МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Лабораторная работа №4

Бекапы

Выполнил студент группы М3201:

Дымчикова Аюна

САНКТ-ПЕТЕРБУРГ

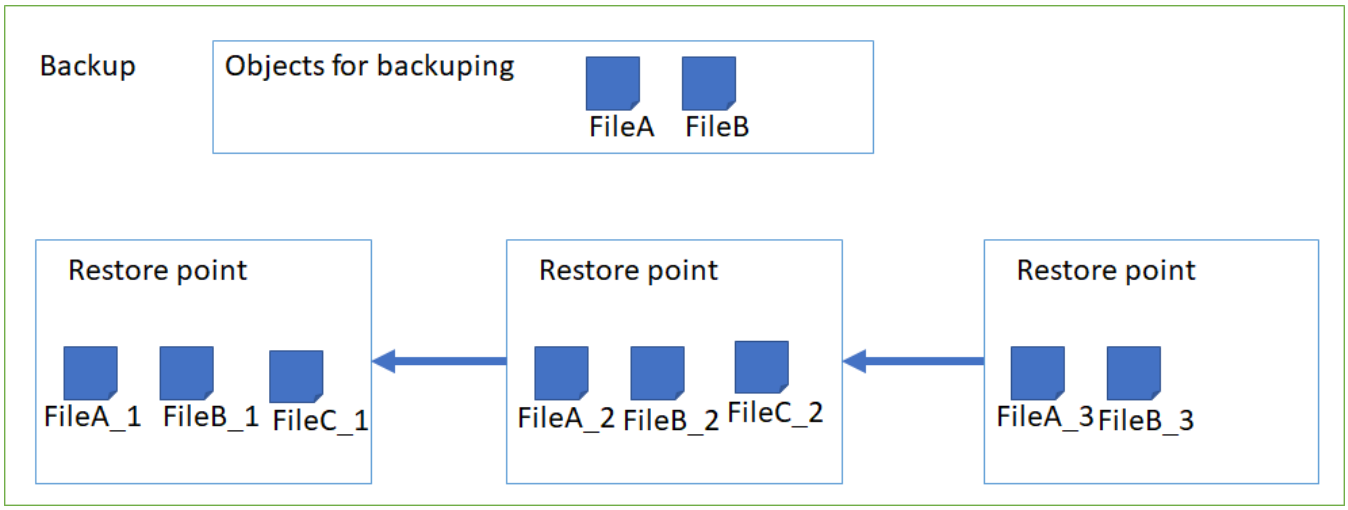
2020

Задание

Теормин

Бекап — в общем случае, это резервная копия каких-то данных, которая делается для того, чтобы в дальнейшем можно было восстановить эти данные, то есть откатиться до того момента, когда она была создана.

Точка восстановления — резервная копия объектов, созданная в определенный момент. Представлена датой создания и список резервных копий объектов, которые бекапились. Есть два типа точек восстановления - полноценные и инкрементальные. Полноценные точки содержат всю информацию про объекты, которые забекапились. Инкрементальные поинты - это разница (дельта) относительно предыдущей точки, т.е. мы храним только изменения.



Условие

В рамках лабораторной работы подразумевается разработка системы, которая управляет процессом создания бекапов. Для упрощения выполнения лабораторной, создавать физически резервные копии указанных файлов не требуется. Достаточно будет создать запись о том, что было сделано резервное копирование.

```
FileRestoreCopyInfo CreateRestore(string filePath)
{
    var fileInfo = new FileInfo(filePath);
    var fileRestoreCopyInfo = new FileRestoreCopyInfo(filePath, fileInfo.Size, DateTime.Now);
    //File.Copy(filePath, _pathWhereWeNeedToStoreOurBackup); <- Вот эту часть мы можем скипать
    return fileRestoreCopyInfo;
}
```

Информация о бекапе представлена в виде набора параметров: Id, CreationTime, BackupSize и список точек восстановления.

