ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент департамента программной инженерии факультета компьютерных наук канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.Ю.Дегтярёв  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. | УТВЕРЖДАЮ Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.В.Шилов  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

**ПРОГРАММА ЛОКАЛЬНОГО ПОИСКА ДОКУМЕНТОВ ПО ИХ ИМЕНИ И СОДЕРЖИМОМУ**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU. 17701729. 505900-01 12 01-1-ЛУ**

Исполнитель

студент группы 143ПИ  
\_\_\_\_\_\_\_\_\_\_\_/А.А.Смилянский/  
«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

УТВЕРЖДЁН

RU. 17701729. 505900-01 12 01-1-ЛУ

**ПРОГРАММА ЛОКАЛЬНОГО ПОИСКА ДОКУМЕНТОВ ПО ИХ ИМЕНИ И СОДЕРЖИМОМУ**

**Текст программы**

**RU. 17701729. 505900-01 12 01-1**

**Листов 83**



**Содержание**

[**1. Текст программы 3**](#_Toc480409598)

[**1.1. Класс Occurrence 3**](#_Toc480409599)

[**1.2. Класс Path 4**](#_Toc480409600)

[**1.3. Класс Word 6**](#_Toc480409601)

[**1.4. Класс ComplexNode 8**](#_Toc480409602)

[**1.5. Класс H2Storage 9**](#_Toc480409603)

[**1.6. Класс Inclusion 14**](#_Toc480409604)

[**1.7. Класс InvertedIndex 15**](#_Toc480409605)

[**1.8. Интерфейс Node 17**](#_Toc480409606)

[**1.9. Класс SimpleNode 17**](#_Toc480409607)

[**1.10. Класс FileVisitorIndexer 19**](#_Toc480409608)

[**1.11. Класс FileVisitorIndexerDB 23**](#_Toc480409609)

[**1.12. Класс Index 26**](#_Toc480409610)

[**1.13. Класс IndexingHandler 29**](#_Toc480409611)

[**1.14. Класс IndexingRequest 31**](#_Toc480409612)

[**1.15. Класс IndexParameters 34**](#_Toc480409613)

[**1.16. Класс IndexStorage 37**](#_Toc480409614)

[**1.17. Класс IndexStorageWithLevels 38**](#_Toc480409615)

[**1.18. Перечисление Parameter 39**](#_Toc480409616)

[**1.19. Интерфейс Request 41**](#_Toc480409617)

[**1.20. Класс SearchRequest 41**](#_Toc480409618)

[**1.21. Класс Controllers 44**](#_Toc480409619)

[**1.22. Класс IndexCellController 47**](#_Toc480409620)

[**1.23. Класс IndexCreationController 49**](#_Toc480409621)

[**1.24. Класс IndexFoldersController 52**](#_Toc480409622)

[**1.25. Класс IndexInfoController 55**](#_Toc480409623)

[**1.26. Класс MainController 58**](#_Toc480409624)

[**1.27. Класс PathCellController 71**](#_Toc480409625)

[**1.28. Класс InclusionCell 75**](#_Toc480409626)

[**1.29. Класс IndexCell 77**](#_Toc480409627)

[**1.30. Класс PathCell 78**](#_Toc480409628)

[**1.31. Класс Main 80**](#_Toc480409629)

[**Приложение 1. UML – диаграмма классов 82**](#_Toc480409630)

[**Лист регистрации изменений 83**](#_Toc480409631)

# **Текст программы**

* 1. **Класс Occurrence**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 27.03.17 17:31  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages.entities;  
  
**import** java.io.Serializable;  
**import** javax.persistence.Entity;  
**import** javax.persistence.FetchType;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.Id;  
**import** javax.persistence.JoinColumn;  
**import** javax.persistence.ManyToOne;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 27.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/*@Entity  
**public class** Occurrence **implements** Serializable {  
  
 @Id  
 @GeneratedValue  
 **private long oid**;  
  
 **private long place**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 @JoinColumn(name = **"PATH\_ID"**)  
 **private** Path **path**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 @JoinColumn(name = **"WORD"**)  
 **private** Word **word**;  
  
 **public** Occurrence(Word word, Path file, **int** place) {  
 **this**.**word** = word;  
 **this**.**path** = file;  
 **this**.**place** = place;  
 }  
  
 **public** Occurrence() {  
 }  
  
  
 **public** Path getPath() {  
 **return path**;  
 }  
  
 **public void** setPath(Path path) {  
 **this**.**path** = path;  
 **if** (!path.getOccurrences().contains(**this**)) {  
 path.addOccurrence(**this**);  
 }  
 }  
  
 **public long** getPlace() {  
 **return place**;  
 }  
  
 **public void** setPlace(**int** place) {  
 **this**.**place** = place;  
 }  
  
 **public** Word getWord() {  
 **return word**;  
 }  
  
 **public void** setWord(Word word) {  
 **this**.**word** = word;  
 **if** (!word.getOccurrences().contains(**this**)) {  
 word.addOccurrence(**this**);  
 }  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Occurrence "** + **"["** + **"word="** + (**word** == **null** ? **null** : **word**.getWord())  
 + **", path="** + (**path** == **null** ? **null** : **path**.getPath())  
 + **", placeInFile="** + **place** + **"]"**;  
*// + ", path=" + (path == null ? null : path.getPid()) + "]";* }  
}

* 1. **Класс Path**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 27.03.17 17:14  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages.entities;  
  
**import** java.io.Serializable;  
**import** java.util.Date;  
**import** java.util.HashSet;  
**import** java.util.List;  
**import** java.util.Set;  
**import** javax.persistence.Column;  
**import** javax.persistence.Entity;  
**import** javax.persistence.FetchType;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.GenerationType;  
**import** javax.persistence.Id;  
**import** javax.persistence.OneToMany;  
**import** javax.persistence.PrePersist;  
**import** javax.persistence.PreUpdate;  
**import** javax.transaction.Transactional;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 27.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/*@Entity  
**public class** Path **implements** Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 **private int pid**;  
 @Column(columnDefinition = **"TIMESTAMP DEFAULT CURRENT\_TIMESTAMP"**)  
 **private** Date **updated**;  
 @OneToMany(orphanRemoval = **true**, mappedBy = **"path"**, fetch = FetchType.***LAZY***)  
 **private** Set<Occurrence> **occurrences** = **new** HashSet<>();  
 ;  
 **private** String **path**;  
  
 **public** Path(String path) {  
 **this**.**path** = path;  
 }  
  
 **public** Path() {  
 }  
  
 **public void** addOccurrence(Occurrence occurrence) {  
 **occurrences**.add(occurrence);  
 **if** (occurrence.getPath() != **this**) {  
 occurrence.setPath(**this**);  
 }  
 }  
  
 **public** Set<Occurrence> getOccurrences() {  
 **return occurrences**;  
 }  
  
 **public** String getPath() {  
 **return path**;  
 }  
  
 **public void** setPath(String path) {  
 **this**.**path** = path;  
 }  
  
 **public** Date getUpdated() {  
 **return updated**;  
 }  
  
 **public void** setUpdated(Date updated) {  
 **this**.**updated** = updated;  
 }  
  
 @PreUpdate  
 @PrePersist  
 **public void** updateTimeStamps() {  
 **updated** = **new** Date();  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Path "** + **"["** + **"path="** + **path** + **", updated="** + **updated** + **"]"**;  
 }  
}

* 1. **Класс Word**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 27.03.17 19:09  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages.entities;  
  
**import** java.io.Serializable;  
**import** java.util.Date;  
**import** java.util.HashSet;  
**import** java.util.Set;  
**import** javax.persistence.Column;  
**import** javax.persistence.Entity;  
**import** javax.persistence.FetchType;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.GenerationType;  
**import** javax.persistence.Id;  
**import** javax.persistence.OneToMany;  
**import** javax.persistence.PrePersist;  
**import** javax.persistence.PreUpdate;  
**import** javax.persistence.Temporal;  
**import** javax.persistence.TemporalType;  
**import** javax.transaction.Transactional;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 27.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/*@Entity  
**public class** Word **implements** Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 **private long wid**;  
 @Temporal(TemporalType.***TIMESTAMP***)  
 **private** Date **lastModified**;  
  
 *// @Id  
// @Column(unique = true)* **private** String **word**;  
  
 @OneToMany(orphanRemoval = **true**, mappedBy = **"word"**, fetch = FetchType.***LAZY***)  
 **private** Set<Occurrence> **occurrences** = **new** HashSet<>();  
 ;  
  
 **public** Word() {  
 }  
  
 **public** Word(String word) {  
 **this**.**word** = word;  
 }  
  
 **public** String getWord() {  
 **return word**;  
 }  
  
 **public void** addOccurrence(Occurrence occurrence) {  
 **occurrences**.add(occurrence);  
 **if** (occurrence.getWord() != **this**) {  
 occurrence.setWord(**this**);  
 }  
 }  
  
 @PreUpdate  
 @PrePersist  
 **public void** updateTimeStamps() {  
 **lastModified** = **new** Date();  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Word [word="** + **word** + **", lastModified="** + (**lastModified** == **null** ? **null** : **lastModified**) + **"]"**;  
 }  
  
 **public** Set<Occurrence> getOccurrences() {  
 **return occurrences**;  
 }  
}

* 1. **Класс ComplexNode**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 14:35  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** java.io.Serializable;  
**import** java.util.HashMap;  
**import** java.util.HashSet;  
**import** java.util.LinkedList;  
**import** java.util.Map;  
**import** java.util.Optional;  
**import** java.util.Set;  
**import** java.util.stream.Collectors;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 04.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** ComplexNode **implements** Node {  
  
 **private final** LinkedList<NodeRow> **rows**;  
  
 **private** ComplexNode() {  
 **rows** = **new** LinkedList<>();  
 }  
  
 ComplexNode(String filepath, **int** description) {  
 **this**();  
 add(filepath, description);  
 }  
  
 @Override  
 **public void** add(String filepath, **int** description) {  
 Optional<NodeRow> found;  
 **synchronized** (**rows**) {  
 found = **rows**.stream().filter(nodeRow -> nodeRow.**filepath**.equals(filepath)).findFirst();  
  
 **if** (found.isPresent()) {  
 found.get().add(description);  
 } **else** {  
 rows.add(**new** NodeRow(filepath, description));  
 }  
 }  
 }  
  
 @Override  
 **public** Set<String> files() {  
 **synchronized** (rows) {  
 **return** rows.stream().map(nodeRow -> nodeRow.filepath).collect(Collectors.toSet());  
 }  
 }  
  
 @Override  
 **public** Map<String, Set<Integer>> filesToPos() {  
 **synchronized** (rows) {  
 Map<String, Set<Integer>> result = **new** HashMap<>();  
 rows.stream().forEach(nodeRow -> {  
 result.put(nodeRow.filepath, nodeRow.rows);  
 });  
 **return** result;  
 }  
 }  
  
}  
  
**class** NodeRow **implements** Serializable {  
  
 String filepath;  
 HashSet<Integer> rows;  
  
 **private** NodeRow() {  
 rows = **new** HashSet<>();  
 }  
  
 NodeRow(String filepath, **int** row) {  
 **this**();  
 **this**.filepath = filepath;  
 rows.add(row);  
 }  
  
 **protected void** add(**int** row) {  
 rows.add(row);  
 }  
}

* 1. **Класс H2Storage**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 27.03.17 21:58  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** index.IndexParameters;  
**import** index.IndexStorageWithLevels;  
**import** index.SearchRequest;  
**import** index.Storages.entities.Occurrence;  
**import** index.Storages.entities.Path;  
**import** index.Storages.entities.Word;  
**import** java.nio.file.Paths;  
**import** java.util.Date;  
**import** java.util.HashMap;  
**import** java.util.HashSet;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.Set;  
**import** java.util.stream.Collectors;  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
**import** javax.persistence.LockModeType;  
**import** javax.persistence.OptimisticLockException;  
**import** javax.persistence.Persistence;  
**import** javax.persistence.Query;  
**import** javax.persistence.RollbackException;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 27.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** H2Storage **extends** IndexStorageWithLevels {  
  
 **private final** EntityManagerFactory **managerFactory**;  
 **public static final int *BATCH\_SIZE*** = 130;  
 **public static final int *MAX\_TRIES*** = 10;  
  
 **public** H2Storage(IndexParameters parameters, String name) {  
 **super**(parameters);  
  
*// managerFactory = Persistence.createEntityManagerFactory("myDbFile.odb");* Map<String, Object> properties = **new** HashMap<>();  
 properties.put(**"javax.persistence.jdbc.driver"**, **"org.h2.Driver"**);  
 properties.put(**"javax.persistence.jdbc.url"**, **"jdbc:h2:file:./indices/"** + name + **";MVCC=true"**);  
 properties.put(**"javax.persistence.jdbc.user"**, **"sa"**);  
 properties.put(**"javax.persistence.jdbc.password"**, **""**);  
 properties.put(**"hibernate.dialect"**, **"org.hibernate.dialect.H2Dialect"**);  
 properties.put(**"hibernate.hbm2ddl.auto"**, **"update"**);  
 properties.put(**"javax.persistence.lock.timeout"**, 15000);  
*// properties.put("hibernate.show\_sql", "true");* **managerFactory** = Persistence.*createEntityManagerFactory*(**"MainEntities"**, properties);  
 }  
  
 **public void** put(Set<Inclusion> inclusions, **int** tryNumber) {  
 **if** (inclusions.size() <= 0) {  
 **return**;  
 }  
 EntityManager manager = createEntityManager();  
 **try** {  
 manager.getTransaction().begin();  
 **int** counter = 0;  
 Path path = manager.merge(**new** Path(inclusions.toArray(**new** Inclusion[]{})[0].getPath().toString()));  
 **for** (Inclusion inclusion : inclusions) {  
*// manager.lock(path, LockModeType.PESSIMISTIC\_WRITE);* Word word = manager.merge(**new** Word(inclusion.getWord()));  
*// manager.lock(word, LockModeType.PESSIMISTIC\_WRITE);* Occurrence occurrence = **new** Occurrence();  
 occurrence.setWord(word);  
 occurrence.setPath(path);  
 occurrence.setPlace((**int**) inclusion.getPlace());  
 manager.persist(occurrence);  
 counter += 1;  
 **if** (counter % ***BATCH\_SIZE*** == 0) {  
 counter = 0;  
 manager.flush();  
 manager.clear();  
 }  
 }  
 **if** (!manager.isOpen()) {  
 **return**;  
 }  
 manager.getTransaction().commit();  
 manager.close();  
 } **catch** (RollbackException e) {  
 manager.getTransaction().rollback();  
 **if** (tryNumber >= ***MAX\_TRIES***) {  
 e.printStackTrace();  
 } **else** {  
 put(inclusions, ++tryNumber);  
 }  
 } **catch** (Exception e) {  
*// e.printStackTrace();* }  
 }  
  
 @Override  
 **public void** put(String key, String filepath, **int** description) {  
 **try** {  
 EntityManager manager = **managerFactory**.createEntityManager();  
 manager.getTransaction().begin();  
  
 Path path = manager.merge(**new** Path(filepath));  
 Word word = manager.merge(**new** Word(key));  
 Occurrence occurrence = **new** Occurrence();  
 occurrence.setPlace(description);  
 occurrence.setWord(word);  
 occurrence.setPath(path);  
 manager.persist(occurrence);  
  
 manager.getTransaction().commit();  
 manager.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 **public long** remove(Set<java.nio.file.Path> temp) {  
 EntityManager manager = **managerFactory**.createEntityManager();  
 manager.getTransaction().begin();  
 **long** deleted = 0;  
 **for** (java.nio.file.Path path : temp) {  
 *// here we can see leak in word table* List<Path> paths = manager.createQuery(**"SELECT p FROM Path p WHERE p.path LIKE :path"**)  
 .setParameter(**"path"**, path.toString().replace(**"\\"**, **"\\\\"**) + **"%"**)  
 .getResultList();  
 **for** (Path realPath : paths) {  
 manager.remove(realPath);  
 }  
 }  
 manager.getTransaction().commit();  
 manager.close();  
 **return** deleted;  
 }  
  
