№ 3-4 Классы, интерфейсы и наследование, коллекции и исключения

Задание

Создать консольное приложение, удовлетворяющее следующим требованиям:

1) Использовать возможности ООП: классы, наследование, полиморфизм, инкапсуляция.
2) Каждый класс должен иметь отражающее смысл название и информативный состав (поля, set,get, конструкторы, toString). Постарайтесь переопределить, где это возможно методы Object.
3) Наследование должно применяться только тогда, когда это имеет смысл.
4) Должен быть минимум один *интерфейс*, *абстрактный класс* и один *внутренний класс*, а также *перечисление*
5) При кодировании должны быть использованы соглашения java code convention.
6) Классы должны быть разложены по пакетам.
7) Для сбора объектов использовать стандартные коллекции.
8) Добавить обработку исключений, использовать стнадратные классы исключений (можно создать свои наследуя стандартные)
9) Подлючить Log4j и весь консольный вывод направлять туда. Информировать о создании объектов, исключениях, выводить результаты запросов.
10) Откомпилировать и выполнить в среде IntellijIdea и в командной строке.

Далее приведены варианты заданий.

Библиотека. Определить иерархию книг, журналов, буклетов и .т.д. Создать несколько объектов. Собрать домашнюю библиотеку с определением ее стоимости. Создать библиотекаря. Его функции: провести сортировку книг в библиотеке на основе года издания, Найти печатное издание, с соответствующим числом страниц.

Электроприборы. Определить иерархию электроприборов. Включить их в розетку. Собрать квартиру с приборами. Создать управляющего. Его функции: подсчитать потребляемую мощность, провести сортировку приборов в квартире на основе мощности, найти прибор, соответствующий заданному диапазону параметров.

Авиакомпания. Определить иерархию самолетов. Создать авиакомпанию. Посчитать общую вместимость и грузоподъемность. Создать диспечера. Его функции: провести сортировку самолетов компании по дальности полета, найти самолет в компании, соответствующий заданному диапазону параметров потребления горючего.

Таксопарк. Определить иерархию автомобилей (любых в том числе и грузовых). Создать таксопарк. Создать управляющего. Его функции: подсчитать стоимость автопарка, провести сортировку автомобилей парка по расходу топлива, найти автомобиль в компании, соответствующий заданному диапазону параметров скорости.

Туристические путевки. Сформировать набор предложений клиенту по выбору туристической путевки различного типа (отдых, экскурсии, лечение, шопинг, круиз и т. д.) для оптимального выбора. Создать оператора. Его функции: подбор предложения, учитывать возможность выбора транспорта, питания и числа дней.

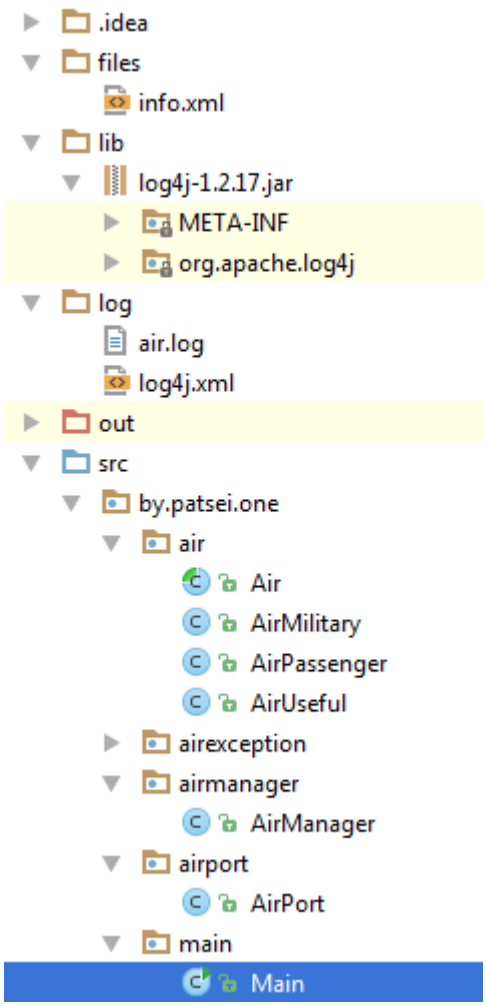
Программа передач. Определить иерархию телевизионных передач (новости, фильмы, мультики, реклама и т.п.). Сформировать программу на день. Создать программного директора. В его функции входит поиск самой длинной передачи, передачи, относящейся к определенному типу, подсчет длительности всей программы, сортировка передач (критерий определите сами).

