees with the given last name..
   *
   * @param lastName - the last name to search for
   * @return the vector of employees found */
  public Vector<Employee> getEmployees(String lastName) {
    // create the resulted variable
    Vector<Employee> employees = new Vector<Employee>();
    // check for the correct last name
    if (lastName == null)
      return employees;
    // search for the all employees with the given last name
    EmployeeNode currentEmployeeNode = this.list.head;
    while (currentEmployeeNode != null) {
      String employeeLastName = currentEmployeeNode.employee.getLastName();
      if (employeeLastName != null
          && employeeLastName.compareToIgnoreCase(lastName) == 0)
```

```
        // add the found employee to the vector
        employees.add(currentEmployeeNode.employee);
      currentEmployeeNode = currentEmployeeNode.next;
    }
    return employees;
  }

  /** Removes the given employee from the list.
   *
   * @param employee - the employee
   * @return true if the employee was successfully removed */
  public boolean removeEmployee(Employee employee) {
    // check for the correct employee
    if (employee == null)
      return false;
    // search for it in the list
    EmployeeNode currentEmployeeNode = this.list.head;
    while (currentEmployeeNode != null
        && currentEmployeeNode.employee != employee)
      currentEmployeeNode = currentEmployeeNode.next;
    // did not find it
    if (currentEmployeeNode == null)
      return false;
    // found it in the head
    if (currentEmployeeNode.prev == null) {
      this.list.head = currentEmployeeNode.next;
      this.list.head.prev = null;
      return true;
    }
    // found it in the tail
    if (currentEmployeeNode.next == null) {
      this.list.tail = currentEmployeeNode.prev;
      this.list.tail.next = null;
      return true;
    }
    // found it somewhere in the list
    currentEmployeeNode.prev.next = currentEmployeeNode.next;
    currentEmployeeNode.next.prev = currentEmployeeNode.prev;
    return true;
  }

  /** Returns the list of employees sorted by the descending order.
   *
   * @return the sorted list represented as a String variable */
  public String toDescendingString() {
    // create the resulted string
    StringBuilder result = new StringBuilder();
    // go through the list and add their toString() result to the our resulted
    // string
    EmployeeNode currentEmployeeNode = this.list.tail;
    while (currentEmployeeNode != null) {
      result.append(currentEmployeeNode.employee);
      result.append("\n");
      currentEmployeeNode = currentEmployeeNode.prev;
    }
    if (result.length() == 0)
      return "The list with employees is empty.";
    return result.toString();
  }

  /* Returns the list of employees sorted by the ascending order. */
  @Override
  public String toString() {
    // create the resulted string
    StringBuilder result = new StringBuilder();
    // go through the list and add their toString() result to the our resulted
    // string
    EmployeeNode currentEmployeeNode = this.list.head;
    while (currentEmployeeNode != null) {
      result.append(currentEmployeeNode.employee);
      result.append("\n");
      currentEmployeeNode = currentEmployeeNode.next;
    }
    if (result.length() == 0)
      return "The list with employees is empty.";
    return result.toString();
  }
}
```

- Employee.java

```java
package uebung8.question2a;

import java.util.Random;

/** The class Employee stores and manages the data about an employee.
 *
 * @author Andrii Dzhyrma */
public class Employee {

  // Private Members ////////////////////////////////////////////////////////

  // Fields ----------------------------------------------------------------
  private long employeeNum;
  private String firstName;
  private String lastName;
  private double salary;

  // Public Members /////////////////////////////////////////////////////////

  // Constructors ----------------------------------------------------------
  /** Initializes a Employee using his first name, last name and the salary.
   * Generates a random employee number.
   *
   * @param firstName - the first name
   * @param lastName - the last name
   * @param salary - the salary */
  public Employee(String firstName, String lastName, double salary) {
    this.employeeNum = (long) (new Random().nextDouble() * 8999999999d) + 1000000000;
    this.firstName = firstName;
    this.lastName = lastName;
    this.salary = Math.round(salary * 100) / 100d;
  }

  // Methods ---------------------------------------------------------------
  /** @return the employee number */
  public final long getEmployeeNum() {
    return this.employeeNum;
  }

  /** @return the first name of the employee */
  public final String getFirstName() {
    return this.firstName;
  }

  /** @return the last name of the employee */
  public final String getLastName() {
    return this.lastName;
  }

  /** @return the salary of the employee */
  public final double getSalary() {
    return this.salary;
  }

  /** Sets a new first name for the employee.
   *
   * @param firstName - the new first name */
  public final void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  /** Sets a new last name for the employee.
   *
   * @param lastName - the new last name */
  public final void setLastName(String lastName) {
    this.lastName = lastName;
  }