 **public** EntityManager createEntityManager() {  
 **if** (!**managerFactory**.isOpen()) {  
 **return null**;  
 }  
 **return managerFactory**.createEntityManager();  
 }  
  
 @Override  
 **protected** Set<String> getKeys() {  
 EntityManager manager = **managerFactory**.createEntityManager();  
 List<Word> resultSet = (List<Word>) manager.createQuery(**"SELECT w FROM Word w"**).getResultList();  
 manager.close();  
 **return** resultSet.stream().map(Word::getWord).collect(Collectors.*toSet*());  
 }  
  
 @Override  
 **protected** Set<Inclusion> get(String key) {  
 EntityManager manager = **managerFactory**.createEntityManager();  
 **if** (!manager.isOpen()) {  
 **return null**;  
 }  
  
 List<Occurrence> resultList = (List<Occurrence>) manager  
 .createQuery(**"SELECT o FROM Occurrence o WHERE o.word.word = :word"**)  
 .setParameter(**"word"**, key)  
 .getResultList();  
  
 Set<Inclusion> inclusions = **new** HashSet<>();  
 **for** (Occurrence entry : resultList) {  
 inclusions  
 .add(  
 **new** Inclusion(key, Paths.get(entry.getPath().getPath()), entry.getPlace(), entry.getPath().getUpdated()));  
 }  
 manager.close();  
  
 **return** inclusions;  
 }  
  
 **protected void** get(SearchRequest request) {  
 **try**{  
 EntityManager manager = managerFactory.createEntityManager();  
 **if** (!manager.isOpen()) {  
 **return**;  
 }  
  
 Query query = manager.createQuery(  
 **"SELECT o.word.word, o.path.path, o.place, o.path.updated FROM Occurrence o WHERE o.word.word = :word"**);  
 query.setParameter(**"word"**, request.getSearchFor());  
 List<Object[]> found = query.getResultList();  
  
 Set<Inclusion> compressed = found.stream().map(  
 objects -> **new** Inclusion((String) objects[0], Paths.*get*((String) objects[1]), (**long**) objects[2],  
 (Date) objects[3])).collect(Collectors.*toSet*());  
  
 request.addResult(compressed);  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }  
  
 }  
  
 @Override  
 **protected void** searchConcrete(SearchRequest request) {  
 get(request);  
*// request.addResult(get(request.getSearchFor()));* }  
  
 @Override  
 **public void** exit() {  
 **managerFactory**.close();  
 }  
  
 @Override  
 **public void** changeName(String name) {  
 **super**.changeName(name);  
  
 }  
}

* 1. **Класс Inclusion**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 28.03.17 16:36  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** java.nio.file.Path;  
**import** java.util.Date;  
  
*/\*\*  
 \* Represents  
 \*/***public class** Inclusion {  
  
 **private** String **word**;  
 **private long place**;  
 **private** Date **updated**;  
 **private** Path **file**;  
  
 **public** Inclusion(String word, Path file, **long** place, Date updated) {  
 **this**.**word** = word;  
 **this**.**place** = place;  
 **this**.**updated** = updated;  
 **this**.**file** = file;  
 }  
  
 **public** Date getUpdated() {  
 **return updated**;  
 }  
  
 **public long** getPlace() {  
 **return place**;  
 }  
  
 **public** Path getPath() {  
 **return file**;  
 }  
  
 **public** String getWord() {  
 **return word**;  
 }  
  
 @Override  
 **public boolean** equals(Object other) {  
 **if** (other == **null**) {  
 **return false**;  
 }  
 **if** (other == **this**) {  
 **return true**;  
 }  
 **if** (!(other **instanceof** Inclusion)) {  
 **return false**;  
 }  
 Inclusion otherMyClass = (Inclusion) other;  
 **boolean** worked = **word** == **null** ? **null** == otherMyClass.**word** : **word**.equals(otherMyClass.**word**);  
 worked &= **place** == otherMyClass.**place**;  
 worked &= **file** == **null** ? **null** == otherMyClass.**file** : **file**.equals(otherMyClass.**file**);  
 **return** worked;  
 }  
}

* 1. **Класс InvertedIndex**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 13:38  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** index.IndexParameters;  
**import** index.IndexStorageWithLevels;  
**import** index.Parameter;  
**import** index.SearchRequest;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.HashMap;  
**import** java.util.HashSet;  
**import** java.util.Map.Entry;  
**import** java.util.Set;  
**import** javafx.beans.value.ObservableValue;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 04.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** InvertedIndex **extends** IndexStorageWithLevels {  
  
 **private final** HashMap<String, Node> **storage**;  
  
  
 **public** InvertedIndex(IndexParameters parameters) {  
 **super**(parameters);  
 **storage** = **new** HashMap<>();  
 **this**.**parameters** = parameters;  
 }  
  
 **private** Set<Inclusion> extractFromNode(Node node) {  
 Set<Inclusion> result = **new** HashSet<>();  
 **for** (Entry<String, Set<Integer>> entry : node.filesToPos().entrySet()) {  
 **for** (Integer position : entry.getValue()) {  
 *//* ***todo: add date*** result.add(**new** Inclusion(entry.getKey(), Paths.*get*(entry.getKey()), position, **null**));  
 }  
 }  
 **return** result;  
 }  
  
  
 @Override  
 **public void** put(String word, String filepath, **int** description) {  
 **synchronized** (**storage**) {  
 Node entry = **storage**.get(word);  
 *// check if value doesn't exist - just create a new one* **if** (entry == **null**) {  
 *// check for --FILE\_INDEX--* ObservableValue fileIndexing = **parameters**.get(Parameter.***FILE\_INDEX***);  
 **if** (fileIndexing != **null** && (Boolean) fileIndexing.getValue()) {  
 entry = **new** ComplexNode(filepath, description);  
 } **else** {  
 entry = **new** SimpleNode(filepath);  
 }  
 **storage**.put(word, entry);  
 }  
 *// else - add description to already existing one* **else** {  
 entry.add(filepath, description);  
 }  
 }  
 }  
  
 @Override  
 **protected** Set<String> getKeys() {  
 **synchronized** (**storage**) {  
 **return storage**.keySet();  
 }  
 }  
  
 @Override  
 **protected** Set<Inclusion> get(String key) {  
 **synchronized** (**storage**) {  
 **return** extractFromNode(**storage**.get(key));  
 }  
 }  
  
 @Override  
 **public long** remove(Set<Path> temp) {  
 **return** 0;  
 }  
  
 @Override  
 **protected void** searchConcrete(SearchRequest request) {  
 **synchronized** (**storage**) {  
 request.addResult(extractFromNode(**storage**.get(request.getSearchFor())));  
 }  
 }  
  
 @Override  
 **protected void** searchSimilar(SearchRequest request) {  
 **synchronized** (**storage**) {  
 **for** (Entry<String, Node> entry : **storage**.entrySet()) {  
 **if** (entry.getKey().contains(request.getSearchFor())) {  
 request.addResult(extractFromNode(**storage**.get(entry.getKey())));  
 }  
 }  
 }  
 }  
}

* 1. **Интерфейс Node**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 14:30  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** java.io.Serializable;  
**import** java.util.Map;  
**import** java.util.Set;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 04.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public interface** Node **extends** Serializable {  
  
 **void** add(String filepath, **int** description);  
  
 Set<String> files();  
  
 Map<String, Set<Integer>> filesToPos();  
  
}

* 1. **Класс SimpleNode**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 14:27  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index.Storages;  
  
**import** java.util.HashSet;  
**import** java.util.LinkedList;  
**import** java.util.Map;  
**import** java.util.Set;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 04.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** SimpleNode **implements** Node {  
  
 **private** LinkedList<String> **filepaths**;  
  
 **public** SimpleNode() {  
 **filepaths** = **new** LinkedList<>();  
 }  
  
 **public** SimpleNode(String filepath) {  
 **this**();  
 **filepaths**.add(filepath);  
 }  
  
 @Override  
 **public void** add(String filepath, **int** description) {  
 *// only if found in filename* **if** (description == -1) {  
 **filepaths**.add(filepath);  
 }  
 }  
  
 @Override  
 **public** Set<String> files() {  
 **return new** HashSet<>(**filepaths**);  
 }  
  
 @Override  
 **public** Map<String, Set<Integer>> filesToPos() {  
 **return null**;  
 }  
  
}

* 1. **Класс FileVisitorIndexer**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 03.03.17 15:47  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** java.io.FileNotFoundException;  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.io.StreamTokenizer;  
**import** java.nio.file.AccessDeniedException;  
**import** java.nio.file.FileVisitResult;  
**import** java.nio.file.Path;  
**import** java.nio.file.SimpleFileVisitor;  
**import** java.nio.file.attribute.BasicFileAttributes;  
**import** java.util.Timer;  
**import** java.util.TimerTask;  
**import** java.util.concurrent.ExecutionException;  
**import** java.util.concurrent.ExecutorService;  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.Semaphore;  
**import** java.util.concurrent.TimeUnit;  
**import** java.util.concurrent.atomic.AtomicInteger;  
**import** java.util.concurrent.atomic.AtomicLong;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.collections.ObservableList;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 24.01.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** FileVisitorIndexer **extends** SimpleFileVisitor<Path> {  
  
 **protected final int FILE\_INDEX\_PERMITS** = 3;  
 *// protected final Semaphore semaphore;* **protected final** ExecutorService **service** = Executors.*newFixedThreadPool*(**FILE\_INDEX\_PERMITS**);  
*// protected final Executor executor;* **protected final** String **exclude** = **"[!@#$%^&\*()\_+1234567890-=|/.,<>]"**;  
 **protected** ObservableList **extensions**;  
 **protected** Logger **log**;  
 **protected** IndexingRequest **request**;  
 **protected** IndexParameters **parameters**;  
 **protected** IndexStorage **storage**;  
 *//* ***todo: delete, only in debug* private** AtomicLong **counter** = **new** AtomicLong();  
 **private** Timer **timer**;  
  
 **public** FileVisitorIndexer(IndexingRequest request) {  
 **this**.**request** = request;  
 **this**.**parameters** = request.getIndex().getParameters();  
 **this**.**storage** = request.getIndex().getStorage();  
 **this**.**extensions** = (ObservableList) **parameters**.getStorage().get(Parameter.***FORMATS***);  
 **this**.**log** = Logger.*getLogger*(FileVisitorIndexer.**class**.getName());  
*// this.executor =* **timer** = **new** Timer();  
 **counter** = **new** AtomicLong();  
 **timer**.schedule(**new** TimerTask() {  
 @Override  
 **public void** run() {  
 System.***out***.println(**"Indexed: "** + **counter**.getAndSet(0));  
 }  
 }, 0, 30000);  
 }  
  
 **private static** String getExtension(String filename) {  
 **if** (filename == **null**) {  
 **return null**;  
 }  
 **int** extensionPos = filename.lastIndexOf(**"."**);  
 **int** lastUnixPos = filename.lastIndexOf(**"/"**);  
 **int** lastWindowsPos = filename.lastIndexOf(**"\\"**);  
 **int** lastSeparator = Math.*max*(lastUnixPos, lastWindowsPos);  
  
 **int** index = lastSeparator > extensionPos ? -1 : extensionPos;  
 **if** (index == -1) {  
 **return ""**;  
 } **else** {  
 **return** filename.substring(index + 1);  
 }  
 }  
  
 @Override  
 **public** FileVisitResult preVisitDirectory(Path path, BasicFileAttributes attrs)  
 **throws** IOException {  
 **if** (**request**.isCancelled()) {  
 **return** FileVisitResult.***TERMINATE***;  
 }  
 **return** FileVisitResult.***CONTINUE***;  
 }  
  
 @Override  
 **public** FileVisitResult visitFile(Path file, BasicFileAttributes attrs) **throws** IOException {  
 *// terminate mechanism* **if** (**request**.isCancelled()) {  
 **return** FileVisitResult.***TERMINATE***;  
 }  
  
 *// index filepath and optionally content (if property is set to true)* **try** {  
 indexNameAndContent(file);  
 } **catch** (ExecutionException | InterruptedException e) {  
 **if** (e.getCause() **instanceof** AccessDeniedException) {  
 log.finest(**"File <"** + file.toString() + **"> protected."**);  
 } **else** {  
 log.fine(**"Indexing file <"** + file.toString() + **"> failed: "** + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
 **return** FileVisitResult.CONTINUE;  
 }  
  
 **private void** indexNameAndContent(Path file) **throws** ExecutionException, InterruptedException {  
 *// index filename and separately - extension* String filename = file.getFileName().toString(); *// pre-ready filename with extension* String extension = getExtension(filename); *// file extension* filename = filename.replaceAll(extension, **""**); *// excluding extension from filename* indexWord(filename, file.toAbsolutePath().toString(), -1); *// index filename* indexWord(extension, file.toAbsolutePath().toString(), -2); *// index extension  
  
 // check extension --FORMATS--* **if** (**""**.equals(extension)) {  
 request.incrementFileCounter(1);  
 **return**;  
 }  
  
 *// index file content --FILE\_INDEX--* ObservableValue fileIndexing = parameters.get(Parameter.FILE\_INDEX);  
 *// check if parameter exists, is true and extension is acceptable* **if** ((fileIndexing != **null** && (Boolean) fileIndexing.getValue())  
 && (extensions.contains(extension) || extensions.contains(**"\*"**))) {  
 indexFile(file);  
 } **else** {  
 request.incrementFileCounter(1);  
 }  
 }  
  
 **protected void** indexFile(Path file) {  
 service.submit(() -> {  
 **try** {  
 String filepath = file.toAbsolutePath().toString();  
 *// acquire semaphore for file index operation* FileReader reader = **new** FileReader(file.toFile());  
 StreamTokenizer tokenizer = **new** StreamTokenizer(reader);  
 *// set to lowercase, cause it is no reason in separating lowercase and simple words* tokenizer.lowerCaseMode(**true**);  
  
 **int** token = -1;  
 **while** ((token = tokenizer.nextToken()) != StreamTokenizer.TT\_EOF) {  
 **switch** (token) {  
 *// checking --NUMBERS--* **case** StreamTokenizer.TT\_NUMBER: {  
 *// check for --NUMBERS--* ObservableValue numberIndexing = parameters.get(Parameter.NUMBERS);  
 *// if it exists and is true* **if** (numberIndexing != **null** && (Boolean) numberIndexing.getValue()) {  
 indexWord(String.valueOf(tokenizer.nval), filepath, tokenizer.lineno());  
 }  
 **break**;  
 }  
 **case** StreamTokenizer.TT\_WORD: {  
 *// checking --WORDS--* ObservableValue wordsIndexing = parameters.get(Parameter.WORDS);  
 *// if it exists and is true* **if** (wordsIndexing != **null** && (Boolean) wordsIndexing.getValue()) {  
 indexWord(tokenizer.sval, filepath, tokenizer.lineno());  
 }  
 **break**;  
 }  
 }  
 }  
  
 request.incrementFileCounter(1);  
 } **catch** (InterruptedException e) {  
 **if** (request.isCancelled()) {  
 **return**;  
 }  
 log.log(Level.FINE, **"File indexing interrupted: {0}"**, file.toString());  
 } **catch** (FileNotFoundException e) {  
 log.log(Level.FINE, **"File not found: {}"**, file.toString());  
 } **catch** (IOException | ExecutionException e) {  
 **if** (e.getCause() **instanceof** AccessDeniedException) {  
 **return**;  
 }  
 log.log(Level.FINE, **"File cannot be opened: {0}"**, file.toString());  
 }  
 });  
 }  
  
 **protected** String handle(String word) {  
 counter.incrementAndGet();  
 String temp = word.toLowerCase().replaceAll(exclude, **""**);  
 **if** (temp.length() > 250) {  
 temp = temp.substring(0, 250);  
 }  
 **return** temp;  
 }  
  
 **private void** indexWord(String word, String filepath, **int** description) {  
 */\*  
 \* description:  
 \* -2: File extension  
 \* -1: Filename  
 \* 0>=: Line number  
 \*/* word = handle(word);  
 **if** (!**""**.equals(word)) {  
 storage.put(word, filepath, description);  
 }  
 }  
  