Алгоритмы создания и хранения

Для создания бекапа указываются объекты - список файлов. Должна быть реализована возможность в последствии этот список редактировать - добавлять и удалять объекты из списка объектов которые будут обрабатываться в алгоритме.

Система должна поддерживать несколько алгоритмов создания точек восстановления для бекапа, а также возможность увеличивать их количество. Результатом работы алгоритма является создание новой точки восстановления для указанного бекапа. Точка восстановления хранит о себе информацию о том, какие объекты были в ней забекаплены. В алгоритме должна быть возможность указать требуется ли создать полноценную точку или только дельту с прошлого раза (т.е. инкремент).

Требуется реализовать как минимум два алгоритма хранения:

1. Алгоритм раздельного хранения — файлы копируются в специальную папку и хранятся там раздельно.
2. Алгоритм общего хранения — все указанные в бекапе объекты складываются в один архив.

Алгоритмы очистки точек

Помимо создания, нужно контролировать количество хранимых точек восстановления. Чтобы не допускать накопления большого количества старых и неактуальных точек, требуется реализовать механизмы их очистки — они должны контролировать, чтобы цепочка точек восстановления не выходила за допустимый лимит. В рамках лабораторной подразумеваются такие типы лимитов:

1. По количеству - ограничивает длину цепочки точек восстановления (храним последние N точек)

2. По дате - ограничивает насколько старые точки будут хранится (храним все точки, которые были сделаны не позднее указанной даты)

3. По размеру - ограничивает суммарный размер, занимаемый бекапом (храним все последние точки, суммарный размер которых не превышает лимит)

4. Гибрид - возможность комбинировать лимиты. Пользователь может указывать, как комбинировать:

    - нужно удалить точку, если вышла за хотя бы один установленный лимит
    - нужно удалить точку, если вышла за все установленные лимиты

    Например, пользователь выбирает гибрид алгоритмов "по количеству" и "по дате". Если по одному из алгоритмов необходимо оставить 3 точки, а по другому — 5, то выбирается количество точек в соответствии с параметром, указанном при создании "гибрида" (использовать максимальное или минимальное значение отобранных точек).

Алгоритм должен учитывать то, что инкрементальные точки не должны остаться без точки, от которой взята дельта. В случае, если пришлось оставить точек больше, чем планировалось, результат выполнения алгоритма должен вернуть соответствующее предупреждение.

Тесты

Для проверки работоспособности разработанной системы предлагается реализовать юз-кейсы (пользовательские сценарии использования программы):

1. Кейс №1
    1. Я создаю бекап, в который добавляю 2 файла.
    2. Я запускаю алгоритм создания точки для этого бекапа — создается точка восстановления.
    3. Я должен убедиться, что в этой точке лежит информация по двум файлам.
    4. Я создаю следующую точку восстановления для цепочки
    5. Я применяю алгоритм очистки цепочки по принципу ограничения максимального количества указав длину 1.
    6. Я убеждаюсь, что в ответ получу цепочку длиной 1.
2. Кейс №2
    1. Я создаю бекап, в который добавляю 2 файла размером по 100 мб.
    2. Я создаю точку восстановления для него.
    3. Я создаю следующую точку, убеждаюсь, что точки две и размер бекапа 200 мб.

    4. Я применяю алгоритм очистки с ограничением по размеру, указываю 150 мб для цепочки и убеждаюсь, что остается один бекап.
3. Кейсы с инкрементами на тестирование алгоритмов сохранения.
4. Кейсы на два способа комбинации в гибридном лимите.

Notes

1. Для проверки работоспособности алгоритма работы со временем вам нужно будет продумать механизм путешествия во времени.

