Marco Fidel Vasquez - A00362057
David Steven Montoya - A00362450
Camilo Escobar Arteaga - A00358632

REPORT OF ENGINEERING METHOD

PROBLEM IDENTIFICATION

Problem Necessities

- Generate registries about the inhabitants with the following data: An auto generated code, name, last name, gender, birthdate, height(it should be generated randomly within an interval which must be reasonable), nationality and a picture.
- The software should take names from this dataset for the registry. It should also take information from this dataset to generate the respective last name.
- The age distribution must be randomly generated based on the United States official distribution.
- The nationality must be generated in a way that keeps the percentages taken from this dataset.
- The pictures should be taken from https://thispersondoesnotexist.com/ and assigned to each client. Gender correspondence can be ignored.
- All the generated registries can be saved in some place where they can be accessed on further use.
- The database should be able to do CRUD operations (Create, Read, Update, Delete).

Problem Definition

- The VIP Simulation team requires a software capable of efficiently processing CRUD (Create, Read, Update & Delete) operations in a database consisting of inhabitants of the American Continent. They require the use of the following datasets to generate the inhabitants, first we must use this dataset to generate the names and this other dataset for the last names. Additionally, it should use this United States age distribution to randomly generate a birthdate within those parameters, and this data of population by country to spread out the population consistently.

Functional Requirements:

FR1: Generate an instance of a person with the following attributes: a randomly generated code, a name, last name, gender, birthdate, height, nationality and a picture and save it to the database.

FR2: Use the various datasets to generate a random name, last name, gender distribution, age distribution (birthdate), height distribution, nationality and picture.

FR3: Have a text field where the user can input the amount of instances to be created, by default this text field should be set to the maximum.

FR4: Show a progress bar if the previous process takes more than 1 second to be completed and show the amount of time taken to register all of the given instances.

FR5: Search the database with the following criteria: Name, last name, full name, code. While the user is inputting the data, if it uses any of the first 3 criteria, the program should find the first 100 (parameterizable) closests names in the database. While the search is being done, to the right of the text field where the user does the search the number of close coincidences should be shown, also, when there are 20(parameterizable) or less coincidences, each name in the list should have a button to its right that loads the data to the interface, should it be clicked.

FR6: Update any camp except code from an element.

FR7: Delete an element from the database.

Non Functional Requirements:

NFR1: The structures to save the users into different ordering criteria will be an AVL Binary search tree.

NFR2: The database will be saved using Serializable.

COMPILATION OF INFORMATION

Definitions

Dataset: A collection of separate sets of information that is treated as a single unit by a computer.

Database: An organized collection of structured information, or data, typically stored electronically in a computer system.[1]

CRUD operations: CRUD refers to create, read, update and delete operations, which are defined down here.[2]

CREATE procedures: Performs the INSERT statement to create a new record.

READ procedures: Reads the table records based on the primary key noted within the input parameter.

UPDATE procedures: Executes an UPDATE statement on the table based on the specified primary key for a record within the WHERE clause of the statement.

---

[1] https://www.oracle.com/database/what-is-database.html
[2] https://stackify.com/what-are-crud-operations/

DELETE procedures: Deletes a specified row in the WHERE clause.

TRIE: A trie is a data structure in the tree type that allows information recovery (from english reTRIEval).

POSSIBLE SOLUTIONS

Idea generation methodology used: Brainstorming.

Possible solutions to show results to the user while they digit the information:

Solution 1: Create a thread which will constantly search for people every second while the user digits the information.

Solution 2: Create a thread which will search for people as the user digits or deletes characters.

Solution 3: Create a method that uses javaFx textField's onKeyPressed that calls the search method every time the user presses a key.

Solution 4: Create a method that uses javaFx textField's onKeyTyped that calls the search method every time the user types a key.

Solution 5: Create a method that uses javaFx textField's onKeyReleased that calls the search method every time the user releases a key.

Possible solutions for the database:

Solution 1: Use various Binary Search Trees to store people, with a different tree for each aspect to search for.

Solution 2: Use various HashTables to store people, with a different HashCode Key for each aspect to search for.

Solution 3: Use an AVL Binary Search Tree to store the people, using a different tree for each aspect to search for.

Solution 4: Use an AVL Binary Search Tree to store the people, using a different tree for each aspect to search for. Additionally, use an red-black BST for one of the aspects to search for.

TRANSFORMING IDEA FORMULATION TO PRELIMINARY DESIGNS

Description solution 1: Create a thread which will constantly search for people every second while the user digits the information. It searches in the tree every second without regard to whether more characters are being written or not.

Description solution 2: Create a thread which will search for people as the user digits or deletes characters. It searches in the tree every time a character is deleted or added to the textfield and shows the results.

Description solution 3: Create a method that uses JavaFx's textField's onKeyPressed, making it call the search method every time the user types a letter or a number.

Description solution 4: Create a method that uses JavaFx's textField's onKeyTyped, making it call the search method every time the user types a letter or a number.

Description solution 5: Create a method that uses JavaFx's textField's onKeyTyped, making it call the search method every time the user types a letter or a number and lets go of it.

DISCARDED IDEAS

Show results to the user while they digit the information

Solution 5: Was discarded because of its likeness to solution 4, thus prompting a choice between both of them, were we decided to pick the use of the method onKeyTyped.

Database storing

Solution 2: Was discarded because of the inefficiency in regards to the worst case scenario search and also because a HashTable, to search properly, requires the use of a predetermined "full" key and thus will not work when trying to continually show the results to the user from an incomplete letter/number String.

EVALUATION AND SELECTION OF THE BEST SOLUTION

solution	knowledge about the topic	Value for the client	Ease of development	Flexibility	Total	Approved/ Not approved
Possible solutions to show results to the user while they digit the information						
solution 1	5	2	5	1	13	Not Approved
solution 2	5	4	5	5	19	Approved
solution 3	4	4	5	4	17	Not Approved
solution 4	4	5	5	4	18	Not Approved
Possible solutions for the database						
solution 1	5	3	5	3	16	Not Approved
solution 3	4	5	5	5	19	Not Approved
solution 4	3	4	4	5	16	Approved

Knowledge about the topic: Our theoretical knowledge about the topic required to implement this solution in the program.

Value for the client: The value that the implementation of this solution that will visually and performance-wise give to the client, such as: Faster recognition, ease of use and resource allocation.

Ease of development: How easy and/or fast it is to develop the solution in code.

Flexibility: How flexible the code is in regards to future-proofing and fixing any errors that may arise in its use.

Solution chosen for showing the results to the user while they digit the information.

We have chosen solution 2 for this problem. The main reason for this is because of the possible problems that may arise in doing operations in the JavaFx's thread, which is what solution 3 and 4 go for and discarding solution 1 because it is incredibly inefficient.

Solution chosen for the database.

We have chosen solution 4 for this problem considering that the tree structures are the most efficient to tackle CRUD operations that we know of. We discarded solution 1 considering the fact that the tree may become unbalanced and end up with efficiency problems. We also discarded solution 3 because we consider that the implementation of a red-black tree in one of our structures will both help with insertions and deletions and also allow us to gather more information in how this structure operates in the contexts of very big amounts of data.