 @Override  
 **public** FileVisitResult visitFileFailed(Path file, IOException exc) **throws** IOException {  
 *// unavailable file - just skip* **if** (!(exc **instanceof** AccessDeniedException)) {  
 log.log(Level.SEVERE, **"File visit failed: {0}\nReason: {1}"**,  
 **new** Object[]{file.toAbsolutePath().toString(), exc.getMessage()});  
 }  
 **return** FileVisitResult.CONTINUE;  
 }  
  
 **public void** waitUntilQueueEnds() {  
 **try** {  
 service.shutdown();  
 service.awaitTermination(Long.MAX\_VALUE, TimeUnit.SECONDS);  
 } **catch** (InterruptedException e) {  
 **if** (request.isCancelled()){  
 **return**;  
 }  
 log.severe(**"Error in waiting for task indexed: "** + e.getMessage());  
*// e.printStackTrace();* }  
 }  
  
 **public void** stopCounter() {  
 System.out.println(**"Indexed: "** + counter.getAndSet(0));  
 timer.cancel();  
 timer.purge();  
 }  
}

* 1. **Класс FileVisitorIndexerDB**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 10.04.17 0:44  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** index.Storages.H2Storage;  
**import** index.Storages.Inclusion;  
**import** index.Storages.entities.Occurrence;  
**import** index.Storages.entities.Word;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.io.StreamTokenizer;  
**import** java.nio.file.AccessDeniedException;  
**import** java.nio.file.Path;  
**import** java.util.HashSet;  
**import** java.util.LinkedList;  
**import** java.util.List;  
**import** java.util.Set;  
**import** java.util.concurrent.ExecutionException;  
**import** java.util.logging.Level;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 29.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** FileVisitorIndexerDB **extends** FileVisitorIndexer {  
  
 *// 130 is a good size for batch based on testing value [10,400] with interval 10.  
 // it's performance is around the best* **public static int** *BATCHING\_SIZE* = 130;  
 **private** H2Storage **h2** = (H2Storage) **storage**;  
 **private boolean numbers** =  
 **parameters**.get(Parameter.***NUMBERS***) != **null** && (**boolean**) **parameters**.get(Parameter.***NUMBERS***).getValue();  
 **private boolean words** =  
 **parameters**.get(Parameter.***WORDS***).getValue() != **null** && (**boolean**) **parameters**.get(Parameter.***WORDS***).getValue();  
  
 **public** FileVisitorIndexerDB(IndexingRequest request) {  
 **super**(request);  
 }  
  
 @Override  
 **protected void** indexFile(Path file) {  
  
 *// submit a task* **service**.submit(() -> {  
 **try** {  
 *// config streams* FileReader reader = **new** FileReader(file.toFile());  
 StreamTokenizer tokenizer = **new** StreamTokenizer(reader);  
 *// set to lowercase, cause it is no reason in separating lowercase and simple words* tokenizer.lowerCaseMode(**true**);  
  
 Set<Inclusion> inclusions = **new** HashSet<>();  
 **int** token;  
 **while** ((token = tokenizer.nextToken()) != StreamTokenizer.***TT\_EOF***) {  
 **if** (**request**.isCancelled()) {  
 **return**;  
 }  
 String string\_token = **null**;  
 **switch** (token) {  
 *// checking --NUMBERS--* **case** StreamTokenizer.***TT\_NUMBER***: {  
 *// check for --NUMBERS--  
 // if it exists and is true* **if** (**numbers**) {  
 string\_token = String.*valueOf*(tokenizer.**nval**);  
 }  
 **break**;  
 }  
 **case** StreamTokenizer.***TT\_WORD***: {  
 *// checking --WORDS--  
 // if it exists and is true* **if** (**words**) {  
 string\_token = String.*valueOf*(tokenizer.**sval**);  
 }  
 **break**;  
 }  
 }  
 **if** (string\_token != **null**) {  
 inclusions.add(**new** Inclusion(handle(string\_token), file, tokenizer.lineno(), **null**));  
 **if** (inclusions.size() > H2Storage.***BATCH\_SIZE*** \* 3) {  
 **h2**.put(inclusions, 0);  
 inclusions = **new** HashSet<>();  
 }  
 }  
 }  
 **h2**.put(inclusions, 0);  
 **request**.incrementFileCounter(1);  
 } **catch** (InterruptedException e) {  
 **if** (**request**.isCancelled()) {  
 **return**;  
 }  
 **log**.log(Level.***FINE***, **"File indexing interrupted: {0}"**, file.toString());  
 } **catch** (FileNotFoundException e) {  
 **log**.log(Level.***FINE***, **"File not found: {0}"**, file.toString());  
 } **catch** (IOException | ExecutionException e) {  
 **if** (e.getCause() **instanceof** AccessDeniedException) {  
 **return**;  
 }  
 **log**.log(Level.***FINE***, **"File cannot be opened: {0}"**, file.toString());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
}

* 1. **Класс Index**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 02.03.17 22:01  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** index.Storages.H2Storage;  
**import** java.io.File;  
**import** java.io.FileInputStream;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.io.RandomAccessFile;  
**import** java.io.Serializable;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.HashSet;  
**import** java.util.Set;  
**import** java.util.UUID;  
**import** java.util.logging.Logger;  
**import** java.util.stream.Collectors;  
**import** javafx.concurrent.Worker.State;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 02.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** Index **implements** Serializable {  
  
 **public static final** String ***INDICES\_DIRECTORY*** = **"indices\\"**;  
 **public static final** String ***INDEX\_FILE\_EXTENSION*** = **".ser"**;  
  
 *// private static final FSTConfiguration conf = FSTConfiguration.createDefaultConfiguration();* **private transient static** Logger *log* = Logger.*getLogger*(Index.**class**.getName());  
  
 **private transient** IndexingHandler **handler**;  
 **private** IndexStorage **storage**;  
 **private** IndexParameters **parameters**;  
 **private** String **name**;  
 **private** Set<Path> **indexedPaths**;  
  
 *// =============== Constructors* **private** Index() {  
 **this**.**indexedPaths** = **new** HashSet<>();  
 }  
  
 **public** Index(String name, IndexParameters parameters, IndexStorage storage) {  
 **this**();  
 **this**.**parameters** = parameters;  
 **this**.**storage** = storage;  
 **this**.**handler** = **new** IndexingHandler(**this**);  
 **this**.**name** = name;  
 }  
  
 **public** Index(String name, IndexParameters parameters) {  
 *// h2 - default h2* **this**(name, parameters, **new** H2Storage(parameters, name));  
 }  
  
 *// =============== Saving* **public static** Index load(String name) {  
 Index index = **null**;  
 **try** {  
 *// creating streams* FileInputStream fileIn = **new** FileInputStream(**new** RandomAccessFile(name, **"rw"**).getFD());  
 ObjectInputStream in = **new** ObjectInputStream(fileIn);  
  
 *// red name + parameters (it's fast operation)* String indexName = (String) in.readObject();  
 IndexParameters indexParameters = (IndexParameters) in.readObject();  
 Set<String> indexedAsStrings = (Set<String>) in.readObject();  
 Set<Path> indexedPaths = indexedAsStrings.stream().map(s -> Paths.*get*(s)).collect(Collectors.*toSet*());  
 Class storageClass = (Class) in.readObject();  
 *// creating index with specified name + parameters* index = **new** Index(indexName, indexParameters);  
 index.**indexedPaths** = indexedPaths;  
  
 **if** (storageClass == H2Storage.**class**) {  
 *log*.info(**"Index "** + name + **" loaded."**);  
 } **else** {  
 *// make final index* Index finalIndex = index;  
 *// run thread with long operation - loading index h2* **new** Thread(() -> {  
 **try** {  
 *// load h2* finalIndex.**storage** = (IndexStorage) in.readObject();  
 *// say info  
 log*.info(**"Index "** + name + **" loaded."**);  
 } **catch** (IOException | ClassNotFoundException e) {  
 *log*.severe(**"File-Index "** + name + **" was not loaded, problem acquired:"** + e.getMessage());  
 } **finally** {  
 **try** {  
 *// closing streams* fileIn.close();  
 in.close();  
 } **catch** (IOException e) {  
 *log*.fine(  
 **"File-Index "** + name + **" was loaded, but streams were not closed. Problem acquired:"** + e  
 .getMessage());  
 }  
 }  
 }).start();  
 }  
 } **catch** (Exception e) {  
 *log*.severe(**"File-Index "** + name + **" was not loaded, problem acquired:"** + e.getMessage());  
 }  
 **return** index;  
 }  
  
 **public void** save(String directory) {  
 **try** {  
 *// create filename* String filename = directory + name + **".ser"**;  
 *// open streams* FileOutputStream fileOut = **new** FileOutputStream(**new** RandomAccessFile(filename, **"rw"**).getFD());  
 ObjectOutputStream out = **new** ObjectOutputStream(fileOut);  
  
 exit();  
 *// write objects in order - name, parameters, h2* out.writeObject(name);  
 out.writeObject(parameters);  
 Set<String> indexedAsStrings = indexedPaths.stream().map(Path::toString).collect(Collectors.toSet());  
 out.writeObject(indexedAsStrings);  
 out.writeObject(storage.getClass());  
 **if** (!(storage **instanceof** H2Storage)) {  
 out.writeObject(storage);  
 }  
  
 *// close streams* out.flush();  
 out.close();  
 fileOut.flush();  
 fileOut.close();  
 log.info(**"Index "** + name + **" saved to "** + filename);  
 } **catch** (IOException i) {  
 log.severe(**"Index "** + name + **" was not saved, problem acquired:"** + i.getMessage());  
 }  
 }  
  
 **public void** exit() {  
 storage.exit();  
 }  
  
 *// =============== Operations* **public long** index(IndexingRequest request) {  
 log.info(**"Indexing with request \""** + request.getId().toString() + **"\" started"**);  
 *// add paths to history* **indexedPaths**.addAll(request.getPaths());  
 *// index paths* **long** indexed = -1;  
 **try** {  
 indexed = **handler**.index(request);  
 *log*.info(**"Indexing with request \""** + request.getId().toString() + **"\" completed."**);  
 **return** indexed;  
 } **catch** (IOException | InterruptedException e) {  
 **if** (request.isCancelled()) {  
 *log*.info(**"Indexing with request \""** + request.getId().toString() + **"\" cancelled."**);  
 **return** -1;  
 }  
 *log*.severe(**"Indexing path \""** + request.getId().toString() + **"\" interrupted.\n"** + e.getMessage());  
 e.printStackTrace();  
 }  
 **return** indexed;  
 }  
  
 **public long** search(SearchRequest request) {  
 **long** found = -1;  
 **try** {  
 *log*.info(**"Searching with request \""** + request.getSearchFor() + **"\" started"**);  
 found = **storage**.search(request);  
 *log*.info(**"Searching with request \""** + request.getSearchFor() + **"\" completed"**);  
 } **catch** (Exception e) {  
 **if** (request.isCancelled()){  
 *log*.info(**"Searching with request \""** + request.getSearchFor() + **"\" cancelled."**);  
 **return** -1;  
 }  
 e.printStackTrace();  
 }  
  
 **return** found;  
 }  
  
 **public void** remove(Set<Path> temp) {  
 UUID uuid = UUID.randomUUID();  
 log.info(**"Removing paths from collection #"** + uuid + **" started."**);  
 **long** deleted = storage.remove(temp);  
 log.info(**"Removing paths from collection #"** + uuid + **" ended. [deleted="** + deleted + **"]"**);  
 }  
  
 *// =============== Getters/Setters* **public** IndexStorage getStorage() {  
 **return** storage;  
 }  
  
 **public** IndexParameters getParameters() {  
 **return** parameters;  
 }  
  
 **public** String getName() {  
 **return** name;  
 }  
  
 **public** Set<Path> getIndexedPaths() {  
 **return** indexedPaths;  
 }  
  
 **public** Index clone() {  
 Index cloned = **new** Index();  
 cloned.handler = **new** IndexingHandler(cloned);  
 cloned.storage = **this**.storage; *// just a reference, not a real copy* cloned.parameters = **this**.parameters.clone();  
 cloned.name = **this**.name;  
 cloned.indexedPaths = **new** HashSet<>(**this**.indexedPaths);  
 **return** cloned;  
 }  
  
 **public void** changeName(String name) {  
 **if** (**""**.equals(name)) {  
 **throw new** IllegalArgumentException(**"Name is empty"**);  
 }  
 *// should delete all savings* **new** File(INDICES\_DIRECTORY + getName() + INDEX\_FILE\_EXTENSION).delete();  
 *// set new name* **this**.name = name;  
 storage.changeName(name);  
*// throw new UnsupportedOperationException("Not implemented");* }  
}

* 1. **Класс IndexingHandler**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 02.03.17 22:02  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** index.Storages.H2Storage;  
**import** java.io.IOException;  
**import** java.io.InterruptedIOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.util.LinkedList;  
**import** java.util.List;  
**import** java.util.concurrent.Semaphore;  
**import** java.util.logging.Logger;  
  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 02.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** IndexingHandler {  
  
 *// public static final int INDEX\_REQUESTS\_PERMITS = 3;* **protected final** IndexStorage **storage**;  
 **protected final** IndexParameters **parameters**;  
 **protected final** List<Parameter> **available**;  
 *// private final Semaphore semaphore;* Logger **log** = Logger.*getLogger*(IndexingHandler.**class**.getName());  
  
 **public** IndexingHandler(Index index) {  
 *// Available parameters dummy creation* **available** = **new** LinkedList<>();  
  
 *// The number of indexing tasks are not limited and can grow a lot.  
 // That is why cached pool will be a good idea.  
// semaphore = new Semaphore(INDEX\_REQUESTS\_PERMITS, false);  
  
 // Unpack h2 and parameters* **this**.**storage** = index.getStorage();  
 **this**.**parameters** = index.getParameters();  
 }  
  
 **public** List<Parameter> getAvailableParameters() {  
 **return available**;  
 }  
  
 **public long** index(IndexingRequest request) **throws** IOException, InterruptedException {  
 *// semaphore.acquire();* FileVisitorIndexer visitor;  
 **if** (**storage instanceof** H2Storage) {  
 visitor = **new** FileVisitorIndexerDB(request);  
 } **else** {  
 visitor = **new** FileVisitorIndexer(request);  
 }  
 *// start file walking* **for** (Path pathToIndex : request.getPaths()) {  
 Files.*walkFileTree*(pathToIndex, visitor);  
 **if** (request.isCancelled()) {  
 request.setStatus(**"Awaiting to stop."**);  
 **break**;  
 }  
 }  
 **if** (request.isCancelled()) {  
 visitor.waitUntilQueueEnds();  
 visitor.stopCounter();  
 **throw new** InterruptedException(**"Task cancelled"**);  
 } **else**{  
 request.setStatus(**"Indexing file content"**);  
 visitor.waitUntilQueueEnds();  
 }  
 visitor.stopCounter();  
 **return** -1;  
 }  
}

* 1. **Класс IndexingRequest**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 03.03.17 9:09  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.util.Collection;  
**import** java.util.LinkedList;  
**import** java.util.List;  
**import** java.util.UUID;  
**import** java.util.concurrent.Callable;  
**import** java.util.concurrent.ExecutionException;  
**import** java.util.concurrent.FutureTask;  
**import** java.util.concurrent.atomic.AtomicLong;  
**import** java.util.stream.Collectors;  
**import** javafx.application.Platform;  
**import** javafx.beans.binding.Bindings;  
**import** javafx.concurrent.Task;  
  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 03.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** IndexingRequest **extends** Task<Long> {  
  
 **private** FutureTask<Long> **fileCount**;  
 **private** AtomicLong **indexed** = **new** AtomicLong(0);  
  
 **private** UUID **id**;  
 **private** List<Path> **pathsToIndex**;  
 **private** Index **index**;  
  
 **private** IndexingRequest() {  
 **pathsToIndex** = **new** LinkedList<>();  
 **id** = UUID.*randomUUID*();  
 }  
  
 @Override  
 **protected** Long call() **throws** Exception {  
 updateMessage(**"Counting files"**);  
 *// counting files* **fileCount** = countFiles(**pathsToIndex**);  
 **new** Thread(() -> {  
 **fileCount**.run();  
 updateMessage(**"Indexing files"**);  
 }).start();  
 **long** indexed = 0;  
 indexed = **index**.index(**this**);  
 updateProgress(1,1);  
 **return** indexed;  
 }  
  