  /** Sets a new salary for the employee.
   *
   * @param salary - the new salary */
  public final void setSalary(int salary) {
    this.salary = salary;
  }
```

```
  /* Returns the string representation of the employee. */
  @Override
  public String toString() {
    return "#" + this.employeeNum + ", " + this.firstName + " " + this.lastName
        + " [salary=" + this.salary + "]";
  }
}
```

- EmployeeNode.java

```
package uebung8.question2a;

/** The class EmployeeNode represents a node for the double linked list with
 * references for the next and the previous nodes and the data variable.
 *
 * @author Andrii Dzhyrma */
public class EmployeeNode {
  final Employee employee;
  EmployeeNode next, prev;

  /** Initializes a EmployeeNode with the given Employee object.
   *
   * @param employee - the employee */
  public EmployeeNode(Employee employee) {
    this.employee = employee;
  }
}
```

- EmployeeList.java

```
package uebung8.question2a;

/** The class EmployeeList stores a head and a tail of the representing double
 * linked list.
 *
 * @author Andrii Dzhyrma */
public class EmployeeList {
  EmployeeNode head, tail;
}
```

- CompanyTest.java

```
package uebung8.question2a;

import java.util.Vector;

/** The class CompanyTest tests the class Company for errors.
 *
 * @author Andrii Dzhyrma */
public class CompanyTest {
  public static void main(String[] args) throws CloneNotSupportedException {
    Employee employee1 = new Employee(null, null, 0);
    Employee employee2 = new Employee("John", "Smith", -100);
    Employee employee3 = new Employee("Michael", "Smith", 500);
    Employee employee4 = new Employee("Andrii", "Dzhyrma", 3000);
    Employee employee5 = new Employee("Sarah", "Connor", 19999.99999);
    Employee employee6 = new Employee("John", "Connor", 1999.49999);
    Employee employee7 = new Employee("Anonymous", "Incognito", 999999999);
    Company company = new Company();
    System.out.println("==============company.toString()=====================================");
    System.out.println(company);
    System.out.println("==============company.toDescendingString()============================");
    System.out.println(company.toDescendingString());
    System.out.println("==============company.addEmployee()===================================");
    addEmployee(company, null);
    addEmployee(company, employee1);
    addEmployee(company, employee2);
    addEmployee(company, employee3);
    addEmployee(company, employee4);
    addEmployee(company, employee5);
    addEmployee(company, employee6);
    System.out.println("==============employee2.getEmployeeNum()==============================");
    System.out.println(employee2.getEmployeeNum());
    System.out.println("==============employee2.getFirstName()================================");
    System.out.println(employee2.getFirstName());
    System.out.println("==============employee2.getLastName()=================================");
    System.out.println(employee2.getLastName());
```

```
    System.out.println("==============employee2.getSalary()====================================");
    System.out.println(employee2.getSalary());
    System.out.println("==============employee2.setSalary(100);company.addEmployee(employee2)==");
    employee2.setSalary(100);
    addEmployee(company, employee2);
    System.out.println("==============company.toString()=======================================");
    System.out.println(company);
    System.out.println("==============company.toDescendingString()=============================");
    System.out.println(company.toDescendingString());
    System.out.println("==============company.getEmployeeListWithSalaryGreaterThan(1000, true)=");
    getEmployeeListWithSalaryGreaterThan(company, 1000, true);
    System.out.println("==============company.getEmployeeListWithSalaryGreaterThan(1000, false)");
    getEmployeeListWithSalaryGreaterThan(company, 1000, false);
    System.out.println("==============company.getEmployeeListWithSalaryGreaterThan(-100, true)=");
    getEmployeeListWithSalaryGreaterThan(company, -100, true);
    System.out.println("==============company.getEmployees()===================================");
    getEmployees(company, "Dzhyrma");
    getEmployees(company, "Connor");
    getEmployees(company, "");
    getEmployees(company, null);
    System.out.println("==============company.removeEmployee()=================================");
    removeEmployee(company, employee7);
    removeEmployee(company, employee1);
    removeEmployee(company, employee3);
    removeEmployee(company, employee3);
    removeEmployee(company, null);
  }

  private static void addEmployee(Company company, Employee employee) {
    if (company != null && company.addEmployee(employee))
      System.out.println("Employee [" + employee
          + "] was successfully added to the company!");
    else
      System.out.println("Employee [" + employee
          + "] was not added to the company!");
  }

  private static void getEmployeeListWithSalaryGreaterThan(Company company, double minSalary, boolean
isAscendingOrder) {
    if (company != null) {
      Vector<Employee> result = company.getEmployeeListWithSalaryGreaterThan(minSalary, isAscendingOrder);
      System.out.println(result);
    }
  }

