Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №4

з дисципліни «Інженерія програмного забезпечення»

на тему **«Шаблони поведінки. Шаблони Iterator, Mediator та Observer»**

ВИКОНАВ:

Студент І курсу ФІОТ

групи ІО-81

Дудка Максим

Варіант №6

ПЕРЕВІРИВ:

Доцент каф. ОТ

к.т.н., с.н.с.

Антонюк А.І.

Київ 2019 р.

**Мета**

Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Iterator, Mediator та Observer.

**Завдання**

1. Вивчити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.

2. Детально вивчити шаблони поведінки для проектування ПЗ - Iterator, Mediator та Observer. Для кожного з них:

* вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
* знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
* вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
* вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.

3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork5. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи. Приклад реалізації бізнес-методу:

void draw(int x, int y){

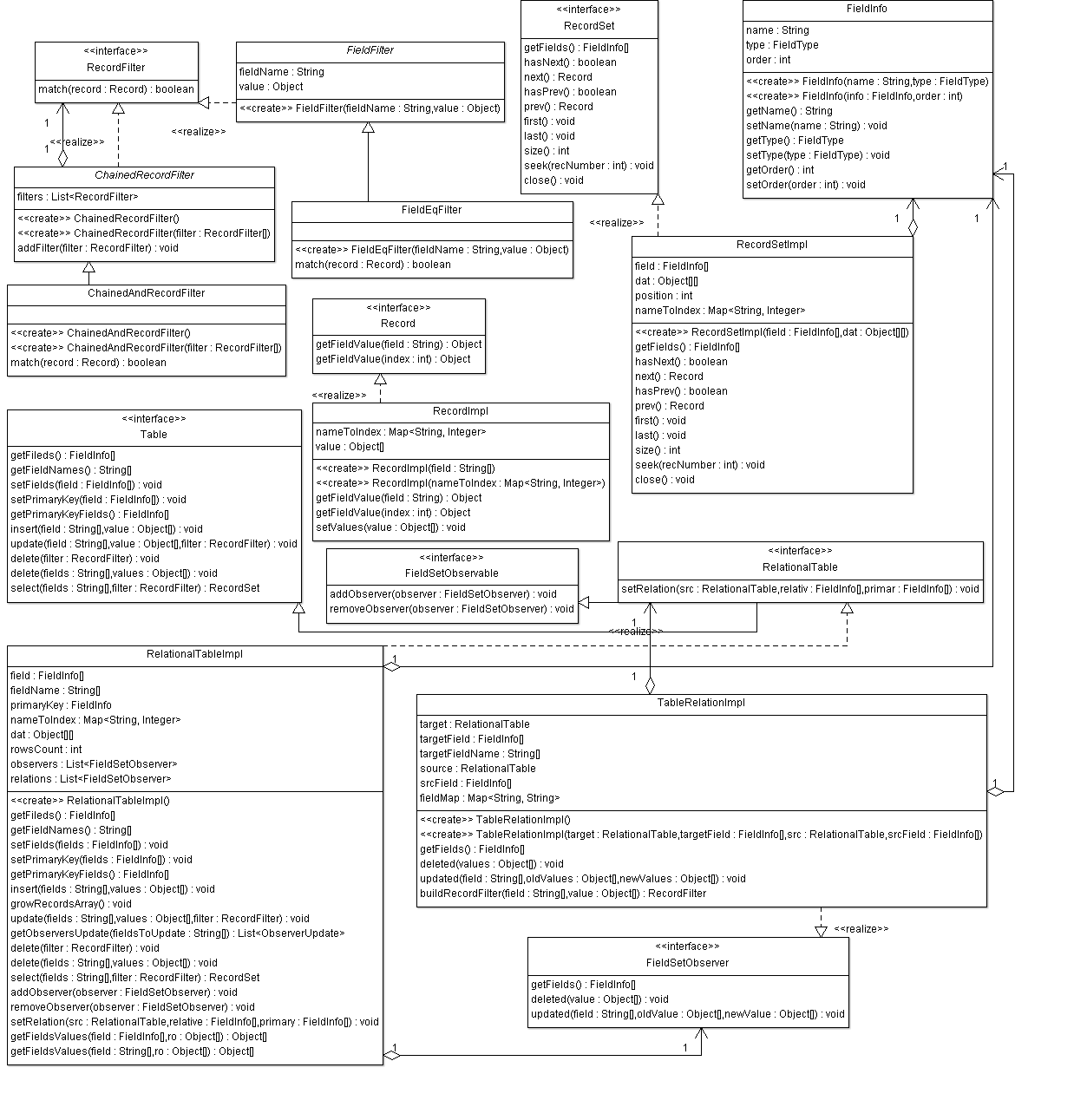
System.out.println(“Метод draw з параметрами x=”+x+” y=”+y);

}

4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

**Варіант**

6. Визначити специфікації класів для подання реляційної таблиці та обмеження зовнішнього ключа з можливістю його перевірки під час зміни значень полів. Забезпечити слабку зв'язаність елементів.