  
 **public static** Builder getBuilder() {  
 **return new** IndexingRequest().**new** Builder();  
 }  
  
*// public void execute() {  
// new Thread(this::run).start();  
// }  
//  
// public void run() {  
// setState(State.RUNNING);  
// // count total number of files  
// new Thread(this::countFileCount).start();  
// index.index(this);  
// setState(State.COMPLETED);  
// }* **private** FutureTask<Long> countFiles(Collection<Path> paths) {  
 FutureTask<Long> task = **new** FutureTask<Long>(() -> {  
 **long** sum = 0;  
 **for** (Path current : paths) {  
 sum += count(current);  
 }  
 **return** sum;  
 });  
 **return** task;  
 }  
  
 **private long** count(Path path) **throws** IOException {  
 **if** (isCancelled()) {  
 **return** 0;  
 }  
  
 *// if method call is invalid* **if** (path == **null**) {  
 **return** 0;  
 }  
 *// if path is single file* **if** (!path.toFile().isDirectory()) {  
 **return** 1;  
 }  
 **long** sum = 0;  
 **for** (Path iterPath : Files.*list*(path).collect(Collectors.*toList*())) {  
 **if** (iterPath.toFile().isDirectory()) {  
 sum += count(iterPath);  
 } **else** {  
 sum += 1;  
 }  
 }  
 **return** sum;  
 }  
  
 **public** Index getIndex() {  
 **return index**;  
 }  
  
 **public** List<Path> getPaths() {  
 **return pathsToIndex**;  
 }  
  
 **public** UUID getId() {  
 **return id**;  
 }  
  
 **public void** incrementFileCounter(**int** indexed) **throws** ExecutionException, InterruptedException {  
 **long** currentlyIndexed = **this**.**indexed**.addAndGet(indexed);  
 **if** (**fileCount**.isDone()) {  
 updateProgress(currentlyIndexed, **fileCount**.get());  
 }  
 }  
  
 **public void** setStatus(String message) {  
 updateMessage(message);  
 }  
  
 **public class** Builder {  
  
 **private** Builder() {  
 }  
  
 **public** Builder setIndex(Index targetIndex) {  
 IndexingRequest.**this**.**index** = targetIndex;  
 **return this**;  
 }  
  
 **public** Builder addPath(Path path) {  
 IndexingRequest.**this**.**pathsToIndex**.add(path);  
 **return this**;  
 }  
  
 **public** Builder addPaths(Collection<Path> paths) {  
 IndexingRequest.**this**.**pathsToIndex**.addAll(paths);  
 **return this**;  
 }  
  
 **public boolean** checkPrepared() {  
 **if** (**index** != **null** && **pathsToIndex** != **null** && **pathsToIndex**.size() > 0) {  
 **return true**;  
 }  
 **return false**;  
 }  
  
 **public** IndexingRequest build() {  
 **if** (!checkPrepared()) {  
 **return null**;  
 }  
 **return** IndexingRequest.**this**;  
 }  
 }  
}

* 1. **Класс IndexParameters**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 02.03.17 22:04  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** java.io.IOException;  
**import** java.io.Serializable;  
**import** java.util.Arrays;  
**import** java.util.Collection;  
**import** java.util.HashMap;  
**import** java.util.LinkedList;  
**import** java.util.Map.Entry;  
**import** javafx.beans.property.SimpleBooleanProperty;  
**import** javafx.beans.property.SimpleListProperty;  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.collections.ObservableMap;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 02.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** IndexParameters **implements** Serializable, Cloneable {  
  
 **private transient** ObservableMap<Parameter, ObservableValue> **storage**;  
  
 **public** IndexParameters(Collection<Parameter> available) {  
 **storage** = FXCollections.*observableMap*(**new** HashMap<>());  
 addAll(available);  
 }  
  
 **public** IndexParameters() {  
 **this**(Arrays.*asList*(Parameter.*values*()));  
 }  
  
 **public** ObservableValue get(Parameter parameter) {  
 **return storage**.get(parameter);  
 }  
  
 **private void** addAll(Collection<Parameter> collection) {  
 **for** (Parameter parameter : collection) {  
 **storage**.put(parameter, parameter.getDefaultValue());  
 }  
 }  
  
 **public** ObservableMap<Parameter, ObservableValue> getStorage() {  
 **return storage**;  
 }  
  
 **public** IndexParameters clone() {  
 IndexParameters cloned = **new** IndexParameters();  
 **for** (Entry<Parameter, ObservableValue> entry : **storage**.entrySet()) {  
 Object value = entry.getValue();  
 **if** (value **instanceof** SimpleBooleanProperty) {  
 cloned.**storage**.put(entry.getKey(), **new** SimpleBooleanProperty(((SimpleBooleanProperty) value).get()));  
 } **else if** (value **instanceof** SimpleListProperty) {  
 cloned.**storage**.put(entry.getKey(), **new** SimpleListProperty(((SimpleListProperty) value).get()));  
 }  
 }  
 **return** cloned;  
 }  
  
 @Override  
 **public boolean** equals(Object other) {  
 **if** (other == **null**) {  
 **return false**;  
 }  
 **if** (other == **this**) {  
 **return true**;  
 }  
 **if** (!(other **instanceof** IndexParameters)) {  
 **return false**;  
 }  
 IndexParameters otherParameters = (IndexParameters) other;  
 **for** (Entry<Parameter, ObservableValue> entry : **storage**.entrySet()) {  
 **if** (!otherParameters.**storage**.containsKey(entry.getKey())) {  
 **return false**;  
 }  
 **boolean** result = otherParameters.**storage**.get(entry.getKey()).getValue().equals(entry.getValue().getValue());  
 **if** (!result) {  
 **return false**;  
 }  
 }  
 **return true**;  
*// return this.h2.equals(((IndexParameters) other).h2);* }  
  
 *// ---------- Serialization --------------* **private void** writeObject(java.io.ObjectOutputStream out) **throws** IOException {  
 HashMap<Parameter, ObservableValue> map = **new** HashMap<>();  
 (**storage**.entrySet()).forEach(parameterObservableValueEntry -> {  
 map.put(parameterObservableValueEntry.getKey(), parameterObservableValueEntry.getValue());  
 });  
  
 HashMap<Parameter, ParameterValueSerialization> serializationMap = **new** HashMap<>();  
 map.forEach((parameter, observableValue) -> {  
 ParameterValueSerialization value = **null**;  
 **if** (parameter.getType() == 0) {  
 value = **new** ParameterValueBoolean((Boolean) observableValue.getValue());  
 } **else if** (parameter.getType() == 2) {  
 LinkedList<String> temp = **new** LinkedList();  
 ((ObservableList<String>) observableValue.getValue()).forEach(temp::add);  
 value = **new** ParameterValueList(temp);  
 }  
 serializationMap.put(parameter, value);  
 });  
  
 out.writeObject(serializationMap);  
 }  
  
 **private void** readObject(java.io.ObjectInputStream in) **throws** IOException {  
 **try** {  
 HashMap<Parameter, ParameterValueSerialization> serializationMap;  
 serializationMap = (HashMap<Parameter, ParameterValueSerialization>) in.readObject();  
  
 HashMap<Parameter, ObservableValue> map = **new** HashMap<>();  
 serializationMap.forEach((parameter, parameterValueSerialization) -> {  
 ObservableValue value = **null**;  
 **if** (parameter.getType() == 0) {  
 value = **new** SimpleBooleanProperty(  
 ((ParameterValueBoolean) parameterValueSerialization).**value**);  
 } **else if** (parameter.getType() == 2) {  
 value = **new** SimpleListProperty(FXCollections.*observableList*(  
 ((ParameterValueList) parameterValueSerialization).**value**));  
 }  
  
 map.put(parameter, value);  
 });  
  
 **this**.**storage** = FXCollections.*observableMap*(map);  
 } **catch** (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 **private interface** ParameterValueSerialization **extends** Serializable {  
  
 }  
  
 **private class** ParameterValueBoolean **implements** ParameterValueSerialization {  
  
 **boolean value**;  
  
 ParameterValueBoolean(Boolean value) {  
 **this**.**value** = value;  
 }  
 }  
  
 **private class** ParameterValueList **implements** ParameterValueSerialization {  
  
 LinkedList<String> **value**;  
  
 ParameterValueList(LinkedList value) {  
 **this**.**value** = value;  
 }  
 }  
}

* 1. **Класс IndexStorage**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 02.03.17 22:05  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** index.Storages.Inclusion;  
**import** java.io.Serializable;  
**import** java.nio.file.Path;  
**import** java.util.Set;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 02.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public abstract class** IndexStorage **implements** Serializable {  
  
 **public static final int *SEARCH\_REQUESTS\_PERMITS*** = 5;  
 **protected transient** IndexParameters **parameters**;  
*// protected transient ExecutorService service = Executors.newFixedThreadPool(SEARCH\_REQUESTS\_PERMITS);* **public** IndexStorage(IndexParameters parameters) {  
 **this**.**parameters** = parameters;  
 }  
  
 **public abstract long** search(SearchRequest request);  
  
 **public abstract void** put(String word, String filepath, **int** description);  
  
 **protected abstract** Set<String> getKeys();  
  
 **protected abstract** Set<Inclusion> get(String key);  
  
 **public void** exit() {  
 }  
  
 **public abstract long** remove(Set<Path> temp);  
  
 **public void** changeName(String name) {  
 }  
}

* 1. **Класс IndexStorageWithLevels**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 14:58  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** java.io.IOException;  
**import** java.util.Set;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 04.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public abstract class** IndexStorageWithLevels **extends** IndexStorage {  
  
 **private static transient** Logger *log*;  
  
 **public** IndexStorageWithLevels(IndexParameters parameters) {  
 **super**(parameters);  
 **if** (*log* == **null**) {  
 *log* = Logger.*getLogger*(FileVisitorIndexer.**class**.getName());  
 }  
 }  
  
 @Override  
 **public long** search(SearchRequest request) {  
 searchConcrete(request);  
 **if** (request.isCancelled()) {  
 **return** request.getResult().size();  
 }  
 **if** (request.getSubstringSearch()) {  
 searchSimilar(request);  
 }  
 **if** (request.isCancelled()) {  
 **return** request.getResult().size();  
 }  
 searchStraightInFiles(request);  
 **return** request.getResult().size();  
 }  
  
 **protected void** searchSimilar(SearchRequest request) {  
 Set<String> keys = getKeys();  
 String searchFor = request.getSearchFor();  
 **for** (String key : keys) {  
 *// if request is a part of some key - return node for this key* **if** (key.contains(searchFor)) {  
 request.addResult(get(key));  
 }  
 }  
 }  
  
 **protected void** searchStraightInFiles(SearchRequest request) {  
 *//* ***todo: search in all indexed files*** *log*.log(Level.***FINE***, **"Search in files."**);  
 }  
  
 **protected abstract void** searchConcrete(SearchRequest request);  
  
 **private void** readObject(java.io.ObjectInputStream in) **throws** IOException {  
 **if** (*log* == **null**) {  
 *log* = Logger.*getLogger*(FileVisitorIndexer.**class**.getName());  
 }  
 }  
}

* 1. **Перечисление Parameter**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 04.03.17 1:39  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** java.util.Arrays;  
**import** java.util.LinkedList;  
**import** java.util.List;  
**import** javafx.beans.property.SimpleBooleanProperty;  
**import** javafx.beans.property.SimpleListProperty;  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.collections.FXCollections;  
  
*/\*\*  
 \* Represents all available indexing algorithms parameters. Each algorithm can support not all of  
 \* the parameters, so every algorithm has set of available parameters.  
 \*/***public enum** Parameter {  
 *// MOST VALUABLE --------------* ***FILE\_INDEX***(**true**, **"Index file content, not just name and extension."**),  
 ***NUMBERS***(**true**, **"Index numbers."**),  
 ***WORDS***(**true**, **"Index words."**),  
 ***FORMATS***(**new** LinkedList<>(Arrays.*asList*(**"txt"**, **"xml"**, **"html"**)), **"Extensions that would be indexed."**),  
 ***LANGUAGES***(**new** LinkedList<>(Arrays.*asList*(**"rus"**, **"eng"**)), **"Supported file languages."**);  
  
 *// OPTIMIZATION ---------------  
// LEMMATISATION(false, "test"),  
// TOKENIZATION(false, "test"),  
// STEMMING(false, "test"),  
  
 //CONTINUES OPRIMIZATIONS -----  
// STOPWORDS(false, "test"),  
// WEIGHTING(false, "test");* **private final int type**;  
 **private final** ObservableValue **defaultValue**;  
 **private final** String **description**;  
  
 Parameter(Object defaultValue, String description) {  
 *// needed to define type of element and also initialize with correct-wrapped default value* **if** (defaultValue **instanceof** Boolean) {  
 **type** = 0;  
 **this**.**defaultValue** = **new** SimpleBooleanProperty((Boolean) defaultValue);  
 } **else if** (defaultValue **instanceof** List) {  
 **type** = 2;  
 **this**.**defaultValue** = **new** SimpleListProperty(  
 FXCollections.*observableList*((List<String>) defaultValue));  
 } **else** {  
 **type** = -1;  
 **this**.**defaultValue** = **null**;  
 }  
 **this**.**description** = description;  
 }  
  
 **public int** getType() {  
 **return type**;  
 }  
  
 **public** ObservableValue getDefaultValue() {  
 **if** (**type** == 0) {  
 **return new** SimpleBooleanProperty((Boolean) **defaultValue**.getValue());  
 }  
 **if** (**type** == 2) {  
 **return defaultValue**;  
 }  
 **return defaultValue**;  
 }  
  
 **public** String getDescription() {  
 **return description**;  
 }  
}

* 1. **Интерфейс Request**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 10.04.17 11:02  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** javafx.beans.binding.StringBinding;  
**import** javafx.beans.property.SimpleDoubleProperty;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 10.04.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public interface** Request {  
  
 StringBinding statusProperty();  
  
 SimpleDoubleProperty progressProperty();  
}

* 1. **Класс SearchRequest**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 03.03.17 12:45  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** index;  
  
**import** index.Storages.Inclusion;  
**import** java.util.HashSet;  
**import** java.util.Set;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableSet;  
**import** javafx.concurrent.Task;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 03.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** SearchRequest **extends** Task<Long> {  
  
 **private** String **searchFor**;  
 **private** Index **index**;  
 **private** ObservableSet<Inclusion> **result**;  
 **private boolean substringSearch**;  
  
 **private** SearchRequest() {  
 **result** = FXCollections.*observableSet*(**new** HashSet<Inclusion>());  
 **substringSearch** = **false**;  
 }  
  
 @Override  
 **protected** Long call() **throws** Exception {  
 updateMessage(**"Searching files"**);  
 **long** found = **index**.search(**this**);  
 updateProgress(1, 1);  
 **return** found;  
 }  
  
 **public static** Builder getBuilder() {  
 **return new** SearchRequest().**new** Builder();  
 }  
  
 **public** ObservableSet<Inclusion> getResult() {  
 **return result**;  
 }  
  
 **public void** addResult(Inclusion inclusion) {  
 **if** (!**result**.contains(inclusion)) {  
 **result**.add(inclusion);  
 }  
 }  
  
 **public void** addResult(Set<Inclusion> inclusionSet) {  
 **result**.addAll(inclusionSet);  
 }  
  
 *//================== GETTERS + SETTERS* **public** Index getIndex() {  
 **return index**;  
 }  
  
 **public** String getSearchFor() {  
 **return searchFor**;  
 }  
  
  
 **public** Boolean getSubstringSearch() {  
 **return substringSearch**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return searchFor**;  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (**this** == obj) {  
 **return true**;  
 }  
 **if** (obj == **null**) {  
 **return false**;  
 }  
 **if** (getClass() != obj.getClass()) {  
 **return false**;  
 }  
  