Ход рассуждений

Для реализации системы, управляющей созданием бекапов, была создана система классов, где основной класс Backup содержит в себе экземпляры классов RestorePoint и FileInfo, экземпляры классов, расширяющих абстрактные классы IPointStore, IPointController, а также использует в своих методах классы, реализующие интерфейс IPointCreate.
RestorePoint - класс, предназначенный для хранения данных точек восстановления и для работы с ними, для чего также содержит экземпляры класса FileInfo - класса для имитации работы с файловой системой, а также для хранения данных о файле.
IPointStore - абстрактный класс, от которого наследуются PointStoreSeparate и PointStoreUnion - алгоритмы хранения: раздельного и общего.
IPointController - абстрактный класс, предназначенный для контроля количества точек восстановления. Имеет потомков DateController, SizeController, NumberController - алгоритмы очистки точек с установленными лимитами по дате создания точек, суммарному размеру бекапа, а также по количеству точек - и потомка Hybrid, способного комбинировать алгоритмы очистки. С помощью классов CombineMaxAlgorithm и CombineMinAlgorithm, реализующих интерфейс ICombineAlgorithm, Hybrid может удалять как точки, вышедшие только за один из лимитов, так и точки, вышедшие за все лимиты.
IPointCreate - интерфейс, реализуемый классами PointCreateFull и PointCreateInc - алгоритмы создания полноценных точек восстановления и инкрементальных соответственно.
Таким образом, данная система классов реализует весь функционал, необходимый для работы системы, управляющей созданием бекапов, с помощью вышеперечисленных классов и интерфейсов.

Листинг

Файл Backup.java

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

public class Backup {
    private int id;
    private Calendar modificationTime;
    private int backupSize;
//    IPointCreate createAlgorithm;
    private IPointStore storeAlgorithm;
    private IPointController controller;
    private ArrayList<RestorePoint> restorePoints;
    private ArrayList<FileInfo> fileList;
    private int generalSize;

    public Backup(int id, IPointStore storeAlgorithm, IPointController controller) {
        this.id = id;
        updateModificationTime();
        this.backupSize = 0;
        this.storeAlgorithm = storeAlgorithm;
        this.controller = controller;
        this.restorePoints = new ArrayList<>();
        this.fileList = new ArrayList<>();
        this.generalSize = 0;
    }

    private void updateModificationTime() {
        this.modificationTime = CurrentDate.getDate();
    }

    private void updateGeneralSize() {
        generalSize = 0;
        for (RestorePoint restorePoint : restorePoints) {
            generalSize += restorePoint.getGeneralSize();
        }
    }

    private void updateBackup() {
        updateModificationTime();
        int oldSize = restorePoints.size();
        controller.deleteRedundant(restorePoints);
        updateGeneralSize();

    }

    public void createPoint(IPointCreate createAlgorithm) {
        createAlgorithm.createPoint(fileList, restorePoints, storeAlgorithm);
```

```
    updateBackup();
}

public void setController(IPointController controller) {
    this.controller = controller;
    updateBackup();
}

public int getGeneralSize() {
    return generalSize;
}

public int getRestorePointsNumber() {
    return restorePoints.size();
}

public int getFileNumber() {
    return fileList.size();
}

public ArrayList<FileInfo> getLastInfos() {
    int fullPoint = 0;
    if(restorePoints.size() == 0) {
        return new ArrayList<>();
    }
    for (int i = 1; i < restorePoints.size(); i++) {
        if(restorePoints.get(i).isFull()) {
            fullPoint = i;
        }
    }
    ArrayList<FileInfo> res = new ArrayList<>(restorePoints.get(fullPoint).getFileDataList());
    for(int i = fullPoint + 1; i < restorePoints.size(); i++) {
        if(restorePoints.get(i).getFileDataList().size() > 0) {
            for (FileInfo info : restorePoints.get(i).getFileDataList()) {
                if(getFile(info.getId(), res) == -1) {
                    res.add(new FileInfo(info));
                } else {
                    res.set(getFile(info.getId(), res), new FileInfo(info));
                }
            }
        }
    }
    return res;
    //return restorePoints.get(restorePoints.size() - 1).getFileDataList();
}