  private static void getEmployees(Company company, String lastName) {
    if (company != null) {
      Vector<Employee> result = company.getEmployees(lastName);
      System.out.println("getEmployees for the last name '" + lastName + "' returned the next result:");
      System.out.println(result);
    }
  }

  private static void removeEmployee(Company company, Employee employee) {
    if (company != null && company.removeEmployee(employee))
      System.out.println("Employee [" + employee
          + "] was successfully removed from the company!");
    else
      System.out.println("Employee [" + employee
          + "] was not removed from the company!");
  }
}
```

c) Test plan

#	Description	Expected output
1	Test printing methods of the class Company without employees and with	**If there are no employees in the company, print correspondent string, otherwise string with employees**
2	Test addEmployee() method	**Do not add incorrect employees**
3	Test getters and setters of the class Employee	**Stable work for the getters and the setters**
4	Test getEmployeeListWithSalaryGreaterThan() method	**A correspondent reaction as described in the requirements**
5	Test getEmployees() method	**A correspondent reaction as described in the requirements**
6	Test removeEmployee() method	**Remove employee if it exists in the company**

d) The output of the program

Output	Comments
==========company.toString()==========	Test #1
The list with employees is empty.	
==========company.toDescendingString()==========	
The list with employees is empty.	
==========company.addEmployee()==========	Test #2
Employee [null] was not added to the company!	
Employee [#9808208635, null null [salary=0.0]] was not added to the company!	
Employee [#4627665010, John Smith [salary=-100.0]] was not added to the company!	
Employee [#6854593986, Michael Smith [salary=500.0]] was successfully added to the company!	
Employee [#8964175566, Andrii Dzhyrma [salary=3000.0]] was successfully added to the company!	
Employee [#4350853500, Sarah Connor [salary=20000.0]] was successfully added to the company!	
Employee [#6306304240, John Connor [salary=1999.5]] was successfully added to the company!	
==========employee2.getEmployeeNum()==========	Test #3
4627665010	
==========employee2.getFirstName()==========	
John	
==========employee2.getLastName()==========	
Smith	
==========employee2.getSalary()==========	
-100.0	
==========employee2.setSalary(100);company.addEmployee(employee2)==	Test #3 + #2
Employee [#4627665010, John Smith [salary=100.0]] was successfully added to the company!	
==========company.toString()==========	Test #1
#4350853500, Sarah Connor [salary=20000.0]	
#4627665010, John Smith [salary=100.0]	
#6306304240, John Connor [salary=1999.5]	
#6854593986, Michael Smith [salary=500.0]	
#8964175566, Andrii Dzhyrma [salary=3000.0]	

==========company.toDescendingString()========== #8964175566, Andrii Dzhyrma [salary=3000.0] #6854593986, Michael Smith [salary=500.0] #6306304240, John Connor [salary=1999.5] #4627665010, John Smith [salary=100.0] #4350853500, Sarah Connor [salary=20000.0]	
==========company.getEmployeeListWithSalaryGreaterThan(1000, true)= [#4350853500, Sarah Connor [salary=20000.0], #6306304240, John Connor [salary=1999.5], #8964175566, Andrii Dzhyrma [salary=3000.0]] ==========company.getEmployeeListWithSalaryGreaterThan(1000, false) [#8964175566, Andrii Dzhyrma [salary=3000.0], #6306304240, John Connor [salary=1999.5], #4350853500, Sarah Connor [salary=20000.0]] ==========company.getEmployeeListWithSalaryGreaterThan(-100, true)= [#4350853500, Sarah Connor [salary=20000.0], #4627665010, John Smith [salary=100.0], #6306304240, John Connor [salary=1999.5], #6854593986, Michael Smith [salary=500.0], #8964175566, Andrii Dzhyrma [salary=3000.0]]	Test #4
==========company.getEmployees()========== getEmployees for the last name 'Dzhyrma' returned the next result: [#8964175566, Andrii Dzhyrma [salary=3000.0]] getEmployees for the last name 'Connor' returned the next result: [#4350853500, Sarah Connor [salary=20000.0], #6306304240, John Connor [salary=1999.5]] getEmployees for the last name '' returned the next result: [] getEmployees for the last name 'null' returned the next result: []	Test #5
==========company.removeEmployee()========== Employee [#2927396466, Anonymous Incognito [salary=9.99999999E8]] was not removed from the company! Employee [#9808208635, null null [salary=0.0]] was not removed from the company! Employee [#6854593986, Michael Smith [salary=500.0]] was successfully removed from the company! Employee [#6854593986, Michael Smith [salary=500.0]] was not removed from the company! Employee [null] was not removed from the company!	Test #6