 SearchRequest other = (SearchRequest) obj;  
 **if** (other.getIndex() != **index**) {  
 **return false**;  
 }  
 **if** (other.getSearchFor() == **null** || !other.getSearchFor().equals(getSearchFor())) {  
 **return false**;  
 }  
 **return true**;  
 }  
  
 */\*\*  
 \* 'Builder' pattern, to correctly initialize all fields.  
 \*/* **public class** Builder {  
  
 */\*\*  
 \* Initializing with Pending state - nothing is ready.  
 \*/* **private** Builder() {  
  
 }  
  
 */\*\*  
 \* Setting search for pattern.  
 \*  
 \** ***@param searchFor*** *Search for pattern.  
 \** ***@return*** *this.  
 \*/* **public** Builder setSearchFor(String searchFor) {  
 SearchRequest.**this**.**searchFor** = searchFor;  
 **return this**;  
 }  
  
 */\*\*  
 \* Setting index algorithm  
 \*  
 \** ***@param index*** *Index to search in  
 \** ***@return*** *this.  
 \*/* **public** Builder setIndex(Index index) {  
 SearchRequest.**this**.**index** = index;  
 **return this**;  
 }  
  
 **public** Builder setSubstringSearch(**boolean** substringSearch) {  
 SearchRequest.**this**.**substringSearch** = substringSearch;  
 **return this**;  
 }  
  
 */\*\*  
 \* Indicates if all fields are correctly initialized.  
 \*  
 \** ***@return*** *Initialized correctly or not.  
 \*/* **public boolean** checkPrepared() {  
 **if** (**searchFor** != **null** && !**""**.equals(**searchFor**)  
 && **index** != **null**) {  
 **return true**;  
 }  
 **return false**;  
 }  
  
 */\*\*  
 \* Creates SearchRequest instance with initialized fields.  
 \*  
 \** ***@return*** *Null if something went wrong, else - initialized instance.  
 \*/* **public** SearchRequest build() {  
 **if** (!checkPrepared()) {  
 **return null**;  
 }  
 **return** SearchRequest.**this**;  
 }  
 }  
}

* 1. **Класс Controllers**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 02.04.17 22:59  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** view.controllers;  
  
**import** index.IndexParameters;  
**import** index.Parameter;  
**import** java.util.Map.Entry;  
**import** javafx.beans.property.Property;  
**import** javafx.beans.property.SimpleListProperty;  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.geometry.Insets;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.CheckBox;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListView;  
**import** javafx.scene.control.cell.TextFieldListCell;  
**import** javafx.scene.layout.AnchorPane;  
**import** javafx.scene.layout.Pane;  
**import** javafx.scene.layout.VBox;  
**import** javafx.util.StringConverter;  
**import** javafx.util.converter.DefaultStringConverter;  
**import** javax.swing.event.HyperlinkEvent.EventType;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 02.04.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** Controllers {  
  
 **public static** Node getParamList(IndexParameters parameters) {  
 *// create vbox for scroll view* VBox content = **new** VBox();  
 content.setPadding(**new** Insets(4, 10, 4, 10));  
 content.setSpacing(10);  
  
 *// fill vbox with values* **for** (Entry<Parameter, ObservableValue> parameterEntry : parameters.getStorage().entrySet()) {  
 Node value = **null**;  
 **switch** (parameterEntry.getKey().getType()) {  
 **case** 0: { *// case boolean* CheckBox temp = **new** CheckBox(parameterEntry.getKey().name());  
 temp.selectedProperty().bindBidirectional((Property<Boolean>) parameterEntry.getValue());  
 Label description = **new** Label(parameterEntry.getKey().getDescription());  
 value = **new** VBox(temp, description);  
 **break**;  
 }  
 **case** 2: { *// case list* Label key = **new** Label(parameterEntry.getKey().name());  
 Label description = **new** Label(parameterEntry.getKey().getDescription());  
  
 AnchorPane pane = **new** AnchorPane();  
  
 ListView<String> temp = **new** ListView<>();  
 temp.setEditable(**true**);  
 temp.setMaxHeight(100);  
 temp.setCellFactory(param -> {  
 TextFieldListCell<String> tf = **new** TextFieldListCell(**new** DefaultStringConverter());  
 tf.setEditable(**true**);  
 tf.setConverter(**new** StringConverter<String>() {  
 @Override  
 **public** String toString(String object) {  
 **return** object;  
 }  
  
 @Override  
 **public** String fromString(String string) {  
 **if** (**""**.equals(string)) {  
 temp.getItems().remove(string);  
 }  
 **return** string;  
 }  
 });  
 **return** tf;  
 });  
 temp.setOnEditCommit(event -> {  
 **if** (**""**.equals(event.getNewValue())) {  
 temp.getItems().remove(event.getIndex());  
 } **else** {  
 temp.getItems().set(event.getIndex(), event.getNewValue());  
 }  
 });  
 temp.itemsProperty().bind(parameterEntry.getValue());  
  
 AnchorPane.*setBottomAnchor*(temp, 0d);  
 AnchorPane.*setTopAnchor*(temp, 0d);  
 AnchorPane.*setLeftAnchor*(temp, 0d);  
 AnchorPane.*setRightAnchor*(temp, 0d);  
  
 Button add = **new** Button(**"+"**);  
 add.setOnAction(event -> {  
 ((SimpleListProperty) parameterEntry.getValue()).add(**"input extension here"**);  
 });  
  
 AnchorPane.*setBottomAnchor*(add, 14d);  
 AnchorPane.*setRightAnchor*(add, 14d);  
 add.setMaxHeight(30d);  
 add.setMaxWidth(30d);  
  
 pane.getChildren().addAll(temp, add);  
 pane.setMaxHeight(Double.***MAX\_VALUE***);  
 pane.setMaxWidth(Double.***MAX\_VALUE***);  
  
 value = **new** VBox(key, description, pane);  
 **break**;  
 }  
 }  
  
 **if** (value != **null**) {  
 content.getChildren().add(value);  
 }  
 }  
 **return** content;  
 }  
}

* 1. **Класс IndexCellController**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 17.03.17 13:36  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** view.controllers;  
  
**import** index.Index;  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.util.ResourceBundle;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.layout.BorderPane;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 17.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** IndexCellController {  
  
 **private final** String **INDEX\_CELL\_FXML** = **"/view/fxml/indexCell.fxml"**;  
  
 @FXML  
 **private** ResourceBundle **resources**;  
  
 @FXML  
 **private** URL **location**;  
  
 @FXML  
 **private** Label **lb\_indexName**;  
  
 @FXML  
 **private** Label **lb\_selectionStatus**;  
  
 @FXML  
 **private** BorderPane **bp\_body**;  
  
 **private** Node **graphic**;  
 **private** Index **item**;  
  
  
 **public** IndexCellController() {  
 FXMLLoader fxmlLoader = **new** FXMLLoader(getClass().getResource(**INDEX\_CELL\_FXML**));  
 fxmlLoader.setController(**this**);  
 **try** {  
 **graphic** = fxmlLoader.load();  
 } **catch** (IOException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 @FXML  
 **void** initialize() {  
 **assert lb\_indexName** != **null** : **"fx:id=\"lb\_indexName\" was not injected: check your FXML file 'indexCell.fxml'."**;  
 **assert lb\_selectionStatus** != **null** : **"fx:id=\"lb\_selectionStatus\" was not injected: check your FXML file 'indexCell.fxml'."**;  
 **assert bp\_body** != **null** : **"fx:id=\"bp\_body\" was not injected: check your FXML file 'indexCell.fxml'."**;  
 }  
  
 **public void** setIndex(Index item) {  
 **if** (item == **null**) {  
 **return**;  
 }  
 **this**.**item** = item;  
 **this**.**lb\_indexName**.setText(item.getName());  
 setSelected(**false**);  
 }  
  
 **public** Node getGraphic() {  
 **return graphic**;  
 }  
  
 **public** IndexCellController setSelected(**boolean** selected) {  
 **if** (selected) {  
 **this**.**lb\_selectionStatus**.setText(**"SELECTED"**);  
 } **else** {  
 **this**.**lb\_selectionStatus**.setText(**""**);  
 }  
 **return this**;  
 }  
}

* 1. **Класс IndexCreationController**

**package** view.controllers;  
  
**import** index.Index;  
**import** index.IndexParameters;  
**import** index.Parameter;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.util.ResourceBundle;  
**import** javafx.beans.property.SimpleListProperty;  
**import** javafx.collections.ObservableList;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.ScrollPane;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.layout.HBox;  
**import** javafx.scene.layout.VBox;  
**import** javafx.stage.Stage;  
  
  
**public class** IndexCreationController {  
  
 *// for errors in name* **private final** String **INDEX\_CREATION\_FXML** = **"/view/fxml/indexCreation.fxml"**;  
 **private final** ObservableList<Index> **indices**;  
 **private** IndexParameters **parameters**;  
 **private** Node **view**;  
 **private boolean confirmed**;  
 @FXML  
 **private** ResourceBundle **resources**;  
  
 @FXML  
 **private** URL **location**;  
  
 @FXML  
 **private** Button **btn\_acceptIndexCreation**;  
  
 @FXML  
 **private** Button **btn\_createIndex**;  
  
 @FXML  
 **private** Button **btn\_returnToConfiguring**;  
  
 @FXML  
 **private** HBox **hb\_bottomBox**;  
  
 @FXML  
 **private** ScrollPane **sp\_parameters**;  
  
 @FXML  
 **private** TextField **tf\_indexName**;  
  
  
 **public** IndexCreationController(ObservableList<Index> indices) {  
 **this**.**indices** = indices;  
 }  
  
 @FXML  
 **void** onAffirmIndexCreation(ActionEvent event) {  
 ((SimpleListProperty<String>)**parameters**.getStorage().get(Parameter.***FORMATS***)).removeIf(o -> ((String) o).contains(**"input extension here"**));  
 ((SimpleListProperty<String>)**parameters**.getStorage().get(Parameter.***LANGUAGES***)).removeIf(o -> ((String) o).contains(**"input extension here"**));  
  
 *// disable view* **view**.setDisable(**true**);  
 *// confirm index creation* **confirmed** = **true**;  
 *// return to main window* ((Stage) (**view**.getScene().getWindow())).close();  
 }  
  
 @FXML  
 **void** onCreateIndex(ActionEvent event) {  
 String temp\_name = **tf\_indexName**.getText();  
 *// if name is not empty && is not (in runtime ||on disk) - go to confirmation* **if** (!**""**.equals(temp\_name) && nameAvailable(temp\_name)) {  
 **tf\_indexName**.getStyleClass().remove(**"error"**);  
 toConfirmation();  
 } **else** {  
 **tf\_indexName**.getStyleClass().add(**"error"**);  
 }  
 }  
  
 @FXML  
 **void** onReturnToConfiguring(ActionEvent event) {  
 toEditing();  
 }  
  
 **private void** toConfirmation() {  
 inputEnabled(**false**);  
 }  
  
 **private void** toEditing() {  
 inputEnabled(**true**);  
 }  
  
 **private void** inputEnabled(**boolean** enabled) {  
 *// set name changing disabled* **tf\_indexName**.setEditable(enabled);  
  
 *// disable all parameter choosers* VBox sp\_content = (VBox) **sp\_parameters**.getContent();  
 **for** (Node node : sp\_content.getChildren()) {  
 node.setDisable(!enabled);  
 }  
  
 *// configure buttons* **btn\_returnToConfiguring**.setDisable(enabled);  
 **btn\_acceptIndexCreation**.setDisable(enabled);  
 **btn\_createIndex**.setDisable(!enabled);  
  
 *// configure buttons visibility  
 // create index button visibility* **btn\_createIndex**.setVisible(enabled);  
 **btn\_createIndex**.setManaged(enabled);  
 *// returnToConfig button visibility* **btn\_returnToConfiguring**.setVisible(!enabled);  
 **btn\_returnToConfiguring**.setManaged(!enabled);  
 *// acceptIndexCreation button visibility* **btn\_acceptIndexCreation**.setVisible(!enabled);  
 **btn\_acceptIndexCreation**.setManaged(!enabled);  
 }  
  
 **private boolean** nameAvailable(String name) {  
 **boolean** fileInMemory = **new** File(MainController.***INDICES\_DIRECTORY***, name).exists();  
 **boolean** fileInRuntime = **indices**.stream().anyMatch(index -> index.getName().equals(name));  
 **return** !(fileInMemory || fileInRuntime);  
 }  
  
 @FXML  
 **void** initialize() {  
 **assert btn\_acceptIndexCreation** != **null** : **"fx:id=\"btn\_acceptIndexCreation\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
 **assert btn\_createIndex** != **null** : **"fx:id=\"btn\_createIndex\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
 **assert btn\_returnToConfiguring** != **null** : **"fx:id=\"btn\_returnToConfiguring\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
 **assert hb\_bottomBox** != **null** : **"fx:id=\"hb\_bottomBox\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
 **assert sp\_parameters** != **null** : **"fx:id=\"sp\_parameters\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
 **assert tf\_indexName** != **null** : **"fx:id=\"tf\_indexName\" was not injected: check your FXML file 'indexCreation.fxml'."**;  
  
 **parameters** = **new** IndexParameters();  
 *// initialize paramlist* initializeParamList();  
  
 *// validation textfield - index name* **tf\_indexName**.focusedProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue) {  
 **tf\_indexName**.getStyleClass().remove(**"error"**);  
 }  
 });  
  
 *// start with editing* toEditing();  
 }  
  
 **private void** initializeParamList() {  
 VBox paramList = (VBox) Controllers.*getParamList*(**parameters**);  
 **sp\_parameters**.widthProperty().addListener((observable, oldValue, newValue) -> {  
 paramList.prefWidthProperty().setValue(newValue.intValue() - 20);  
 });  
 **sp\_parameters**.setContent(paramList);  
 }  
  
 **public** Node getView() {  
 **if** (**view** == **null**) {  
 **try** {  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(**INDEX\_CREATION\_FXML**));  
 loader.setController(**this**);  
 **view** = loader.load();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return view**;  
 }  
  
 **public** IndexParameters getCreatedParameters() {  
 *// if confirmed return parameters* **if** (**confirmed**) {  
 **return parameters**;  
 }  
 *// if not - just null* **return null**;  
 }  
  
 **public** String getCreatedName() {  
 **if** (**confirmed**) {  
 **return tf\_indexName**.getText();  
 }  
 **return null**;  
 }  
}

* 1. **Класс IndexFoldersController**

**package** view.controllers;  
  
**import** index.Index;  
**import** index.IndexingRequest;  
**import** java.io.File;  
**import** java.net.URL;  
**import** java.nio.file.Path;  
**import** java.util.ResourceBundle;  
**import** javafx.beans.InvalidationListener;  
**import** javafx.beans.binding.Bindings;  
**import** javafx.beans.property.SimpleBooleanProperty;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListView;  
**import** javafx.scene.layout.Background;  
**import** javafx.scene.layout.BackgroundFill;  
**import** javafx.scene.paint.Color;  
**import** javafx.stage.DirectoryChooser;  
**import** javafx.stage.Stage;  
**import** view.views.PathCell;  
  
  
**public class** IndexFoldersController {  
  
 **private** ObservableList<Path> **pathList**;  
 **private** Index **index**;  
 **private volatile** SimpleBooleanProperty **hasFiles** = **new** SimpleBooleanProperty(**false**);  
 **private** MainController **mainController**;  
  
 @FXML  
 **private** ResourceBundle **resources**;  
  
 @FXML  
 **private** URL **location**;  
  
 @FXML  
 **private** Button **btn\_index**;  
  
 @FXML  
 **private** Button **btn\_foldersSelection**;  
  
  
 @FXML  
 **private** ListView<Path> **lv\_filesToIndex**;  
 **private boolean validated**;  
  
  
 @FXML  
 **void** onOpenFileDialog(ActionEvent event) {  
 DirectoryChooser directoryChooser = **new** DirectoryChooser();  
 directoryChooser.setTitle(**"Open Resource File"**);  
 File result = directoryChooser.showDialog(**btn\_foldersSelection**.getScene().getWindow());  
 **if** (result != **null**) {  
 **pathList**.add(result.toPath());  
 }  
 }  
  