**Код**

package com.lab111.labwork5;  
  
*/\*\*  
 \* Chained filter with 'AND' type condition: returns true only if all filters  
 \* in the chain also returns true.  
 \*  
 \** ***@author*** *Maksym Dudka  
 \*/*public class ChainedAndRecordFilter extends ChainedRecordFilter {  
  
 public ChainedAndRecordFilter() {  
 super();  
 }  
  
 public ChainedAndRecordFilter(RecordFilter filters[]) {  
 super(filters);  
 }  
  
 @Override  
 public boolean match(Record record) {  
 for (RecordFilter filter : filters) {  
 if (!filter.match(record))  
 return false;  
 }  
 return true;  
 }  
  
}

package com.lab111.labwork5;  
  
  
import java.util.LinkedList;  
  
*/\*\*  
 \* Abstract base class for chained filters.  
 \*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public abstract class ChainedRecordFilter implements RecordFilter {  
  
 protected LinkedList<RecordFilter> filters;  
  
 public ChainedRecordFilter() {  
 filters = new LinkedList<RecordFilter>();  
 }  
  
 public ChainedRecordFilter(RecordFilter filters[]) {  
 this();  
 for (int i=0; i<filters.length; i++) {  
 this.filters.add(filters[i]);  
 }  
 }  
  
 public void addFilter(RecordFilter filter) {  
 filters.add(filter);  
 }  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public class FieldEqFilter extends FieldFilter {  
  
 public FieldEqFilter(String fieldName, Object value) {  
 super(fieldName, value);  
 }  
  
 @Override  
 public boolean match(Record record) {  
 return record.getFieldValue(fieldName).equals(value);  
 }  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public abstract class FieldFilter implements RecordFilter {  
  
 protected String fieldName;  
  
 protected Object value;  
  
 public FieldFilter(String fieldName, Object value) {  
 this.fieldName = fieldName;  
 this.value = value;  
 }  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public class FieldInfo {  
  
 public enum FieldType {  
 *TYPE\_INTEGER*,  
 *TYPE\_LONG*,  
 *TYPE\_STRING*,  
 *TYPE\_BOOLEAN* }  
  
 private String name;  
  
 private FieldType type;  
  
 int order;  
  
 public FieldInfo(String name, FieldType type) {  
 this.name = name;  
 this.type = type;  
 }  
  
 public FieldInfo(FieldInfo info, int order) {  
 this.name = info.getName();  
 this.type = info.getType();  
 this.order = order;  
 }  
  
 */\*\*  
 \** ***@return*** *field name  
 \*/* public String getName() {  
 return name;  
 }  
  
 */\*\*  
 \** ***@param*** *name new field name  
 \*/* public void setName(String name) {  
 this.name = name;  
 }  
  
 */\*\*  
 \** ***@return*** *field type  
 \*/* public FieldType getType() {  
 return type;  
 }  
  
 */\*\*  
 \** ***@param*** *type new field type  
 \*/* public void setType(FieldType type) {  
 this.type = type;  
 }  
  
 */\*\*  
 \** ***@return*** *field order in the table or record set row  
 \*/* public int getOrder() {  
 return order;  
 }  
  
 */\*\*  
 \** ***@param*** *order field order to set  
 \*/* public void setOrder(int order) {  
 this.order = order;  
 }  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \* Interface that allows a table to notify an observer object about records  
 \* changes.  
 \*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface FieldSetObservable {  
  
 */\*\*  
 \* Adds an observer  
 \** ***@param*** *observer  
 \*/* public void addObserver(FieldSetObserver observer);  
  
 */\*\*  
 \* Removes an observer  
 \** ***@param*** *observer  
 \*/* public void removeObserver(FieldSetObserver observer);  
  
}

package com.lab111.labwork5;  
  
import java.util.Objects;  
  
public class Lab5 {  
  
 static RelationalTableImpl *relationalTable* = new RelationalTableImpl();  
 static Object[] *object*;  
 static FieldSetObserver *fieldSetObserver* = new FieldSetObserver() {  
 @Override  
 public FieldInfo[] getFields() {  
 System.*out*.println("FieldInfo[0] = something...");  
 return new FieldInfo[0];  
 }  
  
