**Міністерство освіти і науки, молоді та спорту України**

**Національний технічний університет України**

**«Київський політехнічний інститут»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №7**

**Тема: «Робота з колекціями в мові програмування Java.»**

**Виконав: Студент групи ІО-33**

**Заколенко Р.К.**

**Перевірив: Викладач**

**Невдащенко М.В.**

**Київ 2014**

1. **Завдання**

C2 = 0 ( 3308 mod 2 = 0) Інтерфейс List.

C3 = 2 ( 3308 mod 3 = 2) Двозв’язний список.

Завдання: Створити клас, що описує типізовану колекцію (типом колекції є узагальнений клас з лабораторної роботи №6), що реалізує заданий варіантом інтерфейс (List) та має задану внутрішню структуру (Двозв’язний список). Реалізувати всі методи інтерфейсу, а також створити не менше ніж 3 конструктори (1 – порожній, 2 – в який передається 1 об’єкт узагальненого класу, 3 – в який передається стандартна колекція об’єктів). Всі початкові дані задаються у виконавчому методі. Код повинен відповідати стандартам JCC та бути детально задокументований.

1. **Опис програми**

Програма містить чотири класи:

1. Клас Munitions (узагальнений клас з лабораторної №6);
2. Клас Сell (клас, за допомогою екземплярів якого буде створена структура колекції - двозв’язний список);
3. Клас DoubleList (implements List<Munitions>);
4. Клас Lab7 (із виконавчим методом);

У виконавчому методі ми створюємо об’єкти класу Munitions, які будуть елементами нашої колекції. За допомогою конструктора, у який передається об’єкт класу Munitions, класу DoubleList створюємо об’єкт класу DoubleList - sample, першим елементом якого буде переданий екземпляр класу Munitions. За допомогою методів реалізованих у нашій колекції додаємо (add()) у створений список елементи і т.д. System.out.println(sample) виведе на екран інформацію про наш список : кількість елементів та самі елементи.

1. **Код програми**

**package** ua.kpi.io33.zakolenko.lab7;

/\*\*

\* This class describe data about object Cell.

\*

\* **@author** Roman Zakolenko.

\* **@version** 1,6 20 May 2014.

\*/

**public** **class** Cell {

/\*\*

\* The field "data" contains data about object of class Munitions.

\*/

**private** Munitions data;

/\*\*

\* The field "previous" contains link on the previous cell.

\*/

**private** Cell previous;

/\*\*

\* The field "next" contains link on the next cell.

\*/

**private** Cell next;

/\*\*

\* Empty constructor.

\*/

**public** Cell() {

}

/\*\*

\* **@param** obj

\* contains data about object of class Munitions.

\*/

**public** Cell(Munitions obj) {

setData(obj);

}

/\*\*

\* **@param** prev

\* assigned with the field "previous".

\* **@param** next

\* assigned with the field "next".

\* **@param** obj

\* assigned with the field "data".

\*/

**public** Cell(Cell prev, Cell next, Munitions obj) {

setData(obj);

setNext(next);

setPrevious(prev);

}

/\*\*

\* **@return** link on the object of class Munitions.

\*/

**public** Munitions getData() {

**return** data;

}

/\*\*

\* **@param** data

\* assigned with the field "data".

\*/

**public** **void** setData(Munitions data) {

**this**.data = data;

}

/\*\*

\* **@return** link on the previous cell.

\*/

**public** Cell getPrevious() {

**return** previous;

}

/\*\*

\* Sets the value of field "previous" null.

\*/

**public** **void** setPrevious() {

**this**.previous = **null**;

}

/\*\*

\* **@param** c

\* assigned with field "previous".

\*/

**public** **void** setPrevious(Cell c) {

**this**.previous = c;

}

/\*\*

\* **@param** data

\* contains data about object of class Munitions.

\*/

**public** **void** setPrevious(Munitions data) {

**this**.previous = **new** Cell(data);

**this**.previous.next = **this**;

}

/\*\*

\* **@return** link on the next cell.

\*/

**public** Cell getNext() {

**return** next;

}

/\*\*

\* Sets the value of field "next" null.

\*/

**public** **void** setNext() {

**this**.next = **null**;

}

/\*\*

\* **@param** c

\* assigned with the field "next".

\*/

**public** **void** setNext(Cell c) {

**this**.next = c;

}

/\*\*

\* **@param** data

\* contains data about object of class Munitions.

