package com.javarush.test.level18.lesson03.task01;  
  
import java.io.FileInputStream;  
  
/\* Максимальный байт  
Ввести с консоли имя файла  
Найти максимальный байт в файле, вывести его на экран.  
Закрыть поток ввода-вывода  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson03.task02;  
  
import java.io.FileInputStream;  
  
/\* Минимальный байт  
Ввести с консоли имя файла  
Найти минимальный байт в файле, вывести его на экран.  
Закрыть поток ввода-вывода  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson03.task03;  
  
import java.io.FileInputStream;  
  
/\* Самые частые байты  
Ввести с консоли имя файла  
Найти байт или байты с максимальным количеством повторов  
Вывести их на экран через пробел  
Закрыть поток ввода-вывода  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson03.task04;  
  
import java.io.FileInputStream;  
  
/\* Самые редкие байты  
Ввести с консоли имя файла  
Найти байт или байты с минимальным количеством повторов  
Вывести их на экран через пробел  
Закрыть поток ввода-вывода  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson03.task05;  
  
import java.io.FileInputStream;  
  
  
  
  
/\* Сортировка байт  
Ввести с консоли имя файла  
Считать все байты из файла.  
Не учитывая повторений - отсортировать их по байт-коду в возрастающем порядке.  
Вывести на экран  
Закрыть поток ввода-вывода  
  
Пример байт входного файла  
44 83 44  
  
Пример вывода  
44 83  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson05.task01;  
  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.io.IOException;  
  
/\* Исправить ошибки  
Исправить функциональность в соответствии с требованиями  
Программа должна:  
1. переписать все байты одного файла в другой одним куском.  
2. закрывать потоки ввода-вывода  
Подсказка: 4 ошибки  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws IOException {  
 FileInputStream inputStream = new FileInputStream("c:/data.txt");  
 // Создаем поток-записи-байт-в-файл  
 FileOutputStream outputStream = new FileOutputStream("c:/result.txt");  
  
 if (inputStream.read() >= 0) {  
 //читаем весь файл одним куском  
 byte[] buffer = new byte[inputStream.available()];  
 int count = inputStream.read(buffer);  
 outputStream.write(buffer, 0, count);  
 }  
  
 inputStream.reset();  
 outputStream.flush();  
 }  
}

package com.javarush.test.level18.lesson05.task02;  
  
/\* Подсчет запятых  
С консоли считать имя файла  
Посчитать в файле количество символов ',', количество вывести на консоль  
Закрыть потоки. Не использовать try-with-resources  
  
Подсказка: нужно сравнивать с ascii-кодом символа ','  
\*/  
  
import java.io.\*;  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
   
  
 }  
}

package com.javarush.test.level18.lesson05.task03;  
  
/\* Разделение файла  
Считать с консоли три имени файла: файл1, файл2, файл3.  
Разделить файл1 по следующему критерию:  
Первую половину байт записать в файл2, вторую половину байт записать в файл3.  
Если в файл1 количество байт нечетное, то файл2 должен содержать бОльшую часть.  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.test.level18.lesson05.task04;  
  
/\* Реверс файла  
Считать с консоли 2 имени файла: файл1, файл2.  
Записать в файл2 все байты из файл1, но в обратном порядке  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
 }  
}

package com.javarush.test.level18.lesson05.task05;  
  
/\* DownloadException  
1 Считывать с консоли имена файлов.  
2 Если файл меньше 1000 байт, то:  
2.1 Закрыть потоки  
2.2 выбросить исключение DownloadException  
\*/  
  
import java.io.BufferedReader;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
public class Solution {  
 public static void main(String[] args) throws DownloadException {  
  
 }  
  
 public static class DownloadException extends Exception{  
  
 }  
}

package com.javarush.test.level18.lesson08.task01;  
  
/\* Wrapper (Decorator)  
Разберись, что делает программа  
Аналогично классу DecoratorRunnableImpl создай класс DecoratorMyRunnableImpl  
\*/  
  
public class Solution {  
  
 public static void main(String[] args) {  
 new Thread(new DecoratorRunnableImpl(new DecoratorMyRunnableImpl(new RunnableImpl()))).start();  
 }  
   
 public static class RunnableImpl implements Runnable {  
 @Override  
 public void run() {  
 System.*out*.println("RunnableImpl body");  
 }  
 }  
  
 public static class DecoratorRunnableImpl implements Runnable {  
 private Runnable component;  
  
 public DecoratorRunnableImpl(Runnable component) {  
 this.component = component;  
 }  
  
 @Override  
 public void run() {  
 System.*out*.print("DecoratorRunnableImpl body ");  
 component.run();  
 }  
 }  
  
}

package com.javarush.test.level18.lesson08.task02;  
  
import java.io.\*;  
  