/*public void setCreationTime(Calendar creationTime) {
    this.modificationTime = modificationTime;
}*/

/*public void setBackupSize(int backupSize) {
    this.backupSize = backupSize;
```

```
    }*/

    public void addFile(FileInfo fileInfo) {
        fileList.add(fileInfo);
    }

    public void deleteFile(int id) throws Exception {
        int index = getFile(id, fileList);
        if(index == -1) {
            throw new Exception("No such file: " + id);
        }
        fileList.remove(index);
    }

    public void setFile(FileInfo fileInfo) throws Exception {
        int index = getFile(fileInfo.getId(), fileList);
        if(index == -1) {
            throw new Exception("No such file: " + fileInfo.getFilePath());
        }
        fileList.set(index, fileInfo);
    }


    public int getFile(int id, ArrayList<FileInfo> fileList) {
        for (int i = 0; i < fileList.size(); i++) {
            if (fileList.get(i).getId() == id) {
                return i;
            }
        }
        return -1;
    }
}
```

Файл CombineMaxAlgorithm.java

```
import java.util.ArrayList;

public class CombineMaxAlgorithm implements ICombineAlgorithm {

    public IPointController getHybrid(ArrayList<RestorePoint> restorePoints, IPointController a, IPointController b) {
        if(a.countRedundant(restorePoints) < b.countRedundant(restorePoints)) {
            return a;
        }
        return b;
    }
}
```

Файл CombineMinAlgorithm.java

```
import java.util.ArrayList;

public class CombineMinAlgorithm implements ICombineAlgorithm {

    public IPointController getHybrid(ArrayList<RestorePoint> restorePoints, IPointController a, IPointController b) {
        if(a.countRedundant(restorePoints) > b.countRedundant(restorePoints)) {
            return a;
        }
        return b;
    }
}
```

Файл CurrentDate.java

```
import java.util.Calendar;
import java.util.GregorianCalendar;

public class CurrentDate {
    private static CurrentDate instance;
    private static Calendar calendar;
    private CurrentDate() {
        calendar = new GregorianCalendar(2020, Calendar.OCTOBER, 1, 1, 1, 0);
    }
    public static Calendar getDate() {
        if(instance == null) {
            instance = new CurrentDate();
        }
        return (Calendar) calendar.clone();
    }

    public static void setCalendar(Calendar calendar) {
        CurrentDate.calendar = calendar;
    }
}
```

Файл DateController.java

```
import java.util.ArrayList;
import java.util.Calendar;

public class DateController extends IPointController {
    Calendar calendar;

    public DateController(Calendar calendar) {
        this.calendar = calendar;
    }

    public int countRedundant(ArrayList<RestorePoint> restorePoints) {
        if(restorePoints.size() == 0 || !restorePoints.get(0).getCreationTime().before(calendar)) {
            return 0;
        }
        int fullPoint = 0;
        for(int i = 1; i < restorePoints.size(); i++) {
            if(restorePoints.get(i).isFull()) {
                fullPoint = i;
                if(!restorePoints.get(fullPoint).getCreationTime().before(calendar)) {
                    break;
                }
            }
        }
        int res = restorePoints.size() - fullPoint;
        if(restorePoints.get(fullPoint).getCreationTime().before(calendar)) {
            System.out.println("This backup can't have less points than " + res);
        }
       return fullPoint;
    }
}
```

Файл FileInfo.java

```
import java.util.Calendar;

public class FileInfo {
    private int id;
    private int size;
    private Calendar creationTime;
    private String filePath;
    private String fileName;

    public FileInfo(int id, int size, Calendar creationTime, String filePath) {
        this.id = id;
        this.size = size;
        this.creationTime = creationTime;
        this.filePath = filePath;
        setFileName(filePath);
    }

    public FileInfo(FileInfo fileInfo) {
        this.size = fileInfo.size;
        this.creationTime = fileInfo.creationTime;
        this.filePath = fileInfo.filePath;
        setFileName(filePath);
    }

    public int getId() {
        return id;
    }

    public int getSize() {
        return size;
    }

    public Calendar getCreationTime() {
        return creationTime;
    }

    public String getFilePath() {
        return filePath;
    }

    public void setSize(int size) {
        this.size = size;
        updateCreationTime();
    }