Университет. Определить иерархию студентов по специальностям и форме обучения. Набрать поток. Создать декана. Его функции: подсчитать студентов по курсам, подбор студентов по критериям (курс, успеваемость и т.п.), отсортировать студентов (по курсам и фамилиям).

IT компания. Определить иерархию сотрудников: инженер, сис админ, программисты (junior, senior и т.п.). Набрать компанию по заданному плану набора. Создать директора. Его функции: подсчитать сотрудников, отсортировтаь по зарплате, найти сотрудников с заданным уровнем навыков.

Методические рекомендации

Структура директорий и пакетов должна быть примерно следующая



В директории files - должен быть расположен файл с инициализаторами для объектов. Наприме, info.xml (нужен для выполнения следующей лабараторной работы) info.json

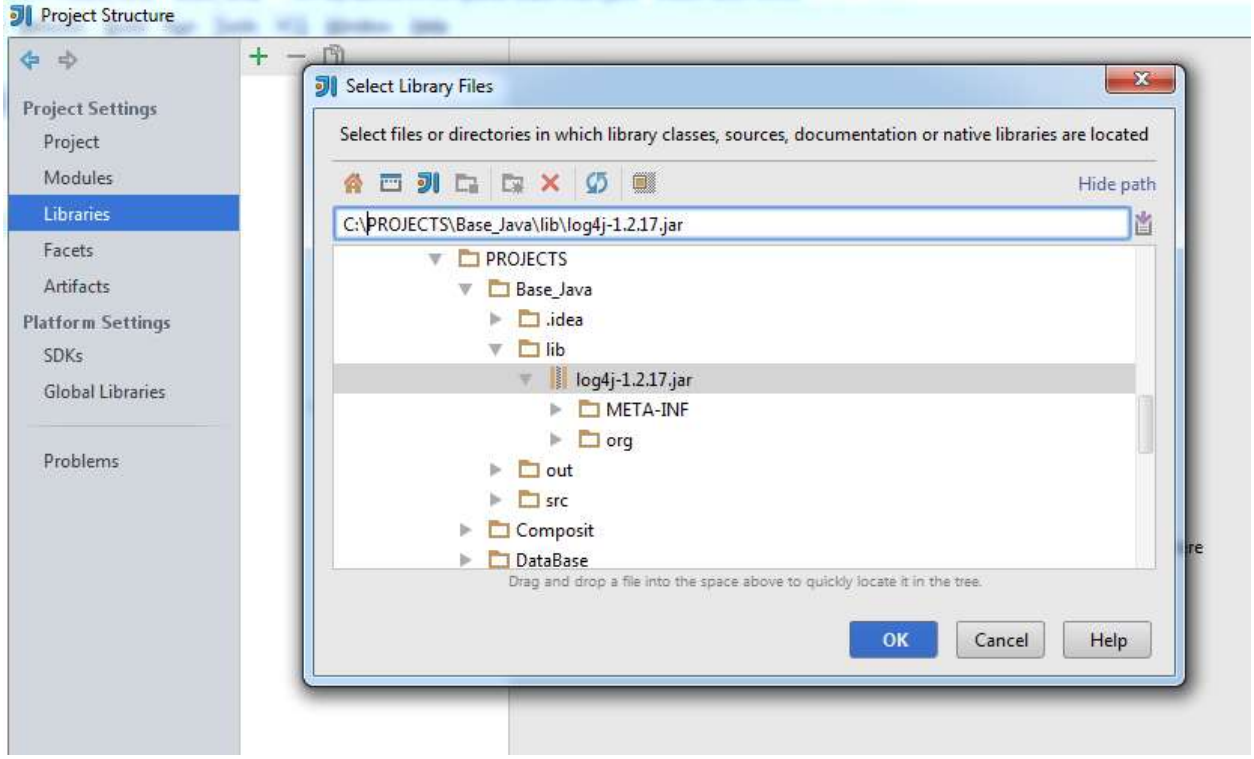
```
<?xml version="1.0" encoding="UTF-8"?>
<air>

    <AirPassenger
      uid = "1233" higth = "40" length = "70.6 " weith = "37000"
rang = "12500" max_speed = "950" count_passangers = "436" model =
"boing">
    </AirPassenger>
    <AirUseful
        uid = "37" higth = "24.5" length = "35.6" weith = "25000"
rang = "900" max_speed = "700" loadCapacity ="23000">
    </AirUseful>
    <AirMilitary
        uid = "89" higth = "40" length = "71.6"  weith = "45000"
rang = "13000" max_speed = "830" equipment = "Pusk">
    </AirMilitary>
</air>
```

Напройка и подключение logger.

Для подключения log4j необходимо выполнить следующее.

Создать в проекте папку lib и скопировать туда log4j-1.x.x.jar. Затем подключите File->Progect Structure. Выберите Libraries и добавьте туда файл.



В окне должна появиться информация о подключении log4j-1.x.x.

Создайте папку log. В ней должна быть конфигурация logger и файлы логирования.
Конфигурационные файлы бывают 2-х видов log4j.xml м log4j.properties. Вообще эти файлы взаимозаменяемые, но некоторые взможности (например фильтры) log4j.properties не поддерживаются.

Конфигурирационный файл log4j.xml может быть слледующий:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration
xmlns:log4j='http://jakarta.apache.org/log4j/'>
   <appender name="Console"
class="org.apache.log4j.ConsoleAppender">
      <layout class="org.apache.log4j.SimpleLayout"/>
   </appender>
   <appender name="one"
class="org.apache.log4j.DailyRollingFileAppender">
      <param name="File" value="log/air.log" />
      <param name="DatePattern" value=".yyyy-MM-dd" />
      <layout class="org.apache.log4j.PatternLayout" >
         <param name="ConversionPattern" value="%d{dd.MM.yyyy
HH:mm:ss} [%t] %-5p %c %x - %m%n"/>
      </layout>
   </appender>
   <root>
```

```
        <priority value="debug"/>
        <appender-ref ref="one" />
    </root>
</log4j:configuration>
```

Вывод регистратора может быть направлен в различные места назначения: файл, консоль и т. д. Каж дому из них соответствует класс, реализующий интерфейс org.apache.log4j.Appender. Кроме того, вывод в базу данных можно произвести с помощью класса JDBCAppender, в журнал событий ОС — NTEventLogAppender, на SMTP-сервер — SMTPAppender. Если логгер — это та точка, откуда уходят сообщения в коде, то аппендер — это та точка, куда они приходят в конечном итоге. Например, файл или консоль. Существует возможность написать собственный класс аппендер и использовать его.

Основными аппендерами, использующимися наиболее широко, являются файловые аппендеры. Их есть несколько типов:

— org.apache.log4j.FileAppender
— org.apache.log4j.RollingFileAppender
— org.apache.log4j.DailyRollingFileAppender

Логгеры связываются с аппендерами в соотношении «многие ко многим» — у одного логгера может быть несколько аппендеров, а к одному аппендеру может быть привязано несколько логгеров. Важно понимать, что аппендеры наследуются от родительских логгеров. Уровень логирования наследуется (или устанавливается) независимо от аппендера.

Напимер, аппендер - консоль

```
<appender name="Console"
class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.SimpleLayout"/>
</appender>
```

Вывод регистратора может иметь различный формат. Каждый формат представлен классом, производным от Layout. Все методы класса Layout предназначены только для создания подклассов. Для конфигурирования формата вывода используются наследники класса org.apache.log4j.Layout:

— org.apache.log4j.SimpleLayout
— org.apache.log4j.HTMLLayout
— org.apache.log4j.xml.XMLLayout
— org.apache.log4j.PatternLayout

org.apache.log4j.SimpleLayout — наиболее простой вариант. На выходе читается уровень вывода и сообщение.

org.apache.log4j.HTMLLayout — данный компоновщик форматирует сообщения в виде HTML-страницы.

org.apache.log4j.xml.XMLLayout — формирует сообщения в виде XML.

org.apache.log4j.PatternLayout и org.apache.log4j.EnhancedPatternLayout — используют шаблонную строку для форматирования выводимого сообщения.

Например:

```
<param name="ConversionPattern" value="%d{dd.MM.yyyy HH:mm:ss} [%t] %-5p %c %x - %m%n"/>
```

Данный форматтер принимает параметром ConversionPattern — шаблон вывода лога, где
%d{DATE} — выводит дату-время. В скобках можно указать собственный формат вывода.
%t — выводит имя потока, выводящего сообщение, для однопоточного приложения будет выводить main;
%5p — выводит уровень лога (ERROR, DEBUG, INFO и пр.), где цифра указывает число выводимых символов, если символов меньше, то сообщение будет дополнено пробелами;
%c{6} — категория с числом выдаваемых уровней. Категорией в общем случае будет имя класса с пакетом. Обычно это строка, где уровни разделены точками. По умолчанию без {} будет выводить полный путь к корню проекта. Верхний уровень при значении 1 будет выводить только имя класса;
%M — имя метода, в котором произошел вызов записи в лог;
%L — номер строки, в которой произошел вызов записи в лог;
%m — собственно сообщение, передаваемое в лог;
%n — перевод строки.

Для того чтобы разобраться с Logпочитате gfgre Loging из папки **Литература**

После этого в классе Main допишите статический блок

```
static{
    new DOMConfigurator().doConfigure("log/log4j.xml",
LogManager.getLoggerRepository());
}

private static final Logger LOG = Logger.getLogger(Main.class);
```

А в методе main

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


    LOG.info("Starting program_____________________________");
```

Запустите проект, проверьте создается ли log file

Для того что какой-либо класс мог записывать log необходимо его подключать:

```
public abstract class Air {
    private static final Logger LOG =
Logger.getLogger(Air.class);
```

Вооще посылаемые сообщения различаются по приоритету:
FATAL - произошла фатальная ошибка - у этого сообщения наивысший приоритет
ERROR - в программе произошла ошибка
WARN - предупреждение в программе что-то не так
INFO - информация. К сведению трудящися.
DEBUG - детальная информация для отладки
TRACE - наиболее полная информация. трассировка выполнения программы. Наиболее низкий приоритет.
Чтобы послать сообщение нужного приоритета вызывается соответствующий метод: LOG.info("информация");
LOG.error("произошла ошибка").

Проектирование класса-контейнера

Класс –контейнер, в данном случае Аэропорт может содержать коллекцию созданных объектов, конструкторы, set и get, добавления и удаления :

```
public class AirPort {
..
    private ArrayList<Air> airlist;
..
    public AirPort() { ..}
    public AirPort(ArrayList<Air> airport) {...    }
    public List<Air> getAirlist() { ...   }
    public void setAirlist(ArrayList<Air> airlist) { ... }
    public boolean add (Air item) throws LogicalException { …}
public boolean remove (Air item){ …}
@Override
public String toString() {..}
```

Проектирование управляющего класса

Класс должен содержать следующие методы: чтения объектов их xml, генерации объектов и запросы.
Например:

```
public class AirManager {
//считать с источника----------------------------------------
public AirPort createAirPort(AirPort someAir, int maxpassCount, int
maxuseCount, int maxmilCount, String filename) throws
ParserConfigurationException, IOException, SAXException, TechnicalException,
LogicalException {
// открывает xml файл и выбирает из него указанное число разных самолетов,
добавляет их в аэропорт
}
// сгенерировать аэропорт-------------------------------------
  public AirPort generateAirPort ( AirPort someAir, int maxpassCount, int
maxuseCount, int maxmilCount)throws LogicalException {
// генерирует заданное количество самолетов определенного вида со случайными
параметрами
```

```
}

// емкость пассажирских----------------------------------------
    public int countPassCapacity (AirPort anyAirport){
//     }
// емкость грузовых ----------------------------------------
    public  int  countLoadCapacity (AirPort anyAirport){
//     }

    Comparator<Air> compare = new Comparator<Air>() {
        @Override
        public int compare(Air o1, Air o2) {
               return Integer.compare(o1.getRang(), o2.getRang());
                }
    };

    public AirPort sortByRange (AirPort anyAirport){
     anyAirport.getAirlist().sort(compare);
        return anyAirport;
    }

    public ArrayList<Air> findBySpeed(AirPort Minsk, int lower, int upper) {
//     }
```

Проектирование класса и метода main

В main создать два объекта-контейнера (например здесь - аэропоты) и одного мэнеджера. Он должен создать объекты контейнеры разными способами (чтение и генерация) и выполить все запросы.

Вопросы

1.Перечислите состав класса.
2. Где и как могут использоваться [**static**] [**abstract**] [**final**] в контексте класса?
3. Где могут использоваться слова super и this?
4. Для чего используется модификатор native?
5. Что такое логический блок?
6. Определите параметризованный класс.
7. Как используется метасимвол «?»
8. Какие существуют generic-ограничения?
9. Что могут содержать перечисления? Приведите пример
10. Какие существуют ограничения для перечислений?
11. Что такое методы подставки?
12. Состав класса Object.
13. Перечислите соглашения по equlas.
14. Перечислите соглашения по hashCode() .
15. Перечислите соглашения по toString().
16. Поясните разницу между «неглубким» и «глубоким» клонированием? Приведите пример.
17. Как можно использовать метод void finalize()?
18. Что такое внутренние классы (inner)? Привила использования.
19. Что такое вложенные (nested) классы? Привила использования.
20. Что такое анонимные (anonymous) классы?

21. Правила определения и наследования интерфейсов.

22. Приведите иерархию исключений и ошибок? Поясните проверяемые и непроверяемые исключения.