I/O, Exceptions and Strings

**Part 1: Representation**

**Create an object which will hold the files**

First you need to create a Person class with three fields:

* firstname of type String
* lastname of type String
* age of type int

Then create **one public constructor which takes the three parameters and correctly assign the variables to them**.

Finally, override the toString() method (which all objects inherit from Object). This is achieved by the function:

public String toString(){

<your code>

}

<your code> should return a string of the following format.

Name : <firstname> <lastname> Age: <age>

where the <bracketed> code refers to the value of the variables.

**Part 2 : Text input**

**Create a collection of Person objects**

Next you need to create a new file holding a collection of these objects. Create a class called Collection. This should contain a private array variable:

private Person[] people;

This array should hold 200 Person objects, so create a **constructor** for this class which set the array to hold 200 elements, i.e.:

People = new People[200];

**Reading from a file**

Next create a method

public void readFromFile(String filename){

<your code>

}

where <your code> fills the collection with Person objects that you create by reading from filename. You can assume that this file have 200 lines, where each line has the following structure:

<last-name> ; <first-name> ; <height> ; <age>

For each line you need to create a new Person object where the lastname field of the

object gets the value of <last-name>, the firstname field of the object gets the value of <first-name> and the age field get the value of <age>. You then need to insert this object in the collection at the next free place.

***Hint!*** *To achieve this you can use the FileReader and BufferedReader which you have learned about in the lecture. The String.split method (with “;” as separator) will also be useful to extract these elements (which returns a String array), as will the Integer wrapping class in order to convert the age to an integer. Also, remember to handle the exceptions...*

Finally, in the Collection class you need to create a main method:

public static void main(String [] args){

<your code>

}

which creates a Collection object and call readFromFile(“people.txt”.).

A file with sample data is provided to you . **Important: “people.txt” must be in the top level directory of the project you are working on!** Try to write some tests for your program (e.g. System.out.println(people[4] will print the 5th element of the array)

**Part 3: Re-representation**

**Last name in CAPITAL LETTER**

We want to store the last name so that all characters are in upper case. Update the constructor and set method of the Person class so that the input string for the last name is automatically converted into upper case.

***Hint!*** *Look online at the String class (you will find it by searching/Google for the String class in Java) which contains a very useful method.*

**Compute minimum, maximum and average age**

Write three methods:

public int minAge(){ <your code> }

public int maxAge() { <your code> }

public float avgAge() { <your code> }

which iterates/loops through the collection and returns the minimum, maximum and average (sum divided by number of elements) age. For the average, you will need to use a float rather than an integer.

**Compute minimum, maximum and average age FASTER**

The minAge, maxAge and avgAge can be made quicker by storing additional information in the collection. In fact, as you will learn later in the course, the methods are all of *linear* complexity, meaning that the execution time increases linearly with the size. We can make it *constant time*, meaning the execution time does not change, by adding three more variables:

int min; // the minimum age

int max; // the maximum age

int total; // the sum of all ages

Add these to your Collection class and give them appropriate initial values in the constructor. Then write three new functions which uses these variables to compute the minimum, maximum and average age using these variables without any loops (**Hint**! you need to update these while reading from file) :

public int fastMinAge(){ <your code> }

public int fastMaxAge() { <your code> }

public float fastAvgAge() { <your code> }

**Part 4: Text output**

Write a method

public void writeToFile(String filename){ <your code> }

in the Collection class which writes all the people of the collection to a file with the given filename. The file should have the following prelude:

Minimum age: <min age>

Maximum age: <max age>

Average age: <avg age>

followed by listing all the person using the format returned by the toString() method in the Person object. Update the main method so that it calls this file (with a filename of your choice).

***Hint!*** *Use the methods you have created to make the prelude of the file:*

*StringBuffer val = new StringBuffer (“Minimum age: ”);*

*val.append(fastMinAge());*

*val.append(“\n Maximum age: “)*

*...*

*and iterate the collection adding the result of the toString() method for each Person object to the string. You can then write this string to the file as shown in the lecture.*

**Part 5: Object IO**

We can also write objects to file directly. To do so, the object must be *serializable*, which in Java means that it must implement the Serializable interface (java.io.Serializable). First you need to update the Person class so that it implements this interface (and imports the java.io.Serializable).

Then add a method

public void writeObjects(){ <your code> }

which writes the collection array to a file (see lecture notes for details on how to do this). Next write a method

public void readObjects(){ <your code> }

which reads the collection and stores it in the collection variable. This must use the same filename as the writeObjects() method. Remember to cast the object, i.e. the actual code for reading when the ObjectInputStream ois is created should be:

collection = (Person []) ois.readObject();