    //public void setCreationTime(Calendar creationTime) {
    //    this.creationTime = creationTime;
    //}
```

```
    public void updateCreationTime() {
        this.creationTime = CurrentDate.getDate();
    }

    /*public void setFilePath(String filePath) {
        this.filePath = filePath;
        setFileName(filePath);
    }*/

    public String getFileName() {
        return fileName;
    }

    private void setFileName(String filePath) {
        this.fileName = filePath.substring(filePath.lastIndexOf('/') + 1);
        updateCreationTime();
    }
}
```

Файл Hybrid.java

```
import java.util.ArrayList;
import java.util.List;

public class Hybrid extends IPointController {
    IPointController[] controllers;
    ICombineAlgorithm combineAlgorithm;
    IPointController chosenController;

    public Hybrid(ICombineAlgorithm combineAlgorithm, IPointController ... controllers) throws Exception {
        if(controllers.length == 0) {
            throw new Exception("No one controller enabled");
        }
        this.controllers = controllers;
        this.combineAlgorithm = combineAlgorithm;
    }

    public int countRedundant(ArrayList<RestorePoint> restorePoints) {
        chosenController = controllers[0];
        for (int i = 1; i < controllers.length; i++) {
            chosenController = combineAlgorithm.getHybrid(restorePoints, chosenController, controllers[i]);
        }
        return chosenController.countRedundant(restorePoints);
    }

    @Override
    public void deleteRedundant(ArrayList<RestorePoint> restorePoints) {
        countRedundant(restorePoints);
        chosenController.deleteRedundant(restorePoints);
    }
}
```

Файл ICombineAlgorithm.java

```
import java.util.ArrayList;

public interface ICombineAlgorithm {
    IPointController getHybrid(ArrayList<RestorePoint> restorePoints, IPointController a, IPointController b);
    //int getStartValue();
}
```

Файл IPointController.java

```
import java.util.ArrayList;
import java.util.List;

public abstract class IPointController {
    abstract int countRedundant(ArrayList<RestorePoint> restorePoints);
    public void deleteRedundant(ArrayList<RestorePoint> restorePoints) {
        int number = countRedundant(restorePoints);
        for(int i = 0; i < number; i++) {
            restorePoints.remove(0);
        }
    }
}
```

Файл IPointCreate.java

```
import java.util.ArrayList;

public interface IPointCreate {
    void createPoint(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, IPointStore storeAlgorithm);
}
```

Файл IPointStore.java

```
import java.util.ArrayList;
import java.util.Map;

public abstract class IPointStore {

    abstract Map.Entry<Integer, ArrayList<FileInfo>> store(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, boolean
incremental);

    public int getSizeDelta(FileInfo newFileInfo, FileInfo oldFileInfo) {
        return newFileInfo.getSize() - oldFileInfo.getSize();
    }

    public int getFile(int id, ArrayList<FileInfo> fileList) {
        for (int i = 0; i < fileList.size(); i++) {
            if (fileList.get(i).getId() == id) {
                return i;
            }
        }
        return -1;
    }

    protected Map.Entry<Integer, ArrayList<FileInfo>> getData(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, String
directoryPath, boolean incremental) {
        int fullPoint = 0;
        for (int i = restorePoints.size() - 1; i > 0; i--) {
            if (restorePoints.get(i).isFull()) {
                fullPoint = i;
                break;
            }
        }

