**LINKS:**

[Enunciado](https://docs.google.com/document/d/16f90Js8HZH1-qw7-ArmLIIqXiD1WBDK7fXXymJnHEK8/edit)

[Diagrama de clases](https://lucid.app/invitations/accept/f4ac4c75-d857-4e51-8775-c5237f83cfa6)

**Engineering method report**

**1. Problem identification:**

It consists of the development of a software prototype that allows to efficiently manage CRUD operations on a database of people from our continent.

**Functional Requirements**

1. Implement a database of persons from the american continent
   1. Implement the CRUD operations
      1. Create
         1. persons records containing:
            1. id

The id must be auto-generated, therefore the user does not specify this element

* + - * 1. name

The names must be randomly generated from the following file [(LINK)](https://data.world/alexandra/baby-names)

* + - * 1. last name

The last names must be randomly generated from the following file [(LINK)](https://data.world/uscensusbureau/frequently-occurring-surnames-from-the-census-2010)

* + - * 1. sex

The sex must be randomly generated with values of:

male

female

* + - * 1. birthday

The birthday must be randomly generated, taking into account the distribution of ages from the United States [(LINK)](https://www.indexmundi.com/es/estados_unidos/distribucion_por_edad.html)

* + - * 1. height

The random height must be generated with a normal distribution

with a mean of 1.65 and a standard deviation of 0.05 for females

with a mean of 1.70 and standard deviation of 0.07 for males

* + - * 1. nationality

The nationality must be randomly generated taking into account the proportion of the population of each country [(LINK)](https://www.kaggle.com/tanuprabhu/population-by-country-2020)

* + - * 1. photography

The photography must be taken from the following website [(LINK)](https://thispersondoesnotexist.com/)

* + - 1. Indicate how long the operation took
         1. Represent the progress of the initialization process with a progress bar if the operation is taking longer than one second
      2. The option to generate must have a text field in which you can type how many records you want to generate.
         1. By default, the maximum possible value should be in the field [(LINK)](https://www.worldometers.info/geography/7-continents/)
    1. Read
       1. The search form must be able to search by any of the following criteria:
          1. First name
          2. Surname
          3. Full name
          4. Id
       2. The look-up must contain an auto-suggestions feature
          1. The program must allow that, as the characters are typed in the search field, a maximum of 100 (parameterizable) names in the database appear in a pop-up list below the field, starting with the characters typed up to the moment. This operation must be with the criteria:

First name

Surname

Full name

* + - * 1. While the search is being carried out, a number that indicates the total number of elements that so far match the prefix entered must be displayed next to the field where the string is entered
        2. As soon as there are 20 (parameterizable) or fewer elements that match the search, a list should be displayed with the matching records and a button next to each record with the Edit option, which will take us to the update form with the possibility to modify or delete that record
    1. Update
       1. The form to update a person must have all the fields of information of a person editable (except the Code), a
       2. nd the option to Update (to save the changes, if any)
    2. Delete
       1. The form to update a person must have the Delete option (if you want to delete this person)

1. The program must have a graphical interface with the user,
   1. have a menu component in the upper part of the window that allows changing all the elements of the window each time you are working on different options.
      1. That is, you do not want to have a window full at the same time with elements that will be used at different times.

**Non-Functional Requirements**

1. The data sets used for the the initialization of the database must be store in the (/data) package
2. All the data of the program must be persistent
3. For each of the four searches, it must maintain a persistent, self-balancing binary search tree.
   1. The tree must be fully implemented by your development team.
      1. The self-balancing binary search tree is a Red and Black Tree.

Responda este par de preguntas: (1) ¿Es posible serializar los datos generados? (2) ¿Cuánto pesa el archivo serializado, cuando se persisten los datos, al generarse el número máximo posible de registros?

**2. Information gathering:**

The population of the American continent is estimated, in 2020, at just over a billion people, representing about 13% of the world total.

For the generation of full names, the names must be taken from this [name database from data.world](https://data.world/alexandra/baby-names). The surnames must be taken from the entire file of this [surname database from data.world](https://data.world/uscensusbureau/frequently-occurring-surnames-from-the-census-2010).

The date of birth should be randomly generated, assuming an age distribution for all of America based on this [age distribution for the United States](https://www.indexmundi.com/es/estados_unidos/distribucion_por_edad.html). For the sex distribution an equal number of men and women can be assumed.

Height should be randomly generated in an interval that makes sense. Nationality must be assigned to each person, generated in such a way that it maintains the relative percentages of the population of each country with respect to the continent according to these [population data by countries](https://www.kaggle.com/tanuprabhu/population-by-country-2020).

Datasets can be filtered to leave only the necessary country records on file.

The photography must be randomly generated from this site: <https://thispersondoesnotexist.com/> regardless of whether the image does not exactly correspond to the person's age, gender or others.

**CRUD operations**

In programming, the four basic functions of persistent storage are known as CRUD (Create, Read, Update and Delete). The program must be able to perform any of these functions. Therefore, the user interface must have a form to add a new person, another to search for a person by the criteria mentioned above and one more to update the fields of an existing person. This last form should also have an option to remove a person.