 @FXML  
 **void** onIndexFolders(ActionEvent event) {  
 **validated** = **true**;  
 ((Stage) (**btn\_index**.getScene().getWindow())).close();  
 }  
  
 @FXML  
 **void** initialize() {  
 **assert btn\_index** != **null** : **"fx:id=\"btn\_index\" was not injected: check your FXML file 'indexFolders.fxml'."**;  
 **assert btn\_foldersSelection** != **null** : **"fx:id=\"btn\_foldersSelection\" was not injected: check your FXML file 'indexFolders.fxml'."**;  
 **assert lv\_filesToIndex** != **null** : **"fx:id=\"lv\_filesToIndex\" was not injected: check your FXML file 'indexFolders.fxml'."**;  
  
 *// initialize list* **pathList** = FXCollections.*observableArrayList*();  
 **lv\_filesToIndex**.setItems(**pathList**);  
 **lv\_filesToIndex**.setCellFactory(param -> **new** PathCell());  
 **lv\_filesToIndex**.setPlaceholder(**new** Label(**"Click button \"Select files and folders\""**));  
  
 *// setting flag (controlling count files)* **pathList**.addListener((InvalidationListener) c -> {  
 **if** (**pathList**.size() > 0) {  
 **hasFiles**.set(**true**);  
 } **else** {  
 **hasFiles**.set(**false**);  
 }  
 });  
  
 *// set index button to be available only if directories is not empty* **btn\_index**.disableProperty().bind(Bindings.*when*(**hasFiles**.not())  
 .then(**true**)  
 .otherwise(**false**));  
  
 *// set selection button to be colourful only if directories if there is no directories in list* **btn\_foldersSelection**.backgroundProperty().bind(Bindings.*when*(**hasFiles**.not())  
 .then(**new** Background(  
 **new** BackgroundFill(Color.*valueOf*(**"#8CC152"**), **null**,  
 **null**)))  
 .otherwise(**new** Background(  
 **new** BackgroundFill(Color.*valueOf*(**"#656D78"**), **null**,  
 **null**))));  
 *// set index button to be colourful only if directories if there is no directories in list* **btn\_index**.backgroundProperty().bind(Bindings.*when*(**hasFiles**)  
 .then(**new** Background(  
 **new** BackgroundFill(Color.*valueOf*(**"#8CC152"**), **null**, **null**)))  
 .otherwise(**new** Background(  
 **new** BackgroundFill(Color.*valueOf*(**"#656D78"**), **null**, **null**))));  
 }  
  
  
 **public** IndexingRequest toRequest() {  
 **if** (!**validated**) {  
 **return null**;  
 }  
 IndexingRequest.Builder builder = IndexingRequest.*getBuilder*();  
 **for** (Path path : **pathList**) {  
 builder.addPath(path);  
 }  
 builder.setIndex(**index**);  
 **return** builder.build();  
 }  
  
 **public void** setIndex(Index index) {  
 **this**.**index** = index;  
 }  
  
 **public void** setMainController(MainController mainController) {  
 **this**.**mainController** = mainController;  
 }  
}

* 1. **Класс IndexInfoController**

**package** view.controllers;*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 01.04.17 22:19  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***import** index.Index;  
**import** index.IndexingRequest;  
**import** java.io.File;  
**import** java.net.URL;  
**import** java.nio.file.Path;  
**import** java.util.ArrayList;  
**import** java.util.HashSet;  
**import** java.util.List;  
**import** java.util.Set;  
**import** java.util.logging.Logger;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListView;  
**import** javafx.scene.control.ScrollPane;  
**import** javafx.scene.control.SelectionMode;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.layout.VBox;  
**import** javafx.stage.DirectoryChooser;  
**import** javafx.stage.Stage;  
**import** view.views.PathCell;  
  
**public class** IndexInfoController {  
  
 **private final** Logger **log** = Logger.*getLogger*(Index.**class**.getName());  
  
 **private final** ListView<Path> **pathsListView** = **new** ListView<>();  
 **private** ObservableList<Path> **paths**;  
  
 **private** Index **oldIndex**;  
 **private** Index **tempIndex**;  
  
 **private** Index **resultIndex** = **null**;  
 **private** IndexingRequest **resultRequest** = **null**;  
 **private boolean validated** = **false**;  
  
 @FXML  
 **private** ScrollPane **sp\_parameters**;  
  
 @FXML  
 **private** TextField **tf\_name**;  
  
 @FXML  
 **private** ScrollPane **sp\_paths**;  
  
 @FXML  
 **private** Button **btn\_reindex**;  
  
 @FXML  
 **private** Button **btn\_removePath**;  
 **private** MainController **mainStage**;  
  
 @FXML  
 **void** onAddPath() {  
 DirectoryChooser directoryChooser = **new** DirectoryChooser();  
 directoryChooser.setTitle(**"Select directory to oldIndex"**);  
 File folder = directoryChooser.showDialog(**null**);  
 **if** (folder != **null**) {  
 **paths**.add(folder.toPath());  
 }  
 }  
  
 @FXML  
 **void** onRemovePath() {  
 **paths**.remove(**pathsListView**.getSelectionModel().getSelectedItem());  
 }  
  
 @FXML  
 **void** onReindex() {  
 validate();  
 ((Stage) (btn\_reindex.getScene().getWindow())).close();  
 }  
  
 **private void** validate() {  
 *// check name change* String text = tf\_name.getText();  
 **if** (!oldIndex.getName().equals(tf\_name.getText()) && !**""**.equals(text)) {  
 oldIndex.changeName(tf\_name.getText());  
 }  
  
 *// changes in indexing paths* **if** (tempIndex.getParameters().equals(oldIndex.getParameters())) {  
 *// retain new paths* Set<Path> newPaths = **new** HashSet<>(paths);  
 Set<Path> oldPaths = oldIndex.getIndexedPaths();  
  
 *// check if there is any changes in paths* **if** (!oldPaths.containsAll(newPaths) || !newPaths.containsAll(oldPaths)) {  
 *// some changes* Set<Path> temp = **new** HashSet<>(oldPaths);  
 temp.removeAll(newPaths);  
 *// check if new paths contains not all old paths* **if** (temp.size() > 0) {  
 *// smth got deleted* oldIndex.remove(temp); *// remove from index* }  
  
 temp = **new** HashSet<>(newPaths);  
 temp.removeAll(oldPaths);  
 *// check if old paths contains not all new paths* **if** (temp.size() > 0) {  
 *// smth got added* resultRequest = IndexingRequest.getBuilder().addPaths(temp).setIndex(oldIndex).build(); *// prepare request* }  
 }  
 } **else** {  
 *// core changes - needs index recreating* resultIndex = **new** Index(oldIndex.getName(), tempIndex.getParameters());  
 }  
  
 *// clear all border references* tempIndex = **null**;  
 oldIndex = **null**;  
 validated = **true**;  
 }  
  
 @FXML  
 **void** initialize() {  
 paths = FXCollections.observableArrayList();  
  
 pathsListView.setPlaceholder(**new** Label(**"No paths found"**));  
 pathsListView.maxHeightProperty().bind(sp\_paths.heightProperty());  
  
 *// set remove option to be enable only if something is selected* btn\_removePath.disableProperty().bind(pathsListView.getSelectionModel().selectedIndexProperty().isEqualTo(-1));  
 }  
  
 **public** IndexingRequest toRequest() {  
 **if** (!validated) {  
 validate();  
 }  
 **return** resultRequest;  
 }  
  
 **public** Index toIndex() {  
 **if** (!validated) {  
 validate();  
 }  
 **return** resultIndex;  
 }  
  
 **public void** setIndex(Index oldIndex) {  
 **if** (oldIndex == **null**) {  
 **return**;  
 }  
 **this**.oldIndex = oldIndex;  
 **this**.tempIndex = oldIndex.clone(); *// clones parameters, name and etc. add link to storage  
  
 // initialize path list* paths.addAll(tempIndex.getIndexedPaths());  
 pathsListView.setItems(paths);  
 pathsListView.setCellFactory(param -> **new** PathCell());  
 pathsListView.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);  
 sp\_paths.setContent(pathsListView);  
  
 *// initialize param list* VBox parametersInContainer = (VBox) Controllers.getParamList(tempIndex.getParameters());  
 sp\_parameters.widthProperty().addListener((observable, oldValue, newValue) -> {  
 parametersInContainer.prefWidthProperty().setValue(newValue.intValue() - 20);  
 });  
 sp\_parameters.setContent(parametersInContainer);  
  
 tf\_name.setText(oldIndex.getName());  
  
 *// set status to operating* }  
  
 **public void** setMainStage(MainController mainStage) {  
 **this**.mainStage = mainStage;  
 }  
}

* 1. **Класс MainController**

**package** view.controllers;  
  
**import** index.Index;  
**import** index.IndexParameters;  
**import** index.IndexingRequest;  
**import** index.SearchRequest;  
**import** index.Storages.Inclusion;  
**import** java.awt.Desktop;  
**import** java.awt.datatransfer.StringSelection;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.List;  
**import** java.util.Scanner;  
**import** java.util.concurrent.ExecutorService;  
**import** java.util.concurrent.Executors;  
**import** java.util.stream.Collectors;  
**import** java.util.stream.Stream;  
**import** javafx.application.Platform;  
**import** javafx.beans.binding.Bindings;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ListChangeListener;  
**import** javafx.collections.ObservableList;  
**import** javafx.collections.ObservableSet;  
**import** javafx.collections.SetChangeListener;  
**import** javafx.concurrent.Task;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Node;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.Alert;  
**import** javafx.scene.control.Alert.AlertType;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.CheckBox;  
**import** javafx.scene.control.ComboBox;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListCell;  
**import** javafx.scene.control.ListView;  
**import** javafx.scene.control.ProgressBar;  
**import** javafx.scene.control.SelectionMode;  
**import** javafx.scene.control.SelectionModel;  
**import** javafx.scene.control.SplitPane;  
**import** javafx.scene.control.Tab;  
**import** javafx.scene.control.TextArea;  
**import** javafx.scene.control.cell.TextFieldListCell;  
**import** javafx.scene.input.Clipboard;  
**import** javafx.scene.input.ClipboardContent;  
**import** javafx.scene.input.KeyCode;  
**import** javafx.scene.input.MouseEvent;  
**import** javafx.scene.layout.HBox;  
**import** javafx.scene.layout.Priority;  
**import** javafx.scene.text.Text;  
**import** javafx.scene.text.TextFlow;  
**import** javafx.stage.Modality;  
**import** javafx.stage.Stage;  
**import** javafx.stage.StageStyle;  
**import** javafx.util.StringConverter;  
**import** view.views.InclusionCell;  
**import** view.views.IndexCell;  
  
  
**public class** MainController {  
  
 **public final static** String ***INDICES\_DIRECTORY*** = **"indices\\"**;  
 **public final static** String ***INDEX\_FOLDERS\_FXML*** = **"/view/fxml/indexFolders.fxml"**;  
 **public final static** String ***INDEX\_INFO\_FXML*** = **"/view/fxml/indexInfo.fxml"**;  
 **public final static** String ***ERROR\_CLASS*** = **"error"**;  
 **public final static int *MAX\_HISTORY*** = 8;  
 **private** ObservableList<Inclusion> **paths**;  
 **private** ObservableList<Index> **indices**;  
 **private** ObservableList<SearchRequest> **history**;  
 **private final** ExecutorService **requests** = Executors.*newCachedThreadPool*();  
 **private final** ObservableList<Task<Long>> **taskQueue** = FXCollections.*observableArrayList*();  
 **private** Task **currentTask** = **null**;  
  
 **private** Node **stashedPane**;  
  
 @FXML  
 **private** Button **btn\_cancel**;  
  
 @FXML  
 **private** ComboBox<SearchRequest> **cb\_search**;  
  
 @FXML  
 **private** Button **btn\_search**;  
  
 @FXML  
 **private** SplitPane **sp\_main**;  
  
 @FXML  
 **private** Button **btn\_createIndex**;  
  
 @FXML  
 **private** Button **btn\_showIndex**;  
  
 @FXML  
 **private** Label **lb\_status**;  
  
 @FXML  
 **private** CheckBox **cb\_seachSubstring**;  
  
 @FXML  
 **private** ListView<Inclusion> **lv\_files**;  
  
 @FXML  
 **private** ListView<Index> **lv\_indices**;  
  
 @FXML  
 **private** ProgressBar **pb\_progress**;  
  
 @FXML  
 **private** Tab **tab\_indexes**;  
  
 @FXML  
 **private** Tab **tab\_search**;  
  
 @FXML  
 **private** TextFlow **tf\_preview**;  
  
 @FXML  
 **void** onSubstringSearchChanged(ActionEvent event) {  
  
 }  
  
 @FXML  
 **void** onCreateIndex(ActionEvent event) {  
 *// initialize stage* Stage indexCreationStage = **new** Stage();  
 indexCreationStage.setTitle(**"Create index"**);  
  
 *// initialize controller* IndexCreationController icc = **new** IndexCreationController(**indices**);  
 javafx.scene.Node view = icc.getView();  
 indexCreationStage.setScene(**new** Scene((Parent) view));  
  
 *// show as a modal* indexCreationStage.initOwner(**lv\_files**.getScene().getWindow());  
 indexCreationStage.initModality(Modality.***APPLICATION\_MODAL***);  
 indexCreationStage.showAndWait();  
  
 *// result handling - adding index to indices* IndexParameters parameters = icc.getCreatedParameters();  
 String name = icc.getCreatedName();  
 **if** (parameters == **null** || name == **null**) {  
 **return**;  
 }  
 Index index = **new** Index(name, parameters);  
 **indices**.add(index);  
  
 openIndexFilesForm(index);  
 }  
  
 @FXML  
 **void** onSearch() {  
 *// get selected index to search in* Index selectedIndex = **lv\_indices**.getSelectionModel().getSelectedItem();  
 *// if there is no index - show dialog and return* **lv\_files**.setPlaceholder(**new** Label(**"No results yet."**));  
 **if** (selectedIndex == **null**) {  
 Alert alert = **new** Alert(AlertType.***INFORMATION***);  
 alert.setTitle(**"No index selected"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Select or create-select an index to search in on indices tab."**);  
 alert.showAndWait();  
 **return**;  
 }  
 String searchFor = **cb\_search**.getEditor().textProperty().get();  
 **if** (searchFor == **null** || **""**.equals(searchFor)) {  
 **return**;  
 }  
  
 SearchRequest request = **cb\_search**.getConverter().fromString(**cb\_search**.getEditor().getText());  
  
 *// todo: обновление списка найденных путей здесь* **paths**.clear();  
  
 *// execute request* **if** (request != **null**) {  
 addTask(request);  
  
 **if** (!**history**.contains(request)) {  
 **history**.add(request);  
 }  
 **cb\_search**.getSelectionModel().select(request);  
  
 } **else** {  
 Alert alert = **new** Alert(AlertType.***ERROR***);  
 alert.setTitle(**"Search failed"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Failed to create search request."**);  
 alert.showAndWait();  
 **return**;  
 }  
  
 ObservableSet<Inclusion> requestResults = request.getResult();  
 requestResults.addListener((SetChangeListener<? **super** Inclusion>) change -> {  
 **if** (change.wasAdded()) {  
 Platform.*runLater*(() -> {  
 **paths**.add(change.getElementAdded());  
 });  
*// Platform.runLater(() -> paths.addAll(change.getElementAdded().getPath()));* }  
 });  
 }  
  
 @FXML  
 **void** onShowIndexInfo(ActionEvent event) {  
 **if** (**lv\_indices**.getSelectionModel().getSelectedItem() == **null**) {  
 Alert alert = **new** Alert(AlertType.***INFORMATION***);  
 alert.setTitle(**""**);  
 alert.setHeaderText(**"Index not selected"**);  
 alert.setContentText(**"Please, select index in list to show info."**);  
 alert.showAndWait();  
 **return**;  
 }  
  