        int generalSize = 0;
        ArrayList<FileInfo> newFileDataList = new ArrayList<>();
        ArrayList<FileInfo> oldFileDataList;
        if(restorePoints.size() != 0) {
            oldFileDataList = restorePoints.get(fullPoint).getFileDataList();
        } else {
            oldFileDataList = new ArrayList<>();
        }
        for (FileInfo fileInfo : fileList) {
            String filePath = directoryPath + "/R_" + fileInfo.getFileName();
            int index = getFile(fileInfo.getId(), oldFileDataList);
            FileInfo delta;
            if (!incremental || index == -1) {
                delta = new FileInfo(fileInfo.getId(), fileInfo.getSize(), fileInfo.getCreationTime(), filePath);
                newFileDataList.add(delta);
                //writeDown(delta, fileInfo.getFilePath());
                generalSize += fileInfo.getSize();
            } else {
```

```
            if (fileInfo.getCreationTime() == oldFileDataList.get(index).getCreationTime()) {
                continue;
            }
            int sizeDelta = getSizeDelta(fileInfo, oldFileDataList.get(index));

            delta = new FileInfo(fileInfo.getId(), sizeDelta, fileInfo.getCreationTime(), filePath);
            newFileDataList.add(delta);
            //writeDownDelta(delta, oldFileInfo.getFilePath(), fileInfo.getFilePath());
            generalSize += sizeDelta;
          }
        }
        return Map.entry(generalSize, newFileDataList);
    }
}
```

Файл Main.java

```
import java.io.File;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;

public class Main {

   public static void main(String[] args) {
      try {
         FileInfo A = new FileInfo(0, 110, CurrentDate.getDate(), "C:/test/A");
         FileInfo B = new FileInfo(1, 100, CurrentDate.getDate(), "C:/test/B");

         IPointStore separateStore = new PointStoreSeparate("C:/test/backups");
         IPointStore unionStore = new PointStoreUnion("C:/test/backups");
         IPointCreate fullCreate = new PointCreateFull();
         IPointCreate incCreate = new PointCreateInc();

         Backup backup = new Backup(0, separateStore,  new NumberController(1));
         backup.addFile(A);
         backup.addFile(B);
         backup.createPoint(fullCreate);
         System.out.println(backup.getGeneralSize() + " - size, files: " + backup.getFileNumber());
         A.setSize(100);
         B.updateCreationTime();
         backup.createPoint(fullCreate);
         System.out.println(backup.getGeneralSize() + " - size, points: " + backup.getRestorePointsNumber());
         backup.setController(new NumberController(3));
         backup.createPoint(fullCreate);
         System.out.println(backup.getGeneralSize() + " - size, points: " + backup.getRestorePointsNumber());
         backup.setController(new SizeController(150));
         System.out.println(backup.getGeneralSize() + " - size, points: " + backup.getRestorePointsNumber());

         FileInfo C = new FileInfo(2, 100, CurrentDate.getDate(), "C:/test/C");
         FileInfo D = new FileInfo(3, 60, CurrentDate.getDate(), "C:/test/D");
         Backup backup1 = new Backup(1, unionStore, new NumberController(4));
         backup1.addFile(C);
         backup1.addFile(D);
         backup1.createPoint(fullCreate);
         //C.setSize(300);
         D.updateCreationTime();
         ArrayList<FileInfo> infos = backup1.getLastInfos();
         System.out.println("C path is: " + infos.get(0).getFilePath());
         System.out.println("D path is: " + infos.get(1).getFilePath());
         backup1.createPoint(incCreate);
         D.setSize(100);
         infos = backup1.getLastInfos();
         System.out.println("C path is: " + infos.get(0).getFilePath());
         System.out.println("D path is: " + infos.get(1).getFilePath());
```

```
            FileInfo E = new FileInfo(4, 120, CurrentDate.getDate(), "C:/test/E");
            FileInfo F = new FileInfo(5, 80, CurrentDate.getDate(), "C:/test/F");

            CurrentDate.setCalendar(new GregorianCalendar(2020, Calendar.OCTOBER, 30, 1, 1, 0));

            IPointController controller = new Hybrid(new CombineMinAlgorithm(), new DateController(CurrentDate.getDate()), new
NumberController(3));
            CurrentDate.setCalendar(new GregorianCalendar(2020, Calendar.OCTOBER, 1, 1, 1, 0));
            Backup backup2 = new Backup(2, unionStore, new NumberController(4));
            backup2.addFile(E);
            backup2.addFile(F);
            backup2.createPoint(fullCreate);
            backup2.createPoint(fullCreate);
            CurrentDate.setCalendar(new GregorianCalendar(2020, Calendar.NOVEMBER, 30, 1, 1, 0));
            backup2.createPoint(fullCreate);
            backup2.createPoint(fullCreate);
            backup2.setController(controller);
            System.out.println("Points number: " + backup2.getRestorePointsNumber());

            //DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
            //System.out.println("/Point_from_" + df.format(Calendar.getInstance().getTime()));

        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Файл NumberController.java

```
import java.util.ArrayList;

public class NumberController extends IPointController {
    int number;
    public NumberController(int number) {
        this.number = number;
    }
    public int countRedundant(ArrayList<RestorePoint> restorePoints) {
        if(restorePoints.size() <= number) {
            return 0;
        }
        int fullPoint = 0;
        for(int i = 1; i < restorePoints.size() && restorePoints.size() - fullPoint > number; i++) {
            if(restorePoints.get(i).isFull()) {
                fullPoint = i;
            }
        }
        int res = restorePoints.size() - fullPoint;
        if(res > number) {
            System.out.println("This backup can't have less points than " + res
                    + "because it has too many incremental points");
        }
        return fullPoint;
    }

}







/*import java.util.ArrayList;

public class PointNumberController extends IPointController {
    private int number;

    public PointNumberController(int number) {
        this.number = number;
    }
    public void deleteRedundant(ArrayList <RestorePoint> restorePoints) {
        if(restorePoints.size() < number) {
            return;
        }
        int lastFullPoint = 0;
        ArrayList <RestorePoint> reducedRestorePoints = new ArrayList<>();
        for(int i = 1; i < restorePoints.size(); i++) {
            if (restorePoints.get(i).isFull() && restorePoints.size() - lastFullPoint - 1 > number) {
                lastFullPoint = i;
```

```
            }
            if(restorePoints.size() - lastFullPoint - 1 <= number) {
                reducedRestorePoints.add(restorePoints.get(i));
            }
        }
        restorePoints.clear();
        restorePoints.addAll(reducedRestorePoints);
    }
}
*/
```

Файл PointCreateFull.java

```
import java.util.ArrayList;
import java.util.Map;

public class PointCreateFull implements IPointCreate {
    public void createPoint(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, IPointStore storeAlgorithm) {

        Map.Entry<Integer, ArrayList<FileInfo>> delta = storeAlgorithm.store(fileList, restorePoints, false);
        restorePoints.add(new RestorePoint(restorePoints.size(), fileList, delta.getValue(), delta.getKey(), false));
    }
}
```

Файл PointCreateInc.java

```
import java.util.ArrayList;
import java.util.Map;

public class PointCreateInc implements IPointCreate {
    //private IPointStore storeAlgorithm;

    //public PointCreateInc(IPointStore storeAlgorithm) {
    //    this.storeAlgorithm = storeAlgorithm;
    //}

    public void createPoint(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, IPointStore storeAlgorithm) {
        if(restorePoints.size() == 0) {
            PointCreateFull fullAlgorithm = new PointCreateFull();
            fullAlgorithm.createPoint(fileList, restorePoints, storeAlgorithm);
            return;
        }

        Map.Entry<Integer, ArrayList<FileInfo>> delta = storeAlgorithm.store(fileList, restorePoints, true);
        restorePoints.add(new RestorePoint(restorePoints.size(), fileList, delta.getValue(), delta.getKey(), true));
    }


}
```

Файл PointStoreSeparate.java

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Map;

public class PointStoreSeparate extends IPointStore{
    String directoryPath;
    //IPointCreate createAlgorithm;
    public PointStoreSeparate(String directoryPath){//, IPointCreate createAlgorithm) {
        this.directoryPath = directoryPath;
        //this.createAlgorithm = createAlgorithm;
        //createDirectory(directoryPath);
        //openDirectory(directoryPath);
    }
    public Map.Entry<Integer, ArrayList<FileInfo>> store(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, boolean
incremental) {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
        String pointPath = directoryPath + "/Point_from_" + df.format(CurrentDate.getDate().getTime());
        //createDirectory(pointPath);
        //openDirectory(pointPath);
        Map.Entry<Integer, ArrayList<FileInfo>> delta = getData(fileList, restorePoints, pointPath, incremental);
        return delta;
    }
}
```

Файл PointStoreUnion.java

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Map;

public class PointStoreUnion extends IPointStore {
    String directoryPath;

    //IPointCreate createAlgorithm;
    public PointStoreUnion(String directoryPath) {//, IPointCreate createAlgorithm) {
        this.directoryPath = directoryPath;
        //this.createAlgorithm = createAlgorithm;
        //createDirectory(directoryPath);
    }

    public Map.Entry<Integer, ArrayList<FileInfo>> store(ArrayList<FileInfo> fileList, ArrayList<RestorePoint> restorePoints, boolean
incremental) {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
        //createDirectory(directoryPath);
        //openDirectory(directoryPath);
        Map.Entry<Integer, ArrayList<FileInfo>> delta = getData(fileList, restorePoints, directoryPath, incremental);
        return delta;
    }


}
```

Файл RestorePoint.java

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.GregorianCalendar;

public class RestorePoint {
    private int id;
    private ArrayList<FileInfo> fileList;
    private ArrayList<FileInfo> fileDataList;
    private Calendar creationTime;
    private int generalSize;
    private boolean incremental;

    public RestorePoint(int id, ArrayList<FileInfo> fileList, ArrayList<FileInfo> fileDataList, int generalSize, boolean incremental) {
        this.id = id;
        this.fileList = new ArrayList<FileInfo>(fileList);
        this.creationTime = CurrentDate.getDate();
        this.fileDataList = fileDataList;
        this.generalSize = generalSize;
        this.incremental = incremental;
    }

    public int getGeneralSize() {
        return generalSize;
    }

    public int getId() {
        return id;
    }

    public boolean isFull() {
        return !incremental;
    }

    public Calendar getCreationTime() {
        return (GregorianCalendar) creationTime.clone();
    }

    ArrayList<FileInfo> getFileDataList() {
        return new ArrayList<>(fileDataList);
    }
}
```

Файл SizeController.java

```
import java.util.ArrayList;

public class SizeController extends IPointController {
    private int size;
    public SizeController(int size) {
        this.size = size;
    }
    public int countRedundant(ArrayList<RestorePoint> restorePoints) {

        int fullPoint = 0;
        int generalSize = 0;
        for(int i = 0; i < restorePoints.size(); i++) {
            generalSize += restorePoints.get(i).getGeneralSize();
        }
        if(generalSize <= size) {
            return 0;
        }
        int uselessSize = 0;
        int currentSize = 0;
        for(int i = 0; i < restorePoints.size() && generalSize - uselessSize > size; i++) {
            if(restorePoints.get(i).isFull())  {
                fullPoint = i;
                uselessSize += currentSize;
                currentSize = 0;
            }
            currentSize += restorePoints.get(i).getGeneralSize();
        }
        int res = restorePoints.size() - fullPoint;
        if(generalSize - uselessSize > size) {
            System.out.println("This backup can't have less points than " + res);
        }
        return fullPoint;
    }
}
```