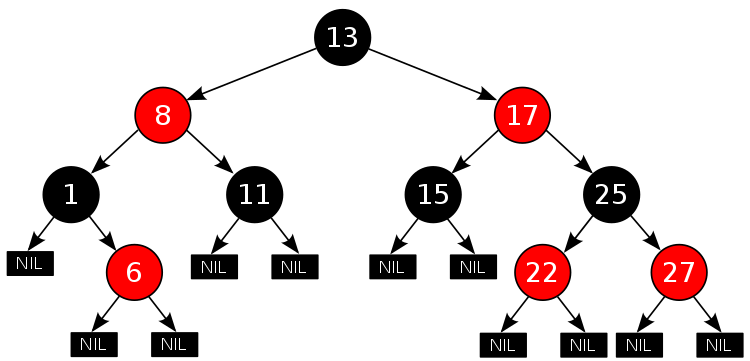
**Red-Black tree**

Red-Black tree, in computer science, is a kind of self-balancing binary search tree. Each node stores an extra bit representing color, used to ensure that the tree remains approximately balanced during insertions and deletions.

**Properties of Red-Black tree**

In addition to the requirements imposed on a [binary search tree](https://en.wikipedia.org/wiki/Binary_search_tree) the following must be satisfied by a red–black tree:

1. Each node is either red or black.
2. The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.
3. All leaves (NIL) are black.
4. If a node is red, then both its children are black.
5. Every [path](https://en.wikipedia.org/wiki/Path_(graph_theory)) from a given node to any of its descendant NIL nodes goes through the same number of black nodes.



**Persistence:** Serialization vs Text files

SEARCH

If two different objects have the same reference to another object. will the object be duplicated while serializing?

API documentation for [ObjectOutputStream](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/ObjectOutputStream.html):

(...) Multiple references to a single object are encoded using a reference sharing mechanism so that graphs of objects can be restored to the same shape as when the original was written. (...)

https://stackoverflow.com/questions/20076784/java-serialization-and-duplicate-objects

**3. Search for creative solutions:**

**People storage:**

1. 4 trees - 4 hashmaps (key: name, surname, code) and the value points to the index in the ArrayList
   1. ArrayList with initial capacity (13% world population)
2. 4 trees (key: name, surname, etc) and the value points to the index in the ArrayList
   1. ArrayList with initial capacity (13% world population)
3. 4 trees (key: name, surname, etc) and the value is a reference to the object
4. 2 tree (key: full name, id) and the value is a reference to the object.

**Persistence:**

1. Text files
2. Serialization

**Auto-suggestions:**

1. Read the keys that are being typed in the keyboard y make a search

**4. Transition from ideas to preliminary designs**

From people storage ideas we discarded the 1 and 2 because they used too much memory and were inefficient in runtime, and we left 3 and 4 as the possible solutions.

**5. Evaluation and selection of the best solution**

We discarded the idea 3, because it led us to an OutOfMemoryError, and decided to go for the 4 option because it’s the most efficient in runtime and space memory usage.

For persistence we decided to go for the serialization, as it is easier and better than text files. The main benefit of serialization in java is that you don't have to manually do the reading/writing into files.

**TestAVLBSTree**

**Stages configuration:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Stage** |
| emptySetup | TestAVLBSTree | An object of the AVLBSTree class empty. |
| nonEmptySetup | TestAVLBSTree | An object of the AVLBSTree class with the nodes 5, 8, 17, 25 and 32 already added. |

**Test cases design:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly adds a new node to an empty AVL tree. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| AVLBSTree | add | emptySetup | 1, 1 | A new node with the given key and value has been correctly added. |
| AVLBSTree | add | emptySetup | 2, 2 | A new node with the given key and value has been correctly added. |
| AVLBSTree | add | emptySetup | 3, 3 | A new node with the given key and value has been correctly added. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly adds a new node to a non-empty AVL tree. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| AVLBSTree | add | nonEmptySetup | 12, 12 | A new node with the given key and value has been correctly added. |
| AVLBSTree | add | nonEmptySetup | 40, 40 | A new node with the given key and value has been correctly added. |
| AVLBSTree | add | nonEmptySetup | 20, 20 | A new node with the given key and value has been correctly added. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly deletes a node with a given key from an empty AVL tree. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| AVLBSTree | delete | emptySetup | 1 | The node with the given key has been deleted. |
| AVLBSTree | delete | emptySetup | 38 | The node with the given key has been deleted. |
| AVLBSTree | delete | emptySetup | 25 | The node with the given key has been deleted. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly deletes a node with a given key from a non-empty AVL tree. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| AVLBSTree | delete | nonEmptySetup | 8 | The node with the given key has been deleted. |
| AVLBSTree | delete | nonEmptySetup | 5 | The node with the given key has been deleted. |
| AVLBSTree | delete | nonEmptySetup | 32 | The node with the given key has been deleted. |

**TestBinarySearchTree**

**Stages configuration:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Stage** |
| nonEmptySetup | TestBinarySearchTree | An object of the BinarySearchTree class with the nodes 8, 15, 17, 20, 23, 28 and 32 already added. |

**Test cases design:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly applies rotateLeft to a given node of the BST. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| BinarySearchTree | rotateLeft | nonEmptySetup | BST.root.left | The given node is rotated to the left. |
| BinarySearchTree | rotateLeft | nonEmptySetup | BST.root | The given node is rotated to the left. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test that the class correctly applies rotateRight to a given node of the BST. | | | | |
| **Class** | **Method** | **Setup** | **Input** | **Result** |
| BinarySearchTree | rotateRight | nonEmptySetup | BST.root.right | The given node is rotated to the right. |
| BinarySearchTree | rotateRight | nonEmptySetup | BST.root | The given node is rotated to the right. |