 **try** {  
 *// load parent node* FXMLLoader loader = **new** FXMLLoader(getClass().getResource(***INDEX\_INFO\_FXML***));  
 Parent indexInfoNode = loader.load();  
 *// prepare stage* Stage indexInfoStage = **new** Stage();  
 indexInfoStage.setTitle(**"Index info"**);  
 indexInfoStage.setScene(**new** Scene(indexInfoNode));  
 *// set up controller* IndexInfoController controller = loader.getController();  
 Index selectedIndex = lv\_indices.getSelectionModel().getSelectedItem();  
 controller.setIndex(selectedIndex);  
 controller.setMainStage(**this**);  
 indexInfoStage.showAndWait();  
  
 IndexingRequest request = controller.toRequest();  
 Index index = controller.toIndex();  
 *// check for core changes* **if** (index != **null**) {  
 *// if core changes made - remake an index* indices.remove(index);  
 indices.add(index);  
 }  
 *// check for checnges in pahts* **if** (request != **null**) {  
 addTask(request);  
 }  
 lv\_indices.refresh();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **void** onCancelTask(ActionEvent event) {  
 **if** (taskQueue.size() == 0) {  
 **return**;  
 }  
 taskQueue.remove(currentTask);  
*// currentTask.cancel();* }  
  
 **private void** openIndexFilesForm(Index index) {  
 *// show indexing window* **try** {  
 FXMLLoader fxmlLoader = **new** FXMLLoader(getClass().getResource(INDEX\_FOLDERS\_FXML));  
 Parent indexFolders = fxmlLoader.load();  
  
 IndexFoldersController controller = fxmlLoader.getController();  
 controller.setIndex(index);  
 controller.setMainController(**this**);  
  
 Stage stage = **new** Stage();  
 stage.initModality(Modality.APPLICATION\_MODAL);  
 stage.initStyle(StageStyle.UTILITY);  
 stage.setTitle(**"Index folders"**);  
 stage.setScene(**new** Scene(indexFolders));  
 stage.showAndWait();  
  
 *// add request to queue* IndexingRequest request = controller.toRequest();  
 **if** (request != **null**) {  
 addTask(request);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** addTask(Task<Long> task) {  
 requests.submit(task);  
 taskQueue.add(task);  
 }  
  
 **public void** saveIndices() {  
 *// check for index existing* **if** (indices.size() == 0) {  
 **return**;  
 }  
  
 File indicesDirectory = **new** File(INDICES\_DIRECTORY);  
 *// check for directory existence, if not - create* **if** (!indicesDirectory.exists()) {  
 indicesDirectory.mkdir();  
 }  
  
 *// start thread for saving indices (cause it is long operation)* **new** Thread(() -> {  
 indices.forEach(index -> {  
 requests.shutdown();  
 index.save(INDICES\_DIRECTORY);  
 index.exit();  
 });  
 }).start();  
 }  
  
 **public void** loadIndices() {  
 File indicesDirectory = **new** File(INDICES\_DIRECTORY);  
 *// check for directory existence, if not - leave method (cause it's no indices dude)* **if** (!indicesDirectory.exists()) {  
 **return**;  
 }  
 *// start thread for loading indices (cause it is long operation)* **new** Thread(() -> {  
 **try** (Stream<Path> paths = Files.walk(indicesDirectory.toPath().toAbsolutePath())) {  
 paths.forEach(filePath -> {  
 *// check if name contain '.ser' - serialization extension* **if** (filePath.getFileName().toString().contains(**".ser"**)) {  
 Index temp = Index.load(INDICES\_DIRECTORY + filePath.getFileName().toString());  
 Platform.runLater(() -> {  
 indices.add(temp);  
 });  
 }  
 });  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }).start();  
 }  
  
 **public void** setPreview(**boolean** active) {  
 **if** (active) {  
 **if** (stashedPane == **null**) {  
 **return**;  
 }  
 sp\_main.getItems().add(1, stashedPane);  
 sp\_main.setDividerPosition(1, 0.8);  
 stashedPane.setVisible(active);  
 stashedPane = **null**;  
 } **else** {  
 stashedPane = sp\_main.getItems().get(1);  
 sp\_main.getItems().remove(stashedPane);  
 stashedPane.setVisible(active);  
 }  
 }  
  
 @FXML  
 **void** initialize() {  
 *// initialize items* initializePathList();  
 initializeIndexList();  
 initializeTaskQueue();  
 initializeHistory();  
  
 *// connect properties* initializeSmallBindings();  
  
 *// set hidable pane* stashedPane = sp\_main.getItems().get(1);  
 bindPreview();  
 setPreview(**false**);  
 }  
  
 **private void** initializeSmallBindings() {  
 btn\_showIndex.disableProperty().bind(lv\_indices.getSelectionModel().selectedIndexProperty().isEqualTo(-1));  
 btn\_search.disableProperty().bind(lv\_indices.getSelectionModel().selectedIndexProperty().isEqualTo(-1)  
 .or(cb\_search.getEditor().textProperty().isEqualTo(**""**)));  
 cb\_search.getEditor().textProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null** || **""**.equals(newValue)) {  
 cb\_search.getStyleClass().add(ERROR\_CLASS);  
 } **else** {  
 cb\_search.getStyleClass().remove(ERROR\_CLASS);  
 }  
 });  
 cb\_search.setOnKeyPressed(event -> {  
 **if** (event.getCode() != KeyCode.ENTER) {  
 **return**;  
 }  
 onSearch();  
 });  
 }  
  
 **private void** bindPreview() {  
 lv\_files.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 setPreview(**false**);  
 } **else** {  
 setPreview(**true**);  
 displayFile(newValue);  
 }  
 });  
 }  
  
 **private void** displayFile(Inclusion inclusion) {  
 tf\_preview.getChildren().clear();  
 tf\_preview.getChildren().add(**new** Text(**"Loading ..."**));  
 **try** (Stream<String> stream = Files.lines(inclusion.getPath())) {  
 **if** (!inclusion.getPath().toFile().exists()) {  
 tf\_preview.getChildren().clear();  
 tf\_preview.getChildren().add(**new** Text(**"Not exists anymore ..."**));  
 **return**;  
 }  
 Stream<String> temp = stream;  
 **long** lowbound = inclusion.getPlace() - 3;  
 **if** (lowbound >= 0) {  
 temp = temp.skip(lowbound);  
 }  
 List<String> list = temp.limit(5)  
 .map(s -> {  
 List<String> words = **new** ArrayList<>(Arrays.asList(s.split(**" "**)));  
 words.add(**"\n"**);  
 **return** words.toArray(**new** String[words.size()]);  
 })  
 .flatMap(Arrays::stream)  
 .collect(Collectors.toList());  
 Text before = **null**,  
 actual = **null**,  
 after;  
 StringBuilder builder = **new** StringBuilder();  
 **for** (String string : list) {  
 **if** (!string.contains(inclusion.getWord())) {  
 builder.append(string).append(**" "**);  
 } **else** {  
 before = **new** Text(builder.toString());  
 actual = **new** Text(string + **" "**);  
 actual.setStyle(**"-fx-fill: red"**);  
 builder = **new** StringBuilder();  
 }  
 }  
 after = **new** Text(builder.toString());  
 tf\_preview.getChildren().clear();  
 tf\_preview.getChildren().addAll(before, actual, after);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** initializePathList() {  
*// lv\_files.setCellFactory(param -> new PathCell());* lv\_files.setCellFactory(param -> {  
 ListCell cell = **new** InclusionCell();  
 SelectionModel selectionModel = lv\_files.getSelectionModel();  
 cell.addEventFilter(MouseEvent.MOUSE\_PRESSED, event -> {  
 lv\_files.requestFocus();  
 **if** (!cell.isEmpty() && selectionModel.getSelectedIndex() != cell.getIndex()) {  
 selectionModel.select(cell.getIndex());  
 } **else** {  
 lv\_files.getSelectionModel().clearSelection();  
 }  
 event.consume();  
 });  
 **return** cell;  
 });  
 paths = FXCollections.observableArrayList();  
 lv\_files.setItems(paths);  
 lv\_files.setPlaceholder(**new** Label(**"Select index to search in and word for search."**));  
 lv\_files.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);  
 }  
  
 **private void** initializeIndexList() {  
 lv\_indices.setCellFactory(param -> {  
 ListCell cell = **new** IndexCell();  
 SelectionModel selectionModel = lv\_indices.getSelectionModel();  
 cell.addEventFilter(MouseEvent.MOUSE\_PRESSED, event -> {  
 lv\_indices.requestFocus();  
 **if** (!cell.isEmpty() && selectionModel.getSelectedIndex() != cell.getIndex()) {  
 selectionModel.select(cell.getIndex());  
 } **else** {  
 lv\_indices.getSelectionModel().clearSelection();  
 }  
 event.consume();  
 });  
 **return** cell;  
 });  
 indices = FXCollections.observableArrayList();  
 lv\_indices.setItems(indices);  
 lv\_indices.setPlaceholder(**new** Label(**"No indices found."**));  
 lv\_indices.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);  
 }  
  
 **private void** initializeTaskQueue() {  
 pb\_progress.visibleProperty().bind(Bindings.size(taskQueue).greaterThan(0));  
 lb\_status.visibleProperty().bind(Bindings.size(taskQueue).greaterThan(0));  
 btn\_cancel.visibleProperty().bind(Bindings.size(taskQueue).greaterThan(0));  
 taskQueue.addListener((ListChangeListener<? **super** Task<Long>>) c -> {  
 c.next();  
 **if** (c.wasAdded() && c.getAddedSize() > 0) {  
 currentTask = c.getAddedSubList().get(0);  
 } **else if** (c.wasRemoved() && taskQueue.size() > 0) {  
 currentTask = taskQueue.get(0);  
 } **else** {  
 **if** (c.wasRemoved() && taskQueue.size() == 0) {  
 currentTask = **null**;  
 }  
 **return**;  
 }  
  
 pb\_progress.progressProperty().bind(currentTask.progressProperty());  
 lb\_status.textProperty().bind(currentTask.messageProperty());  
  
 **final** Task temp = currentTask;  
 currentTask.setOnSucceeded(event -> {  
 taskQueue.remove(temp);  
 });  
 currentTask.setOnCancelled(event -> {  
 taskQueue.remove(temp);  
 });  
  
 });  
 }  
  
 **private void** initializeHistory() {  
 lv\_indices.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 cb\_search.setEditable(**false**);  
 cb\_search.setPromptText(**"Select index on index tab"**);  
 } **else** {  
 cb\_search.setEditable(**true**);  
 cb\_search.setPromptText(**"Input text"**);  
 cb\_search.getEditor().setText(**""**);  
 }  
 });  
 cb\_search.setEditable(**false**);  
 cb\_search.setPromptText(**"Select index on index tab"**);  
 cb\_search.setPlaceholder(**new** Label(**"No history"**));  
  
 cb\_search.setCellFactory(param -> **new** TextFieldListCell<SearchRequest>() {  
 @Override  
 **public void** updateItem(SearchRequest item, **boolean** empty) {  
 **super**.updateItem(item, empty);  
 **if** (item == **null** || empty) {  
 setGraphic(**null**);  
 } **else** {  
 HBox container = **new** HBox();  
  
 Label temp\_searchFor = **new** Label(**"word: "**);  
 temp\_searchFor.setTextFill(javafx.scene.paint.Color.GRAY);  
 temp\_searchFor.setOpacity(0.8);  
  
 Label searchFor = **new** Label(item.getSearchFor());  
 HBox.setHgrow(searchFor, Priority.ALWAYS);  
 searchFor.setMaxWidth(Double.MAX\_VALUE);  
  
 Label temp\_from = **new** Label(**"index: "**);  
 temp\_from.setTextFill(javafx.scene.paint.Color.GRAY);  
 temp\_from.setOpacity(0.8);  
  
 Label from = **new** Label(item.getIndex().getName());  
 HBox.setHgrow(from, Priority.ALWAYS);  
 from.setMaxWidth(Double.MAX\_VALUE);  
 from.setTextFill(javafx.scene.paint.Color.BLUE);  
  
 container.getChildren().addAll(temp\_searchFor, searchFor, temp\_from, from);  
 setGraphic(container);  
 setText(**null**);  
 }  
 }  
 });  
 cb\_search.setConverter(**new** StringConverter<SearchRequest>() {  
 @Override  
 **public** String toString(SearchRequest object) {  
 **if** (object == **null**) {  
 **return null**;  
 }  
 **return** object.toString();  
 }  
  
 @Override  
 **public** SearchRequest fromString(String string) {  
 **if** (string == **null** || **""**.equals(string)) {  
 **return null**;  
 }  
  
 Index index = lv\_indices.getSelectionModel().getSelectedItem();  
 **if** (index == **null**) {  
 **return null**;  
 }  
  
 SearchRequest request = SearchRequest.getBuilder().setIndex(index)  
 .setSearchFor(string.toLowerCase())  
 .setSubstringSearch(cb\_seachSubstring.isSelected()).build();  
 **if** (request == **null**) {  
 **return null**;  
 }  
 **return** request;  
 }  
 });  
  
 history = FXCollections.observableArrayList();  
 history.addListener((ListChangeListener<? **super** SearchRequest>) c -> {  
 c.next();  
 **if** (c.wasAdded()) {  
 **if** (history.size() >= MAX\_HISTORY) {  
 history.remove(0);  
 }  
 }  
 });  
 cb\_search.setItems(history);  
 cb\_search.valueProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 **return**;  
 }  
 paths.clear();  
  
 paths.setAll(newValue.getResult());  
 });  
 }  
  
 @FXML  
 **void** onOpenFileDir(ActionEvent event) {  
 **if** (lv\_files.getSelectionModel().getSelectedItem() == **null**) {  
 Alert alert = **new** Alert(AlertType.INFORMATION);  
 alert.setTitle(**null**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Select path to open."**);  
 alert.showAndWait();  
 **return**;  
 }  
 **if** (Desktop.isDesktopSupported()) {  
 **try** {  
 Desktop.getDesktop().open(lv\_files.getSelectionModel().getSelectedItem().getPath().getParent().toFile());  
 } **catch** (Exception e) {  
 Alert alert = **new** Alert(AlertType.ERROR);  
 alert.setTitle(**null**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Path can not be opened."**);  
 alert.showAndWait();  
 e.printStackTrace();  
 }  
 }  
 }  
  
 @FXML  
 **void** onCopyToBuffer(ActionEvent event) {  
 **if** (lv\_files.getSelectionModel().getSelectedItem() == **null**) {  
 Alert alert = **new** Alert(AlertType.INFORMATION);  
 alert.setTitle(**null**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Select path to open."**);  
 alert.showAndWait();  
 **return**;  
 }  
 Clipboard clipboard = Clipboard.getSystemClipboard();  
 ClipboardContent content = **new** ClipboardContent();  
 content.putString(lv\_files.getSelectionModel().getSelectedItem().getPath().toString());  
 clipboard.setContent(content);  
 }  
  
 @FXML  
 **void** onOpenFile(ActionEvent event) {  
 **if** (lv\_files.getSelectionModel().getSelectedItem() == **null**) {  
 Alert alert = **new** Alert(AlertType.INFORMATION);  
 alert.setTitle(**null**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Select path to open."**);  
 alert.showAndWait();  
 **return**;  
 }  
 **if** (Desktop.isDesktopSupported()) {  
 **try** {  
 Desktop.getDesktop().open(lv\_files.getSelectionModel().getSelectedItem().getPath().toFile());  
 } **catch** (Exception e) {  
 Alert alert = **new** Alert(AlertType.ERROR);  
 alert.setTitle(**null**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(**"Path can not be opened."**);  
 alert.showAndWait();  
 e.printStackTrace();  
 }  
 }  
 }  
}

* 1. **Класс PathCellController**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 19.02.17 17:02  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/  
  
/\*  
 \* Created by: Aleksandr  
 \* Date: 15.01.2017  
 \* Project: FileFinder  
 \* <p>  
 \* "The more we do, the more we can do" ©  
 \*/***package** view.controllers;  
  
  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** javafx.application.Platform;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.image.ImageView;  
  
  
*/\*\*  
 \* Controller for PathCell. It includes loading the cell, item, and also updating its elements on  
 \* the view.  
 \*/***public class** PathCellController {  
  