 @Override  
 public void deleted(Object[] values) {  
 System.*out*.println("FieldInfo[0] was deleted");  
 }  
  
 @Override  
 public void updated(String[] fields, Object[] oldValues, Object[] newValues) {  
 System.*out*.println("Updated fields info...");  
 }  
 };  
 public static void main(String[] args)  
 {  
 *relationalTable*.addObserver(*fieldSetObserver*);  
 *fieldSetObserver*.getFields();  
 *fieldSetObserver*.deleted(*object*);  
 *fieldSetObserver*.updated(new String[0], *object*, *object*);  
 }  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface Record {  
  
 Object getFieldValue(String field);  
  
 Object getFieldValue(int index);  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface RecordFilter {  
  
 public boolean match(Record record);  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*import java.util.HashMap;  
import java.util.Map;  
  
public class RecordImpl implements Record {  
  
 private Map<String, Integer> nameToIndex;  
 private Object values[];  
  
 public RecordImpl(String fields[]) {  
 this.nameToIndex = new HashMap<String, Integer>();  
 for (int i=0; i<fields.length; i++) {  
 nameToIndex.put(fields[i], new Integer(i));  
 }  
 }  
  
 public RecordImpl(Map<String, Integer> nameToIndex) {  
 this.nameToIndex = nameToIndex;  
 }  
  
 @Override  
 public Object getFieldValue(String field) {  
 Integer index = nameToIndex.get(field);  
 if (index == null)  
 return null;  
 return values[index];  
 }  
  
 @Override  
 public Object getFieldValue(int index) {  
 if (index < 0 || index >= values.length)  
 throw new IndexOutOfBoundsException();  
 return values[index];  
 }  
  
 */\*\*  
 \* Sets values for current row.  
 \* Note the 'default' (aka 'package') visibility: this is not a public method  
 \** ***@param*** *values  
 \*/* void setValues(Object values[]) {  
 this.values = values;  
 }  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \* Interface to record set returned by query to table(s).  
 \* Utilizes 'iterator' pattern for access to particular records.  
 \*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface RecordSet {  
  
 */\*\*  
 \* Returns information about fields set in contained records  
 \** ***@return*** *\*/* public FieldInfo [] getFields();  
  
 */\*\*  
 \** ***@return*** *true if there is next record to retrieve  
 \*/* public boolean hasNext();  
  
 */\*\*  
 \** ***@return*** *next record in record set  
 \*/* public Record next();  
  
 */\*\*  
 \** ***@return*** *true if there is previous record to retrieve  
 \*/* public boolean hasPrev();  
  
 */\*\*  
 \** ***@return*** *previous record in record set  
 \*/* public Record prev();  
  
 */\*\*  
 \* Places an iterator to a first record  
 \*/* public void first();  
  
 */\*\*  
 \* Places an iterator to a last record  
 \*/* public void last();  
  
 */\*\*  
 \** ***@return*** *number of records in record set  
 \*/* public int size();  
  
 */\*\*  
 \* Places an iterator to a record at given index  
 \** ***@param*** *recNumber index of record to point to  
 \** ***@throws*** *IndexOutOfBoundsException when index is negative or greater than number of records  
 \*/* public void seek(int recNumber) throws IndexOutOfBoundsException;  
  
 */\*\*  
 \* Closes the record set reclaiming all associated resources  
 \*/* public void close();  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*import java.util.HashMap;  
import java.util.Map;  
  
public class RecordSetImpl implements RecordSet {  
 private FieldInfo fields[];  
 private Object data[][];  
 private int position;  
 private Map<String, Integer> nameToIndex;  
  
 public RecordSetImpl(FieldInfo fields[], Object data[][]) {  
 this.fields = fields;  
 this.data = data;  
 position = 0;  
 nameToIndex = new HashMap<String, Integer>();  
 for (int i=0; i<fields.length; i++) {  
 nameToIndex.put(fields[i].getName(), new Integer(i));  
 }  
 }  
  
 @Override  
 public FieldInfo[] getFields() {  
 return fields;  
 }  
  
 @Override  
 public boolean hasNext() {  
 return position >=0 && position < data.length;  
 }  
  
 @Override  
 public Record next() {  
 if (position < 0 || position >= data.length)  
 return null;  
 RecordImpl rec = new RecordImpl(nameToIndex);  
 rec.setValues(data[position]);  
 position++;  
 return rec;  
 }  
  
 @Override  
 public boolean hasPrev() {  
 return position > 0 && position <= data.length;  
 }  
  
 @Override  
 public Record prev() {  
 if (position <= 0 || position > data.length)  
 return null;  
 RecordImpl rec = new RecordImpl(nameToIndex);  
 position--;  
 rec.setValues(data[position]);  
 return rec;  
 }  
  
 @Override  
 public void first() {  
 position = 0;  
 }  
  
 @Override  
 public void last() {  
 position = data.length;  
 }  
  
 @Override  
 public int size() {  
 return data.length;  
 }  
  
 @Override  
 public void seek(int recNumber) throws IndexOutOfBoundsException {  
 if (recNumber < 0 || recNumber > data.length)  
 throw new IndexOutOfBoundsException();  
 position = recNumber;  
 }  
  
 @Override  
 public void close() {  
 data = null;  
 nameToIndex = null;  
 fields = null;  
 }  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface RelationalTable extends Table, FieldSetObservable {  
  
 public void setRelation(RelationalTable src, FieldInfo relative[], FieldInfo primary[]);  
  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public class RelationalTableFactory {  
  
 public static RelationalTable createRelationalTable(FieldInfo fields[]) {  
 RelationalTable table = new RelationalTableImpl();  
 table.setFields(fields);  
 return table;  
 }  
  
}

package com.lab111.labwork5;  
  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Map;  
  
class RelationalTableImpl implements RelationalTable {  
  
 protected FieldInfo fields[];  
 protected String fieldNames[];  
 protected FieldInfo primaryKey;  
 private Map<String, Integer> nameToIndex;  
 private Object data[][];  
 private int rowsCount;  
  
 protected List<FieldSetObserver> observers;  
 // protected Map<String, List<FieldSetObserver>> fieldObservers;  
  
 protected List<FieldSetObserver> relations;  
  
 /\*  
 \* Internal class for simplifying observer notifications on update  
 \*/  
 private class ObserverUpdate {  
 public FieldSetObserver observer;  
 public String fields[];  
 public Object values[];  
 }  
  
 public RelationalTableImpl() {  
 nameToIndex = new HashMap<String, Integer>();  
 // fieldObservers = new HashMap<String, List<FieldSetObserver>>();  
 observers = new LinkedList<FieldSetObserver>();  
 relations = new LinkedList<FieldSetObserver>();  
 rowsCount = 0;  
 }  
  
 @Override  
 public FieldInfo[] getFileds() {  
 return fields;  
 }  
  
 @Override  
 public String[] getFieldNames() {  
 return fieldNames;  
 }  
  
 @Override  
 public void setFields(FieldInfo[] fields) {  
 if (this.fields != null)  
 throw new UnsupportedOperationException();  
 this.fields = fields;  
 fieldNames = new String[fields.length];  
 for (int i = 0; i < fields.length; i++) {  
 fieldNames[i] = fields[i].getName();  
 nameToIndex.put(fieldNames[i], new Integer(i));  
 }  
 data = new Object[64][fields.length];  
 }  
  
 @Override  
 public void setPrimaryKey(FieldInfo[] fields) {  
 // *TODO Auto-generated method stub* }  
  
 @Override  
 public FieldInfo[] getPrimaryKeyFields() {  
 // *TODO Auto-generated method stub* return null;  
 }  
  
 @Override  
 public void insert(String[] fields, Object[] values) {  
 if (data.length == rowsCount)  
 growRecordsArray();  
 for (int i = 0; i < fields.length; i++) {  
 data[rowsCount][nameToIndex.get(fields[i])] = values[i];  
 }  
 rowsCount++;  
 }  
  
 private void growRecordsArray() {  
 int length = data.length \* 2;  
 Object newData[][] = new Object[length][fields.length];  
 for (int i = 0; i < rowsCount; i++) {  
 for (int j = 0; j < fields.length; j++) {  
 newData[i][j] = data[i][j];  
 data[i][j] = null;  
 }  
 }  
 data = newData;  
 }  
  
 @Override  
 public void update(String[] fields, Object[] values, RecordFilter filter) {  
 if (rowsCount == 0)  
 return;  
 List<ObserverUpdate> ous = getObserversUpdate(fields);  
 RecordImpl rec = new RecordImpl(nameToIndex);  
 for (int i = 0; i < rowsCount; i++) {  
 rec.setValues(data[i]);  
 if (filter.match(rec)) { // record will be updated  
 for (ObserverUpdate ou : ous) { // save old values for observers  
 ou.values = getFieldsValues(ou.fields, data[i]);  
 }  
 for (int j = 0; j < fields.length; j++) { // update record  
 data[i][nameToIndex.get(fields[j])] = values[j];  
 }  
 for (ObserverUpdate ou : ous) { // notify observers  
 ou.observer.updated(ou.fields, ou.values, getFieldsValues(ou.fields, data[i]));  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Builds list of ObserverUpdate objects that will be notified during  
 \* given fields update.  
 \*  
 \** ***@param*** *fieldsToUpdate  
 \** ***@return*** *\*/* private List<ObserverUpdate> getObserversUpdate(String[] fieldsToUpdate) {  
 List<ObserverUpdate> result = new LinkedList<RelationalTableImpl.ObserverUpdate>();  
 List<String> fieldsList = Arrays.*asList*(fieldsToUpdate);  
 List<String> observedFields = new LinkedList<String>();  
 for (FieldSetObserver observer : observers) {  
 FieldInfo oFields[] = observer.getFields();  
 for (int i = 0; i < oFields.length; i++) {  
 if (fieldsList.contains(oFields[i].getName())) {  
 observedFields.add(oFields[i].getName());  
 }  
 }  
 if (!observedFields.isEmpty()) {  
 ObserverUpdate ou = new ObserverUpdate();  
 ou.observer = observer;  
 ou.fields = observedFields.toArray(new String[]{});  
 ou.values = null;  
 result.add(ou);  
 }  
 }  
 return result;  
 }  
  
 @Override  
 public void delete(RecordFilter filter) {  
 RecordImpl record = new RecordImpl(nameToIndex);  
 for (int i = 0; i < rowsCount; ) {  
 record.setValues(data[i]);  
 if (filter.match(record)) {  
 for (FieldSetObserver observer : observers) {  
 Object values[] = getFieldsValues(observer.getFields(), data[i]);  
 observer.deleted(values);  
 }  
 for (int j = i; j < rowsCount - 1; j++) {  
 for (int k = 0; k < fields.length; k++) {  
 data[j][k] = data[j + 1][k];  
 }  
 }  
 rowsCount--;  
 for (int j = 0; j < fields.length; j++) {  
 data[rowsCount][j] = null;  
 }  
 } else {  
 i++;  
 }  
 }  
 }  
  
 @Override  
 public void delete(String[] fields, Object[] values) {  
 ChainedRecordFilter filter = new ChainedAndRecordFilter();  
 for (int i = 0; i < fields.length; i++) {  
 filter.addFilter(new FieldEqFilter(fields[i], values[i]));  
 }  
 delete(filter);  
 }  
  
 @Override  
 public RecordSet select(String[] fields, RecordFilter filter) {  
 List<Object[]> records = new LinkedList<Object[]>();  
 RecordImpl rec = new RecordImpl(nameToIndex);  
 for (int i = 0; i < rowsCount; i++) {  
 rec.setValues(data[i]);  
 if (filter == null || filter.match(rec)) {  
 records.add(getFieldsValues(fields, data[i]));  
 }  
 }  
 FieldInfo resultFields[] = new FieldInfo[fields.length];  
 for (int i = 0; i < resultFields.length; i++) {  
 resultFields[i] = new FieldInfo(this.fields[nameToIndex.get(fields[i])], i);  
 }  
 RecordSetImpl result = new RecordSetImpl(resultFields, records.toArray(new Object[][]{}));  
 return result;  
 }  
  
 @Override  
 public void addObserver(FieldSetObserver observer) {  
 observers.add(observer);  
 }  
  
 @Override  
 public void removeObserver(FieldSetObserver observer) {  
 observers.remove(observer);  
 }  
  
 @Override  
 public void setRelation(RelationalTable src, FieldInfo[] relative,  
 FieldInfo[] primary) {  
 FieldSetObserver observer = new TableRelationImpl(this, relative, src, primary);  
 relations.add(observer);  
 }  
  
 protected Object[] getFieldsValues(FieldInfo fields[], Object row[]) {  
 Object values[] = new Object[fields.length];  
 for (int i = 0; i < fields.length; i++) {  
 values[i] = row[nameToIndex.get(fields[i].getName())];  
 }  
 return values;  
 }  
  
 protected Object[] getFieldsValues(String fields[], Object row[]) {  
 Object values[] = new Object[fields.length];  
 int index;  
 for (int i = 0; i < fields.length; i++) {  
 index = nameToIndex.get(fields[i]);  
 values[i] = row[index];  
 }  
 return values;  
 }  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface RelationChecker {  
  
 public boolean isRecordValid(Object values[]);  
}

package com.lab111.labwork5;  
  
*/\*\*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public interface Table {  
  
 public FieldInfo [] getFileds();  
  
 public String [] getFieldNames();  
  
 public void setFields(FieldInfo fields[]);  
  
 public void setPrimaryKey(FieldInfo fields[]);  
  
 public FieldInfo [] getPrimaryKeyFields();  
  
 public void insert(String fields[], Object values[]);  
  
 public void update(String fields[], Object values[], RecordFilter filter);  
  
 public void delete(RecordFilter filter);  
  
 public void delete(String [] fields, Object [] values);  
  
 public RecordSet select(String [] fields, RecordFilter filter);  
  
}

package com.lab111.labwork5;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* Class that implements tables relation on some field sets.  
 \* It uses an 'observer' pattern to get notified of events on primary  
 \* table. Also, it serves as 'mediator' cause it implements business logic  
 \* of these events processing by propagating changes to related table.  
 \*  
 \** ***@author*** *Maxym Dudka  
 \** ***@version*** *1.0  
 \*/*public class TableRelationImpl implements FieldSetObserver {  
  
 private RelationalTable target;  
  
 @SuppressWarnings("unused")  
 private FieldInfo targetFields[];  
  
 private String targetFieldNames[];  
  
 private RelationalTable source;  
  
 private FieldInfo srcFields[];  
  
 private Map<String, String> fieldMap;  
  
 */\*\*  
 \* Default constructor, disabled by making it private  
 \*/* @SuppressWarnings("unused")  
 private TableRelationImpl() {}  
  
 */\*\*  
 \* Constructor that accepts all necessary relation data.  
 \* Note that type matching checks was skipped for simplicity reason.  
 \** ***@param*** *target related table  
 \** ***@param*** *targetFields fields in related table that takes part in relation  
 \** ***@param*** *src primary table  
 \** ***@param*** *srcFields fields in primary table that takes part in relation  
 \*/* public TableRelationImpl(RelationalTable target, FieldInfo targetFields[], RelationalTable src, FieldInfo srcFields[]) {  
 targetFieldNames = new String[targetFields.length];  
 fieldMap = new HashMap<String, String>();  
 for (int i=0; i<targetFields.length; i++) {  
 targetFieldNames[i] = targetFields[i].getName();  
 fieldMap.put(srcFields[i].getName(), targetFieldNames[i]);  
 }  
 this.targetFields = targetFields;  
 this.srcFields = srcFields;  
 this.target = target;  
 this.source = src;  
 source.addObserver(this);  
 }  
  
 @Override  
 public FieldInfo[] getFields() {  
 return srcFields;  
 }  
  
 @Override  
 public void deleted(Object[] values) {  
 // build filter for records to be deleted  
 RecordFilter filter = buildRecordFilter(targetFieldNames, values);  
 // Fire deletion  
 target.delete(filter);  
 }  
  
 @Override  
 public void updated(String fields[], Object[] oldValues, Object[] newValues) {  
 // build filter for records needs updating  
 String targetFields[] = new String[fields.length];  
 for (int i=0; i<fields.length; i++)  
 targetFields[i] = fieldMap.get(fields[i]);  
 RecordFilter filter = buildRecordFilter(targetFields, oldValues);  
 // update them  
 target.update(targetFieldNames, newValues, filter);  
 }  
  
 */\*\*  
 \* Builds record filter matching all records with given values of known  
 \* fields in related table.  
 \** ***@param*** *values array of field values  
 \** ***@return*** *\*/* private RecordFilter buildRecordFilter(String fields[], Object values[]) {  
 if (fields.length == 1)  
 return new FieldEqFilter(fields[0], values[0]);  
 ChainedRecordFilter filter = new ChainedAndRecordFilter();  
 for (int i=0; i<fields.length; i++) {  
 RecordFilter fieldFilter = new FieldEqFilter(fields[i], values[i]);  
 filter.addFilter(fieldFilter);  
 }  
 return filter;  
 }  
}

**Висновок**

Було ознайомлено із шаблонами поведінки Iterator, Mediator та Observer. Отримано навички із застосування шаблонів. Була розроблена відповідна тестова програма. Результати успішної роботи тестової програми наведені вище підтверджують правильність обраних рішень.