/\* Расширяем AmigoOutputStream  
Используя шаблон проектирования Wrapper (Decorator) расширьте функциональность AmigoOutputStream  
В классе QuestionFileOutputStream при вызове метода close() должна быть реализована следующая функциональность:  
1. Вывести в консоль фразу [Вы действительно хотите закрыть поток? Д/Н]  
2. Считайте строку  
3. Если считанная строка равна [Д], то закрыть поток  
4. Если считанная строка не равна [Д], то не закрывать поток  
\*/  
  
public class QuestionFileOutputStream implements AmigoOutputStream {  
  
}

package com.javarush.test.level18.lesson08.task02;  
  
import java.io.IOException;  
  
public interface AmigoOutputStream {  
 public void flush() throws IOException;  
  
 void write(int b) throws IOException;  
  
 void write(byte[] b) throws IOException;  
  
 void write(byte[] b, int off, int len) throws IOException;  
  
 void close() throws IOException;  
}

package com.javarush.test.level18.lesson08.task03;  
  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
  
/\* AmigoOutputStream  
1 Измените класс AmigoOutputStream так, чтобы он стал Wrapper-ом для класса FileOutputStream. Используйте наследование.  
2 При вызове метода close() должны выполняться следующая последовательность действий:  
2.1 вызвать метод flush()  
2.2 дописать следующий текст [JavaRush © 2012-2013 All rights reserved.], используйте метод getBytes()  
2.3 закрыть поток методом close()  
\*/  
  
public class AmigoOutputStream {  
 public static String *fileName* = "C:/tmp/result.txt";  
  
 public static void main(String[] args) throws FileNotFoundException {  
 new AmigoOutputStream(new FileOutputStream(*fileName*));  
 }  
  
}  
}

package com.javarush.test.level18.lesson08.task04;  
  
public class UnsupportedFileNameException extends Exception {  
}

package com.javarush.test.level18.lesson08.task04;  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
  
/\* UnsupportedFileName  
Измените класс TxtInputStream так, чтобы он работал только с txt-файлами (\*.txt)  
Например, first.txt или name.1.part3.txt  
Если передан не txt-файл, например, file.txt.exe, то конструктор должен выбрасывать исключение UnsupportedFileNameException  
\*/  
  
public class TxtInputStream extends FileInputStream {  
 public TxtInputStream(String fileName) {  
 }  
  
}

package com.javarush.test.level18.lesson08.task05;  
  
import java.util.List;  
  
/\* Таблица  
Измените класс TableInterfaceWrapper так, чтобы он стал Wrapper-ом для ATableInterface.  
Метод setModel должен вывести в консоль количество элементов в новом листе перед обновлением модели  
Метод getHeaderText должен возвращать текст в верхнем регистре - используйте метод toUpperCase()  
\*/  
  
public class Solution {  
 public class TableInterfaceWrapper {  
  
 }  
  
 public interface ATableInterface {  
 void setModel(List rows);  
  
 String getHeaderText();  
  
 void setHeaderText(String newHeaderText);  
 }  
}

package com.javarush.test.level18.lesson10.home01;  
  
/\* Английские буквы  
В метод main первым параметром приходит имя файла.  
Посчитать количество букв английского алфавита, которое есть в этом файле.  
Вывести на экран число (количество букв)  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
}

package com.javarush.test.level18.lesson10.home02;  
  
/\* Пробелы  
В метод main первым параметром приходит имя файла.  
Вывести на экран соотношение количества пробелов к количеству всех символов. Например, 10.45  
1. Посчитать количество всех символов.  
2. Посчитать количество пробелов.  
3. Вывести на экран п2/п1\*100, округлив до 2 знаков после запятой  
4. Закрыть потоки. Не использовать try-with-resources  
\*/  
  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
 }  
}

package com.javarush.test.level18.lesson10.home03;  
  
/\* Два в одном  
Считать с консоли 3 имени файла  
Записать в первый файл содержимого второго файла, а потом дописать в первый файл содержимое третьего файла  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.test.level18.lesson10.home04;  
  
/\* Объединение файлов  
Считать с консоли 2 имени файла  
В начало первого файла записать содержимое второго файла так, чтобы получилось объединение файлов  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.test.level18.lesson10.home05;  
  
/\* Округление чисел  
Считать с консоли 2 имени файла  
Первый файл содержит вещественные(дробные) числа, разделенные пробелом. Например, 3.1415  
Округлить числа до целых и записать через пробел во второй файл  
Закрыть потоки. Не использовать try-with-resources  
Принцип округления:  
3.49 - 3  
3.50 - 4  
3.51 - 4  
-3.49 - -3  
-3.50 - -3  
-3.51 - -4  
\*/  
  
import java.io.\*;  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
 }  
}

package com.javarush.test.level18.lesson10.home06;  
  
/\* Встречаемость символов  
Программа запускается с одним параметром - именем файла, который содержит английский текст.  
Посчитать частоту встречания каждого символа.  
Отсортировать результат по возрастанию кода ASCII (почитать в инете). Пример: ','=44, 's'=115, 't'=116  
Вывести на консоль отсортированный результат:  
[символ1] частота1  
[символ2] частота2  
Закрыть потоки. Не использовать try-with-resources  
  
Пример вывода:  
, 19  
- 7  
f 361  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
   