 */\*\*  
 \* Path to fxml view file.  
 \*/* **private final** String **cellFileName** = **"/view/fxml/pathCell.fxml"**;  
 */\*\*  
 \* Thumbnail for file.  
 \*/* @FXML  
 **private** ImageView **iv\_thumbnail**;  
 */\*\*  
 \* Label for "created at" parameter.  
 \*/* @FXML  
 **private** Label **lb\_created**;  
 */\*\*  
 \* Label for "name" parameter.  
 \*/* @FXML  
 **private** Label **lb\_name**;  
 */\*\*  
 \* Label for "file size" parameter.  
 \*/* @FXML  
 **private** Label **lb\_size**;  
 */\*\*  
 \* View for the cell. Empty if was not called {****@link*** *PathCellController#getView()} yet.  
 \* Otherwise, if loading is complete without errors - here will lay loaded view.  
 \*/* **private** Node **view**;  
 */\*\*  
 \* The main item - Path, which cell is representing.  
 \*/* **private** Path **item**;  
  
 */\*\*  
 \* Indicates if cell is selected.  
 \*/* **private boolean selected**;  
 */\*\*  
 \* Indicates if cell is hovered.  
 \*/* **private boolean hovered**;  
  
 */\*\*  
 \* Initializes cell, without item in it. Also doesn't load view.  
 \*/* **public** PathCellController() {  
 **selected** = **false**;  
 **hovered** = **false**;  
 }  
  
 */\*\*  
 \* Initializes cell with item in it. Also doesn't load the view.  
 \*  
 \** ***@param item*** *Item - Path, the cell to hold in.  
 \*/* **public** PathCellController(Path item) {  
 **this**();  
 **this**.**item** = item;  
 }  
  
 */\*\*  
 \* Asserts, that all @JAVAFX values are loaded.  
 \*/* @FXML  
 **void** initialize() {  
 **assert iv\_thumbnail** != **null** : **"fx:id=\"iv\_thumbnail\" was not injected: check your FXML file 'pathCell.fxml'."**;  
 **assert lb\_created** != **null** : **"fx:id=\"lb\_created\" was not injected: check your FXML file 'pathCell.fxml'."**;  
 **assert  
 lb\_name** != **null** : **"fx:id=\"lb\_name\" was not injected: check your FXML file 'pathCell.fxml'."**;  
 **assert  
 lb\_size** != **null** : **"fx:id=\"lb\_size\" was not injected: check your FXML file 'pathCell.fxml'."**;  
 }  
  
 */\*\*  
 \* Loads the view if it is not loaded, otherwise - returns already loaded value.  
 \* If error on load occurred - returns null.  
 \*/* **public** Node getView() {  
 **if** (**view** == **null**) {  
 **try** {  
 FXMLLoader loader = **new** FXMLLoader(**this**.getClass().getResource(**cellFileName**));  
 loader.setController(**this**);  
 **view** = loader.load();  
 Platform.*runLater*(**this**::update);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return view**;  
 }  
  
 */\*\*  
 \* Sets the item to cell and updates the view.  
 \*  
 \** ***@param path*** *Cell item to hold.  
 \** ***@return*** *this.  
 \*/* **public synchronized** PathCellController setItem(Path path) {  
 **this**.**item** = path;  
 update();  
 **return this**;  
 }  
  
 */\*\*  
 \* Setting is cell selected or not.  
 \*  
 \** ***@param selected*** *Cell selection status.  
 \** ***@return*** *this.  
 \*/* **public synchronized** PathCellController setSelected(**boolean** selected) {  
 **this**.**selected** = selected;  
 **return this**;  
 }  
  
 */\*\*  
 \* Setting is cell hovered or not.  
 \*  
 \** ***@param hovered*** *Cell hovering status.  
 \** ***@return*** *this.  
 \*/* **public synchronized** PathCellController setHovered(**boolean** hovered) {  
 **this**.**hovered** = hovered;  
 **return this**;  
 }  
  
 */\*\*  
 \* Updates the view. If view was not initialized, or item was not set just skips the operations.  
 \*/* **public void** update() {  
 **if** (**item** == **null** || **view** == **null**) {  
 **return**;  
 }  
 setName();  
 setCreated();  
 setSize();  
 }  
  
 */\*\*  
 \* Updates {****@link*** *PathCellController#lb\_name} according to item path.  
 \* If cell is hovered or selected - will show the full path, otherwise - just filepath.  
 \*/* **private void** setName() {  
 String name = **null**;  
 **if** (**selected** || **hovered**) {  
 name = **item**.normalize().toString();  
 } **else** {  
 name = **item**.normalize().toString();  
 }  
 **lb\_name**.setText(name);  
 }  
  
 */\*\*  
 \* Updates {****@link*** *PathCellController#lb\_created} according to item creating date.  
 \*/* **private void** setCreated() {  
 *// todo: make created at field* **return**;  
 }  
  
 */\*\*  
 \* Updates {****@link*** *PathCellController#lb\_size} according to file/folder size.  
 \*/* **private void** setSize() {  
 **try** {  
 **lb\_size**.setText(String.*valueOf*(Files.*size*(**item**)));  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

* 1. **Класс InclusionCell**

**package** view.views;  
  
**import** index.Storages.Inclusion;  
**import** java.io.IOException;  
**import** java.nio.file.Path;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListCell;  
**import** javafx.scene.layout.BorderPane;  
  
*/\*\*  
 \* Created by Александр on 15.04.2017.  
 \*/***public class** InclusionCell **extends** ListCell<Inclusion> {  
  
 FXMLLoader **loader**;  
 **public static final** String ***FXML\_INCLUSION\_CELL*** = **"/view/fxml/inclusionCell.fxml"**;  
 **private boolean hovered** = **false**;  
 **private boolean selected** = **false**;  
  
 @FXML  
 **private** BorderPane **bp\_main**;  
  
 @FXML  
 **private** Label **lbl\_date**;  
  
 @FXML  
 **private** Label **lbl\_place**;  
  
 @FXML  
 **private** Label **lbl\_name**;  
  
 **public** InclusionCell() {  
 selectedProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 **return**;  
 }  
 **selected** = newValue;  
 refresh();  
 });  
 hoverProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 **return**;  
 }  
 **hovered** = newValue;  
 refresh();  
 });  
  
 }  
  
 **private void** refresh() {  
 **if** (getItem() == **null**) {  
 **return**;  
 }  
 **if** (**hovered** || **selected**) {  
 **lbl\_name**.setText(getItem().getPath().toString());  
 } **else** {  
 Path filename = getItem().getPath().getFileName();  
 String name = **null**;  
 **if** (filename == **null**) {  
 name = getItem().getPath().getParent().getFileName().toString();  
 } **else** {  
 name = filename.toString();  
 }  
 **lbl\_name**.setText(name);  
 }  
 }  
  
 @Override  
 **protected void** updateItem(Inclusion item, **boolean** empty) {  
 **super**.updateItem(item, empty);  
 **if** (item == **null** || empty) {  
 setGraphic(**null**);  
 } **else** {  
 **if** (**loader** == **null**) {  
 **loader** = **new** FXMLLoader(getClass().getResource(***FXML\_INCLUSION\_CELL***));  
 **loader**.setController(**this**);  
  
 **try** {  
 **loader**.load();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 setItem(item);  
 refresh();  
 **lbl\_date**.setText(item.getUpdated().toString());  
 **switch** ((**int**) item.getPlace()) {  
 **case** -2:  
 **lbl\_place**.setText(**"extension"**);  
 **break**;  
 **case** -1:  
 **lbl\_place**.setText(**"filename"**);  
 **break**;  
 **default**:  
 **lbl\_place**.setText(**"line "** + item.getPlace());  
 }  
 setGraphic(bp\_main);  
 }  
 }  
}

* 1. **Класс IndexCell**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 17.03.17 13:35  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/***package** view.views;  
  
**import** index.Index;  
**import** javafx.application.Platform;  
**import** javafx.scene.control.ListCell;  
**import** view.controllers.IndexCellController;  
  
*/\*\*  
 \* Created by: Aleksandr  
 \* Date: 17.03.2017  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do" ©  
 \*/***public class** IndexCell **extends** ListCell<Index> {  
  
 **private** IndexCellController **controller**;  
  
 **public** IndexCell() {  
  
 }  
  
 @Override  
 **protected void** updateItem(Index item, **boolean** empty) {  
 **super**.updateItem(item, empty);  
  
 **if** (empty || item == **null**) {  
 setGraphic(**null**);  
 } **else** {  
 **controller** = **new** IndexCellController();  
 **controller**.setIndex(item);  
  
 setGraphic(**controller**.getGraphic());  
  
  
 }  
 }  
  
 @Override  
 **public void** updateSelected(**boolean** selected) {  
 **super**.updateSelected(selected);  
 **if** (**controller** != **null**) {  
 **controller**.setSelected(selected);  
 }  
 }  
}

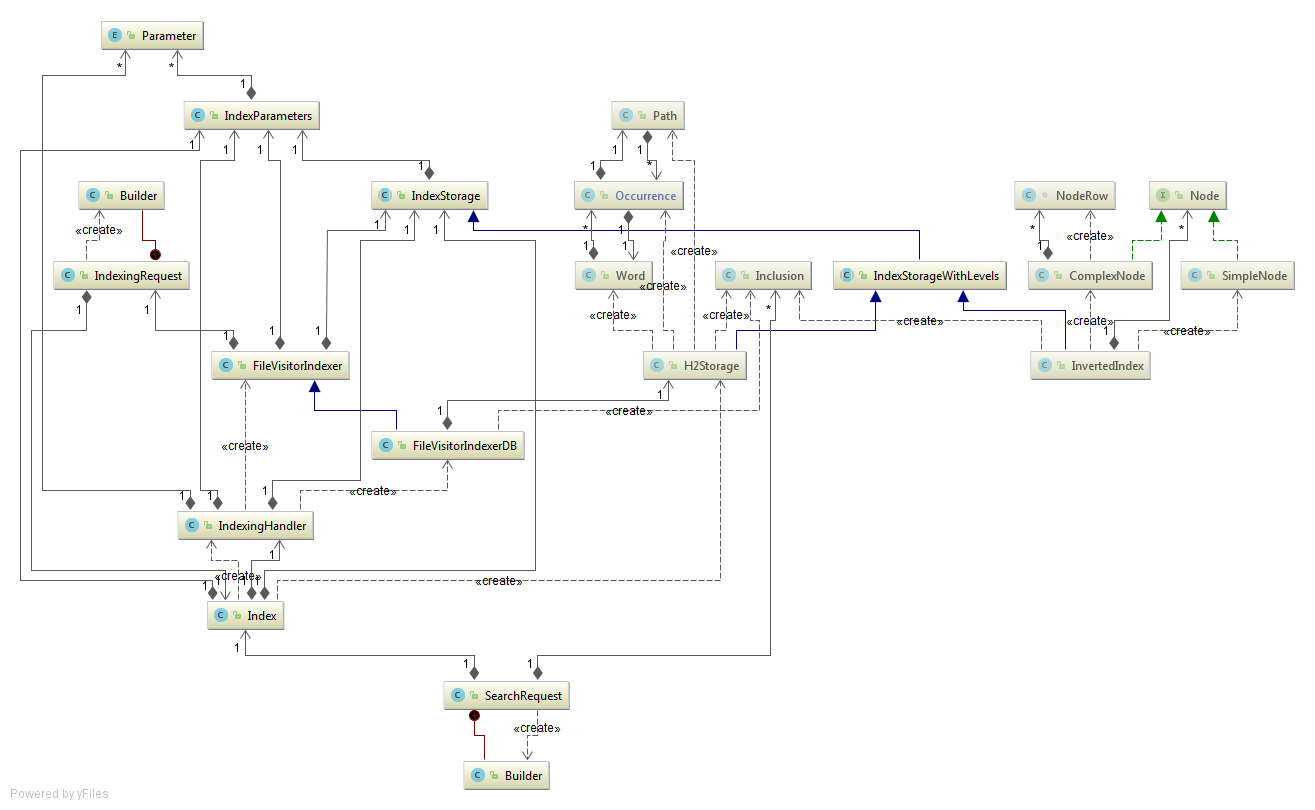
* 1. **Класс PathCell**

*/\*  
 \* Created by Aleksandr Smilyanskiy  
 \* Date: 19.02.17 17:06  
 \* Project: FileFinder  
 \*  
 \* "The more we do, the more we can do"  
 \* Copyright (c) 2017.  
 \*/  
  
/\*  
 \* Created by: Aleksandr  
 \* Date: 15.01.2017  
 \* Project: FileFinder  
 \* <p>  
 \* "The more we do, the more we can do" ©  
 \*/***package** view.views;  
  
**import** java.io.IOException;  
**import** java.nio.file.Path;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.ListCell;  
**import** javafx.scene.layout.BorderPane;  
  
*/\*\*  
 \* Cell for representing needed path fields.  
 \*/***public class** PathCell **extends** ListCell<Path> {  
  
 **public static final** String ***FXML\_PATH\_CELL*** = **"/view/fxml/pathCell.fxml"**;  
 FXMLLoader **loader**;  
 **private boolean hovered** = **false**;  
 **private boolean selected** = **false**;  
  
 *// @FXML  
// private Label lbl\_size;* @FXML  
 **private** Label **lbl\_filePath**;  
 @FXML  
 **private** BorderPane **bp\_main**;  
  
 **public** PathCell() {  
 selectedProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 **return**;  
 }  
 **selected** = newValue;  
 refresh();  
 });  
 hoverProperty().addListener((observable, oldValue, newValue) -> {  
 **if** (newValue == **null**) {  
 **return**;  
 }  
 **hovered** = newValue;  
 refresh();  
 });  
 }  
  
 **private void** refresh() {  
 **if** (getItem() == **null**) {  
 **return**;  
 }  
 **if** (**hovered** || **selected**) {  
 **lbl\_filePath**.setText(getItem().toString());  
 } **else** {  
 Path filename = getItem().getFileName();  
 String name = **null**;  
 **if** (filename == **null**) {  
 name = getItem().getFileName().getParent().getFileName().toString();  
 } **else** {  
 name = filename.toString();  
 }  
 **lbl\_filePath**.setText(name);  
 }  
 }  
  
 @Override  
 **protected void** updateItem(Path item, **boolean** empty) {  
 **super**.updateItem(item, empty);  
 **if** (item == **null** || empty) {  
 setGraphic(**null**);  
 } **else** {  
 **if** (**loader** == **null**) {  
 **loader** = **new** FXMLLoader(getClass().getResource(***FXML\_PATH\_CELL***));  
 **loader**.setController(**this**);  
 **try** {  
 **loader**.load();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 setItem(item);  
 refresh();  
 setGraphic(**bp\_main**);  
 }  
 }  
}

* 1. **Класс Main**

**package** view;  
  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.util.logging.LogManager;  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
**import** view.controllers.MainController;  
  
**public class** Main **extends** Application {  
  
 **private static** String *MAIN\_FXML* = **"/view/fxml/main.fxml"**;  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
  
 **private void** configLogger() {  
*// Logger log = Logger.getLogger(getClass().getName());  
// log.setLevel(Level.ALL);  
// ConsoleHandler handler = new ConsoleHandler();  
// handler.setFormatter(new SimpleFormatter());  
// handler.setLevel(Level.ALL);  
// log.addHandler(handler);* **try** {  
 LogManager.*getLogManager*().readConfiguration(getClass().getResourceAsStream(**"/META-INF/logging.properties"**));  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(*MAIN\_FXML*));  
 Parent root = loader.load();  
 Scene scene = **new** Scene(root);  
 primaryStage.setHeight(450);  
 primaryStage.setWidth(600);  
 primaryStage.setMinHeight(250);  
 primaryStage.setMinWidth(430);  
  
 MainController controller = loader.getController();  
 primaryStage.setOnShown(event -> {  
 controller.loadIndices();  
 });  
 primaryStage.setOnCloseRequest(event -> {  
 controller.saveIndices();  
 });  
  
 configLogger();  
  
 primaryStage.setTitle(**"Search files"**);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

# **Приложение 1. UML – диаграмма классов**

****

*Рис. 1.* Uml – диаграмма всех классов проекта

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ** | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в документе) | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| Изм. | Измененных | Замененных | Новых | Анулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |