Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesora: Rocío Alejandra Aldeco Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 6

No de Práctica(s): 2

Integrante(s): Ugalde Velasco Armando

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: 2021-1

Fecha de entrega: 9 de octubre de 2020

Observaciones:

CALIFICACIÓN: _____

Práctica de Estudio 2: Fundamentos y sintaxis del lenguaje
Programación Orientada a Objetos Grupo 6
Facultad de Ingeniería
Departamento de Computación

Objetivo de la práctica: Crear programas que implementen variables y constantes de diferentes tipos de datos, expresiones y estructuras de control de flujo

Realiza las siguientes actividades después de leer y revisar en clase la **Práctica de Estudio 2: Fundamentos y sintaxis del lenguaje**.

1. Diseña una clase que contenga los atributos:

- Nombre
- Apellido Paterno
- Apellido Materno
- Fecha de Nacimiento

Y los métodos:

- **nombreCompleto**: Genera el nombre completo de una persona
- **calcularEdad**: Dada la fecha de nacimiento calcula la edad en años
- **calcularRFC**: Dados todos los atributos calcula el **RFC**

Las fechas pueden usar la clase **Date** y **DateFormat**. El **RFC** se calcula siguiendo el procedimiento descrito en la referencia.

```java
public class Persona
{
    private static final Set<String> NOMBRES_PROHIBIDOS;
    private static final Set<String> APELLIDOS_PROHIBIDOS;
    private static final Set<Character> VOCALES;

    static
    {
        APELLIDOS_PROHIBIDOS = Set.of("DA", "DAS", "DE", "DEL", "DER", "DI", "DIE", "DD", "EL", "LA", "LOS", "LAS
        NOMBRES_PROHIBIDOS = new HashSet<>();
        NOMBRES_PROHIBIDOS.addAll(APELLIDOS_PROHIBIDOS);
        NOMBRES_PROHIBIDOS.addAll(Set.of("MARIA", "MA.", "MA", "JOSE", "J", "J."));
        VOCALES = Set.of('A', 'E', 'I', 'O', 'U');
    }

    private final String nombre;
    private final String aPaterno;
    private final String aMaterno;
    private final LocalDate nacimiento;
```

Atributos de la clase Persona

Se creó la clase **Persona** con los atributos respectivos constantes: **nombre**, **aPaterno** y, **aMaterno**, de tipo **String**; y **nacimiento**, de tipo **LocalDate**. Además, se declararon e inicializaron de forma estática los atributos constantes que contendrían los conjuntos de **nombres y apellidos prohibidos**, además de las **vocales**.

En el constructor, se consideraron cuatro parámetros de tipo **String** con la información necesaria para inicializar las variables de instancia respectivas. En el caso de la fecha, se utilizó el método estático **parse** de la clase **LocalDate**, para crear una instancia de ésta última a partir de la cadena proporcionada, utilizando además el método estático **ofPattern** de la clase **DateTimeFormatter** para indicar el patrón de la cadena.

Cabe mencionar que, debido a que la mayor parte de métodos presentes en la clase **Date** se encuentran obsoletos, se decidió utilizar la clase antes mencionada: **LocalDate**.

```java
public Persona(String nombre, String aPaterno, String materno, String fecha)
{
    this.nombre = nombre;
    this.aPaterno = aPaterno;
    this.aMaterno = materno;
    this.nacimiento = LocalDate.parse(fecha, DateTimeFormatter.ofPattern("dd/MM/yyyy"));
}
```

Costructor de la clase Persona

2. Explica el funcionamiento de los métodos **calcularEdad** y **calcularRFC** mostrando el código correspondiente.

```java
public int calcularEdad()
{
    return LocalDate.now().getYear() - this.nacimiento.getYear();
}
```

Método calcularEdad

Como se logra observar en la captura de pantalla, la implementación del método **calcularEdad** fue muy sencilla: únicamente se retornó la diferencia entre los años de la fecha actual y la fecha de nacimiento de la persona, utilizando **LocalDate.now()** para obtener la fecha actual, y el método **getYear()** para obtener el año correspondiente.

```
public String calcularRFC() throws Exception
{
    String cleanName = this.getCleanName(this.nombre, Persona.NOMBRES_PROHIBIDOS);
    String cleanFirstLastName = this.getCleanName(this.aPaterno, Persona.APELLIDOS_PROHIBIDOS);
    String cleanSecondLastName = this.getCleanName(this.aMaterno, Persona.APELLIDOS_PROHIBIDOS);
    String dateInfo = this.nacimiento.format(DateTimeFormatter.ofPattern("yyMMdd"));

    return cleanFirstLastName.substring(0, 1)
            + this.findFirstInternalVowel(cleanFirstLastName)
            + cleanSecondLastName.charAt(0)
            + cleanName.charAt(0)
            + dateInfo;
}
```

Método calcularRFC

La implementación del método **calcularRFC** se realizó utilizando los métodos auxiliares privados **getCleanName**, que retorna la cadena proporcionada en mayúsculas, sin acentos, y siguiendo las reglas de excepciones acorde al conjunto proporcionado; y **findFirstInternalVowel**, que retorna la primera vocal interna en la cadena proporcionada.

Asumiendo que su comportamiento es el esperado, concluimos que las primeras líneas obtienen los nombres y apellidos **“limpios”** de las personas, es decir, en mayúsculas y siguiendo las reglas correspondientes del **RFC**. También se obtiene la cadena correspondiente a la fecha en el **RFC**, utilizando el método **format** de la clase **LocalDate**, y proporcionando el patrón deseado.

Finalmente, tomando en cuenta que los nombres y apellidos ya se encuentran con la información sanitizada de excepciones, se retorna la cadena concatenada correspondiente al RFC, compuesto por:

- Una cadena compuesta por la primera letra del apellido paterno. Se utiliza el método **substring** para generar una cadena y que sea posible concatenar caracteres después, provocando una conversión de estos últimos a **String**.

- La primera vocal interna del apellido paterno, utilizando el método privado antes mencionado: **findFirstInternalVowel.**
- El primer carácter del apellido materno, utilizando el método **charAt**.
- El primer carácter del nombre, utilizando el método **charAt.**
- La información correspondiente a la fecha, ya obtenida en la variable **dateInfo.**

A continuación, se explicarán con mayor detalle los métodos auxiliares utilizados:

```
private char findFirstInternalVowel(String str) throws Exception
{
    char[] internalStr = str.substring(1, str.length() - 1).toCharArray();
    for (char c : internalStr)
    {
        if (Persona.VOCALES.contains(c)) return c;
    }
    throw new Exception("La cadena no contiene ninguna vocal interna");
}
```

Método findFirstInternalVowel

En este método, primero se crea un arreglo con los caracteres internos de la cadena proporcionada, utilizando los métodos **substring** y **toCharArray**. Finalmente, se itera sobre éste y, si el carácter analizado se encuentra dentro del conjunto **VOCALES**, éste se retorna. Si el ciclo finaliza, quiere decir que la cadena no contenía ninguna vocal, por lo tanto, se lanza una excepción.

```java
private String getCleanName(String name, Set<String> forbiddenNames)
{
    // Quitar acentos y transformar a mayúsculas
    name = Normalizer.normalize(name.toUpperCase(), Normalizer.Form.NFD).replaceAll( regex: "\\p{M}",  replacement: "");

    String[] names = name.split( regex: " ");
    // Si tiene un solo nombre, retornarlo
    if (names.length == 1) return names[0];

    // Limpiar nombre
    String cleanName = Arrays.stream(names)
            .filter(Predicate.not(forbiddenNames::contains))
            .reduce( identity: "", (acc, part) -> acc + " " + part)
            .strip();

    // Si el nombre está compuesto de excepciones, retornar el primero
    if (cleanName.length() == 0) return names[0];
    else return cleanName;
}
```

Método getCleanName

En este método, primero se "*transforma"* el nombre proporcionado a una cadena sin acentos y en mayúsculas, utilizando el método **normalizer**, que "separa" las letras de sus acentos, y el método **replaceAll**, que toma como entrada una expresión regular que identifica los acentos y los reemplaza con un carácter vacío, en este caso. Ambos métodos forman parte de la clase **Normalizer**. Para transformar la cadena a mayúsculas se utilizó el método **toUpperCase**, de la clase **String.**

Posteriormente, se generó un arreglo de cadenas para obtener las partes de los nombres compuestos, utilizando el método **split**. En este punto, si la longitud del arreglo es **1**, quiere decir que el nombre es simple, y, por lo tanto, se retorna. Después, con ayuda de las utilidades funcionales **filter** y **reduce**, se eliminaron las cadenas prohibidas presentes en la lista proporcionada, y se unieron las partes restantes en una cadena.

Finalmente, se consideró el caso de un nombre compuesto únicamente con cadenas prohibidas, implicando el hecho de que, en este punto, la cadena se encontraría vacía, por lo tanto, se retorna el primer nombre. En caso contrario, se retorna el nuevo nombre libre de palabras prohibidas, considerando la mayor parte de las excepciones.

CONCLUSIONES

El paradigma principal del lenguaje **Java**, como ya se mencionó en clase y en la práctica anterior, es el orientado a objetos, a diferencia del paradigma procedural presente en el lenguaje **C**. Gracias a lo anterior, en **Java** existe una gran cantidad de constructos inexistentes en el lenguaje **C** que cumplen distintos objetivos. En la mayoría de los casos, dichos constructos principalmente se encuentran relacionados a los objetos. Dos claros ejemplos de lo anterior son la palabra reservada **class** y los modificadores de acceso.

A pesar de lo anterior, gran parte de la sintaxis en ambos lenguajes coincide, principalmente en las cuestiones no relacionadas al paradigma orientado a objetos. La división de los bloques de código con corchetes, el tipado de variables y las estructuras **if-else** son claros ejemplos de lo anterior.

Uno de los puntos más importantes y útiles en el lenguaje **Java** es la extensa funcionalidad proporcionada por las distintas **APIs**. A diferencia del lenguaje **C**, en Java la forma de modularizar funcionalidad y conceptos relacionados es mediante clases. En el programa realizado en la práctica, se utilizaron **APIs** como **LocalDate** y **Normalizer**, que, sin duda alguna, facilitaron en gran parte el proceso de desarrollo al proporcionar las funcionalidades pertinentes *“fuera de la caja”.* Al utilizar dichas clases, se crearon instancias de estas, o bien, se utilizaron métodos estáticos, contrastando con el funcionamiento de las librerías en el lenguaje C.