 }  
}

package com.javarush.test.level18.lesson10.home07;  
  
/\* Поиск данных внутри файла  
Считать с консоли имя файла  
Найти в файле информацию, которая относится к заданному id,  
и вывести ее на экран в виде, в котором она записана в файле.  
Программа запускается с одним параметром: id (int)  
Закрыть потоки. Не использовать try-with-resources  
  
В файле данные разделены пробелом и хранятся в следующей последовательности:  
id productName price quantity  
  
где id - int  
productName - название товара, может содержать пробелы, String  
price - цена, double  
quantity - количество, int  
  
Информация по каждому товару хранится в отдельной строке  
\*/  
  
import java.io.\*;  
  
public class Solution {  
 public static void main(String[] args) throws IOException  
 {  
 }  
}

package com.javarush.test.level18.lesson10.home08;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Map;  
  
/\* Нити и байты  
Читайте с консоли имена файлов, пока не будет введено слово "exit"  
Передайте имя файла в нить ReadThread  
Нить ReadThread должна найти байт, который встречается в файле максимальное число раз, и добавить его в словарь resultMap,  
где параметр String - это имя файла, параметр Integer - это искомый байт.  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static Map<String, Integer> *resultMap* = new HashMap<String, Integer>();  
  
 public static void main(String[] args) throws IOException  
 {  
 }  
  
 public static class ReadThread extends Thread {  
 public ReadThread(String fileName) {  
 //implement constructor body  
 }  
 // implement file reading here - реализуйте чтение из файла тут  
 }  
}

package com.javarush.test.level18.lesson10.home09;  
  
/\* Файлы и исключения  
Читайте с консоли имена файлов  
Если файла не существует (передано неправильное имя файла), то  
перехватить исключение FileNotFoundException, вывести в консоль переданное неправильное имя файла и завершить работу программы.  
Закрыть потоки. Не использовать try-with-resources  
Не используйте System.exit();  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
}

package com.javarush.test.level18.lesson10.home10;  
  
/\* Собираем файл  
Собираем файл из кусочков  
Считывать с консоли имена файлов  
Каждый файл имеет имя: [someName].partN. Например, Lion.avi.part1, Lion.avi.part2, ..., Lion.avi.part37.  
Имена файлов подаются в произвольном порядке. Ввод заканчивается словом "end"  
В папке, где находятся все прочтенные файлы, создать файл без приставки [.partN]. Например, Lion.avi  
В него переписать все байты из файлов-частей используя буфер.  
Файлы переписывать в строгой последовательности, сначала первую часть, потом вторую, ..., в конце - последнюю.  
Закрыть потоки. Не использовать try-with-resources  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
}

package com.javarush.test.level18.lesson10.bonus01;  
  
/\* Шифровка  
Придумать механизм шифровки/дешифровки  
  
Программа запускается с одним из следующих наборов параметров:  
-e fileName fileOutputName  
-d fileName fileOutputName  
где  
fileName - имя файла, который необходимо зашифровать/расшифровать  
fileOutputName - имя файла, куда необходимо записать результат шифрования/дешифрования  
-e - ключ указывает, что необходимо зашифровать данные  
-d - ключ указывает, что необходимо расшифровать данные  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
  
}

package com.javarush.test.level18.lesson10.bonus02;  
  
/\* Прайсы  
CrUD для таблицы внутри файла  
Считать с консоли имя файла для операций CrUD  
Программа запускается со следующим набором параметров:  
-c productName price quantity  
Значения параметров:  
где id - 8 символов  
productName - название товара, 30 chars (60 bytes)  
price - цена, 8 символов  
quantity - количество, 4 символа  
-c - добавляет товар с заданными параметрами в конец файла, генерирует id самостоятельно, инкрементируя максимальный id, найденный в файле  
  
В файле данные хранятся в следующей последовательности (без разделяющих пробелов):  
id productName price quantity  
Данные дополнены пробелами до их длины  
  
Пример:  
19846 Шорты пляжные синие 159.00 12  
198478 Шорты пляжные черные с рисунко173.00 17  
19847983Куртка для сноубордистов, разм10173.991234  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws Exception {  
 }  
}

package com.javarush.test.level18.lesson10.bonus03;  
  
/\* Прайсы 2  
CrUD для таблицы внутри файла  
Считать с консоли имя файла для операций CrUD  
Программа запускается с одним из следующих наборов параметров:  
-u id productName price quantity  
-d id  
Значения параметров:  
где id - 8 символов  
productName - название товара, 30 chars (60 bytes)  
price - цена, 8 символов  
quantity - количество, 4 символа  
-u - обновляет данные товара с заданным id  
-d - производит физическое удаление товара с заданным id (все данные, которые относятся к переданному id)  
  
В файле данные хранятся в следующей последовательности (без разделяющих пробелов):  
id productName price quantity  
Данные дополнены пробелами до их длины  
  
Пример:  
19846 Шорты пляжные синие 159.00 12  
198478 Шорты пляжные черные с рисунко173.00 17  
19847983Куртка для сноубордистов, разм10173.991234  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
}