\*/

**public** **void** setNext(Munitions data) {

**this**.next = **new** Cell(data);

**this**.next.previous = **this**;

}

}

package ua.kpi.io33.zakolenko.lab7;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Iterator;

import java.util.List;

import java.util.ListIterator;

/\*\*

\* Class DoubleList implements interface List. This class describe data about

\* object DoubleList.

\*

\* @author Roman Zakolenko.

\* @version 1,6 20 May 2014.

\*/

public class DoubleList implements List<Munitions> {

/\*\*

\* The field "first" contains link on the top of the list.

\*/

private Cell first;

/\*\*

\* Empty constructor.

\*/

public DoubleList() {

}

/\*\*

\* @param obj

\* contains data about object be placed in the top of the list.

\*/

public DoubleList(Munitions obj) {

setFirst(new Cell(obj));

}

/\*\*

\* @param collection

\* contains objects which will be placed on the list.

\*/

public DoubleList(Collection<? extends Munitions> collection) {

addAll(collection);

}

/\*\*

\* @return link of first element of list.

\*/

public Cell getFirst() {

return first;

}

/\*\*

\* @param cell

\* assigned with the fields first.

\*/

public void setFirst(Cell cell) {

this.first = cell;

}

/\*\*

\* @return string which contains information about object of class

\* DoubleList.

\*/

@Override

public String toString() {

StringBuffer s = new StringBuffer();

if (isEmpty()) {

s.append("The list is empty."+"\n");

} else {

s.append("The list contains "+size()+" elements:"+"\n");

ListIterator<Munitions> iter = listIterator();

while (iter.hasNext()) {

s.append("Munition " + iter.nextIndex() + " - "

+ iter.next().toString() + "\n");

}

}

return s.toString();

}

/\*\*

\* Appends the object of class Munitions to the end of this list.

\*/

@Override

public boolean add(Munitions e) {

Cell a = first;

if (a == null) {

setFirst(new Cell(null, null, e));

} else {

while (a.getNext() != null) {

a = a.getNext();

}

a.setNext(new Cell(e));

a.getNext().setPrevious(a);

}

return true;

}

/\*\*

\* Inserts the object of class Munitions at the specified position in this

\* list.

\*/

@Override

public void add(int index, Munitions element) {

if (index < this.size()) {

if (index == 0) {

setFirst(new Cell(null, first, element));

getFirst().getNext().setPrevious(getFirst());

} else {

ListIterator<Munitions> iter = this.listIterator();

while (iter.hasNext()) {

iter.next();

if (iter.nextIndex() == index) {

iter.add(element);

}

}

}

}

}

/\*\*

\* Appends all of the elements in the specified collection to the end of

\* this list.

\*/

@Override

public boolean addAll(Collection<? extends Munitions> c) {

Iterator<? extends Munitions> iter = c.iterator();

while (iter.hasNext()) {

this.add(iter.next());

}

return true;

}

/\*\*

\* Inserts all of the elements in the specified collection into this list at

\* the specified position.

\*/

@Override

public boolean addAll(int index, Collection<? extends Munitions> c) {

Iterator<? extends Munitions> iter = c.iterator();

while (iter.hasNext()) {

this.add(index, iter.next());

index++;

}

return true;

}

/\*\*

\* Removes all of the elements from this list.

\*/

@Override

public void clear() {

setFirst(null);

}

/\*\*

\* @return true if this list contains the specified element.

\*/

@Override

public boolean contains(Object o) {

return this.lastIndexOf(o) > -1;

}

/\*\*

\* @return true if this list contains all of the elements of the specified

\* collection.

\*/

@Override

public boolean containsAll(Collection<?> c) {

Iterator<?> iter = c.iterator();

boolean b = true;

while (iter.hasNext()) {

if (!this.contains(iter.next())) {

b = false;

}

}

return b;

}

/\*\*

\* @return the element at the specified position in this list.

\*/

@Override

public Munitions get(int index) {

ListIterator<Munitions> iter = this.listIterator();

Munitions obj = null;

while (iter.hasNext() && obj == null) {

if (iter.nextIndex() == index)

obj = iter.next();

else

iter.next();

}

return obj;

}

/\*\*

\* @return the index of the first occurrence of the specified element in

\* this list, or -1 if this list does not contain the element.

\*/

@Override

public int indexOf(Object o) {

int pos = -1;

Cell a = this.first;

for (int i = 0; i < this.size() && pos == -1; i++) {

if (a.getData().equals(o)) {

pos = i;

}

a = a.getNext();

}

return pos;

}

/\*\*

\* @return true if this list contains no elements.

\*/

@Override

public boolean isEmpty() {

return this.first == null;

}

/\*\*

\* @return an iterator over the elements in this list in proper sequence.

\*/

@Override

public Iterator<Munitions> iterator() {

return new Iterator<Munitions>() {

public Cell present = new Cell(null, first, null);

/\*\*

\* Removes from the underlying collection the last element returned

\* by this iterator.

\*/

@Override

public void remove() {

if (present == first) {

if (present.getNext() != null) {

setFirst(new Cell(null, present.getNext().getNext(),

present.getNext().getData()));

getFirst().getNext().setPrevious(getFirst());

present = getFirst();

}

} else {

if (present.getNext() != null) {

present.getPrevious().setNext(present.getNext());

present.getNext().setPrevious(present.getPrevious());

present = present.getPrevious();

} else {

present.getPrevious().setNext();

present = present.getPrevious();

}

}

}

/\*\*

\* @return the next element in the iteration.

\*/

@Override

public Munitions next() {

if (hasNext()) {

present = present.getNext();

return present.getData();

} else

return null;

}

/\*\*

\* @return true if the iteration has more elements.

\*/

@Override

public boolean hasNext() {

return present.getNext() != null;

}

};

}

/\*\*

\* @return the index of the last occurrence of the specified element in this

\* list, or -1 if this list does not contain the element.

\*/

@Override

public int lastIndexOf(Object o) {

int pos = -1;

Cell a = this.first;

for (int i = 0; i < this.size(); i++) {

if (a.getData().equals(o)) {

pos = i;

}

a = a.getNext();

}

return pos;

}

/\*\*

\* @return a list iterator over the elements in this list.

\*/

@Override

public ListIterator<Munitions> listIterator() {

return new ListIterator<Munitions>() {

/\*\*

\* The field "present" contains link on present cell of list.

\*/

public Cell present = new Cell(null, first, null);

/\*\*

\* The field "index" contains position of cursor in list.

\*/

public int index = -1;

/\*\*

\* Inserts the specified element into the list.

\*/

@Override

public void add(Munitions arg0) {

Cell a = new Cell(present, present.getNext(), arg0);

if (a.getNext() != null)

a.getNext().setPrevious(a);

present.setNext(a);

present = present.getNext();

}

/\*\*

\* @return true if this list iterator has more elements when

\* traversing the list in the forward direction.

\*/

@Override

public boolean hasNext() {

return present.getNext() != null;

}

/\*\*

\* @return true if this list iterator has more elements when

\* traversing the list in the reverse direction.

\*/

@Override

public boolean hasPrevious() {

return present.getPrevious() != null;

}

/\*\*

\* @return the next element in the list and advances the cursor

\* position.

\*/

@Override

public Munitions next() {

if (hasNext()) {

present = present.getNext();

index++;

return present.getData();

} else

return null;

}

/\*\*

\* @return the index of the element that would be returned by a

\* subsequent call to next().

\*/

@Override

public int nextIndex() {

return index + 1;

}

/\*\*

\* @return the previous element in the list and moves the cursor

\* position backwards.

\*/

@Override

public Munitions previous() {

if (hasPrevious()) {

present = present.getPrevious();

index--;

return present.getData();

} else

return null;

}

/\*\*

\* @return the index of the element that would be returned by a

\* subsequent call to previous().

\*/

@Override

public int previousIndex() {

if (hasPrevious()) {

return index - 1;

} else

return -1;

}

/\*\*

\* Removes from the list the last element that was returned by

\* next() or previous().

\*/

@Override

public void remove() {

if (present == first) {

if (present.getNext() != null) {

setFirst(new Cell(null, present.getNext().getNext(),

present.getNext().getData()));

getFirst().getNext().setPrevious(getFirst());

present = getFirst();

}

} else {

if (present.getNext() != null) {

present.getPrevious().setNext(present.getNext());

present.getNext().setPrevious(present.getPrevious());

present = present.getPrevious();

} else {

present.getPrevious().setNext();

present = present.getPrevious();

}

}

}

/\*\*

\* Replaces the last element returned by next() or previous() with

\* the specified element.

\*/

@Override

public void set(Munitions data) {

present.setData(data);

}

};

}

/\*\*

\* @return a list iterator of the elements in this list, starting at the

\* specified position in this list.

\*/

@Override

public ListIterator<Munitions> listIterator(int index) {

ListIterator<Munitions> iter = this.listIterator();

while (iter.hasNext() && iter.nextIndex() <= index) {

iter.next();

}

return iter;

}

/\*\*

\* Removes the element at the specified position in this list.

\*/

@Override

public boolean remove(Object o) {

ListIterator<Munitions> iter = this.listIterator();

boolean b = false;

while (iter.hasNext() && !b) {

if (iter.next().equals(o)) {

iter.remove();

b = true;

}

}

return b;

}

/\*\*

\* Removes the first occurrence of the specified element from this list, if

\* it is present.

\*/

@Override

public Munitions remove(int index) {

ListIterator<Munitions> iter = this.listIterator();

Munitions b = null;

while (iter.hasNext() && b == null) {

if (iter.nextIndex() == index) {

b = iter.next();

iter.remove();

} else

iter.next();

}

return b;

}

/\*\*

\* Removes from this list all of its elements that are contained in the

\* specified collection.

\*/

@Override

public boolean removeAll(Collection<?> c) {

boolean b = true;

Iterator<?> iter = c.iterator();

while (iter.hasNext()) {

b = this.remove(iter.next());

}

return b;

}

/\*\*

\* Retains only the elements in this list that are contained in the

\* specified collection.

\*/

@Override

public boolean retainAll(Collection<?> c) {

boolean b = false;

ListIterator<Munitions> iter = this.listIterator();

while (iter.hasNext()) {

if (!c.contains(iter.next())) {

iter.remove();

b = true;

}

}

return b;

}

/\*\*

\* Replaces the element at the specified position in this list with the

\* specified element.

\*/

@Override

public Munitions set(int index, Munitions element) {

ListIterator<Munitions> iter = this.listIterator();

Munitions result = null;

while (iter.hasNext() && result == null) {

if (iter.nextIndex() == index) {

result = iter.next();

iter.set(element);

} else

iter.next();

}

return result;

}

/\*\*

\* @return the number of elements in this list.

\*/

@Override

public int size() {

ListIterator<Munitions> iter = this.listIterator();

while (iter.hasNext()) {

iter.next();

}

return iter.nextIndex();

}

/\*\*

\* @return the sublist from a fromIndex on toIndex.

\*/

@Override

public List<Munitions> subList(int fromIndex, int toIndex) {

List<Munitions> result = new ArrayList<Munitions>();

if (fromIndex == toIndex) {

return result;

} else {

ListIterator<Munitions> iter = this.listIterator();

while (iter.hasNext()) {

if (iter.nextIndex() >= fromIndex && iter.nextIndex() < toIndex) {

result.add(iter.next());

} else

iter.next();

}

return result;

}

}

/\*\*

\* @return an array containing all of the elements in this list in proper

\* sequence.

\*/

@Override

public Object[] toArray() {

Object obj[] = new Object[this.size()];

Cell a = this.first;

for (int i = 0; i < obj.length; i++) {

obj[i] = a.getData();

a = a.getNext();

}

return obj;

}

/\*\*

\* @return an array containing all of the elements in this list in proper

\* sequence.

\*/

@Override

public <T> T[] toArray(T[] a) {

if (a instanceof Munitions[]) {

if (a.length >= size()) {

System.arraycopy(toArray(), 0, a, 0, size());

} else {

a = (T[]) new Munitions[size()];

System.arraycopy(toArray(), 0, a, 0, size());

}

}

return a;

}

}

**package** ua.kpi.io33.zakolenko.lab7;

**public** **class** Lab7 {

**public** **static** **void** main(String args[]) {

Munitions m1 = **new** Munitions(15, 20, "Алюминий");

Munitions m3 = **new** Munitions(10, 30, "Платина");

Munitions m4 = **new** Munitions(11, 45, "Медь");

Munitions m2 = **new** Munitions(15, 22, "Алюминий");

Munitions m5 = **new** Munitions(12, 67, "Железо");

DoubleList list = **new** DoubleList(m1);

list.add(m5);

list.add(0, m4);

DoubleList list1 = **new** DoubleList();

list1.add(m3);

list1.add(m2);

list.addAll(list1);

list.set(1, m5);

System.*out*.println(list);

list.clear();

System.*out*.println(list);

}

}

package ua.kpi.io33.zakolenko.lab7;

import java.util.Arrays;

import java.util.Comparator;

/\*\*

\* This class describe data about object Munitions.

\*

\* @author Roman Zakolenko.

\* @version 1,6 20 May 2014.

\*/

public class Munitions {

/\*\*

\* The field "weight" contains value of weight munition.

\*/

private int weight;

/\*\*

\* The field "price" contains value of price munition.

\*/

private int price;

/\*\*

\* THe field "material" contains value of material munition.

\*/

private String material;

/\*\*

\* @param w

\* assigned with field weight.

\* @param p

\* assigned with field price.

\* @param m

\* assigned with field material.

\*/

public Munitions(int w, int p, String m) {

setWeight(w);

setPrice(p);

setMaterial(m);

}

/\*\*

\* @param obj

\* contains the field whose values ​​will be installed at the new

\* object of class Munitions.

\*/

public Munitions(Munitions obj) {

setWeight(obj.getWeight());

setPrice(obj.getPrice());

setMaterial(obj.getMaterial());

}

/\*\*

\* @return value of field weight.

\*/

public int getWeight() {

return weight;

}

/\*\*

\* @param weight

\* assigned with field weight.

\*/

public void setWeight(int weight) {

this.weight = weight;

}

/\*\*

\* @return value of field price.

\*/

public int getPrice() {

return price;

}

/\*\*

\* @param price

\* assigned with field price.

\*/

public void setPrice(int price) {

this.price = price;

}

/\*\*

\* @return value of fields material.

\*/

public String getMaterial() {

return new String(material);

}

/\*\*

\* @param material

\* assigned with field material.

\*/

public void setMaterial(String material) {

this.material = new String(material);

}

/\*\*

\* @return true if compared objects of class Munitions are equal else return

\* false.

\*/

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof Munitions))

return false;

Munitions obj1 = (Munitions) obj;

if (this.weight == obj1.weight && this.price == obj1.price

&& this.material.equals(obj1.material)) {

return true;

} else {

return false;

}

}

/\*

\* (non-Javadoc)

\*

\* @see java.lang.Object#hashCode()

\*/

@Override

public int hashCode() {

int hash = 11;

hash = hash \* this.weight + this.price;

hash = hash \* this.price + this.material.hashCode();

return hash;

}

/\*\*

\* @return string which contains information about object of class

\* Munitions.

\*/

@Override

public String toString() {

return new String("Weight: " + getWeight() + "; Price: " + getPrice()

+ "; Material: " + getMaterial() + ".");

}

/\*\*

\* @param m

\* contains elements of class Munitions.

\* @return the sum of the fields price of objects of class Munitions that

\* contains in array m.

\*/

public static int cost(Munitions[] m) {

int price = 0;

for (int i = 0; i < m.length; i++) {

price = price + m[i].getPrice();

}

return price;

}

/\*\*

\* @param m

\* contains elements of class Munitions.

\*/

public static void sortedByWeight(Munitions[] m) {

Arrays.sort(m, new Comparator<Munitions>() {

@Override

public int compare(Munitions m1, Munitions m2) {

return m1.getWeight() - m2.getWeight();

}

});

}

/\*\*

\* @param m

\* contains elements of class Munitions.

\* @param a

\* is initial value range.

\* @param b

\* is end of the range.

\* @return array of objects of class Munitions that go to the specified

\* range.

\*/

public static Munitions[] priceRange(Munitions[] m, int a, int b) {

int length = 0;

for (int i = 0; i < m.length; i++) {

if (m[i].getPrice() >= a && m[i].getPrice() <= b) {

length++;

}

}

Munitions[] m1 = new Munitions[length];

for (int i = 0; i < m.length; i++) {

if (m[i].getPrice() >= a && m[i].getPrice() <= b) {

m1[i] = m[i];

}

}

return m1;

}

}

1. **Висновок**

В результаті виконання лабораторної роботи я ознайомився із колекціями в мові програмування Java та створив власну, яка реалізує інтерфейс List. Для того щоб реалізувати більшість методів даного інтерфейсу спочатку був перевизначений метод listIterator(), за допомогою якого здійснюється прохід по колекції. Для коректного порівняння об’єктів класу Munitions був перевизначений метод equals() і як наслідок hashCode(). Також у класах Munitions та DoubleList був перевизначений метод toString(), який повертає строку, яка містить інформацію про об’єкти даних класів у коректному для користувача вигляді.