Examen Backend Nombre: Martin Aldair Ramírez Ramos Fecha: 7/Enero/2022

1. Supone que debes establecer una contraseña para una cuenta bancaria. Sin embargo, hay tres restricciones en el formato de la contraseña:

a) Debe contener solo caracteres alfanuméricos (a−z, A−Z, 0−9);

b) Debe haber un número par de letras;

c) Debe haber un número impar de dígitos.

Como input debe recibir una cadena S de N caracteres. La cadena S se puede dividir en palabras dividiéndola y eliminando los espacios. El objetivo es elegir la palabra más larga que sea una contraseña válida. Puede suponer que, si hay K espacios en la cadena S, entonces hay exactamente K + 1 palabras.

Por ejemplo, dado "test 5 a0A pass007 ?xy1", hay cinco palabras y tres de ellas son contraseñas válidas: "5", "a0A" y "pass007". Por lo tanto, la contraseña más larga es "pass007" y su longitud es 7.

**Genera una clase que, dada una cadena S no vacía que consta de N caracteres, devuelve la longitud de la palabra más larga de la cadena que es una contraseña válida. Si no existe tal palabra, su función debería devolver −1, como se explicó anteriormente.**

Asumir que:

N es un número entero dentro del rango [1..200];

La cadena S consta únicamente de caracteres y espacios ASCII imprimibles.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Ingresa la cadena S: ");  
 String s = scanner.nextLine();  
 scanner.close();  
  
 PasswordMasLargaPermitida passwordPermitido = new PasswordMasLargaPermitida();  
 int result = passwordPermitido.logitudPassworLargo(s);  
 System.*out*.println(result);  
 }  
}

public class PasswordMasLargaPermitida {  
 public int logitudPassworLargo(String s) {  
 String[] aregloDePasswords = s.split(" ");  
 int tamañoMaximo = -1;  
  
 for (String password : aregloDePasswords) {  
 if (validarPasswords(password)) {  
 tamañoMaximo = Math.*max*(tamañoMaximo, password.length());  
 }  
 }  
  
 return tamañoMaximo;  
 }  
 private boolean validarPasswords(String password) {  
  
 int numLetras = 0;  
 int numDigitos = 0;  
 for (char c : password.toCharArray()) {  
 if (Character.*isLetter*(c)) {  
 numLetras++;  
 } else if (Character.*isDigit*(c)) {  
 numDigitos++;  
 }  
 }  
 return numLetras % 2 == 0 && numDigitos % 2 != 0;  
 }  
  
 }

1. **Manejo de Errores**
   1. Dado la siguiente clase agrega el manejo de errores correspondientes y justifica él porque

|  |
| --- |
| public class Examen {  public static void main (String [] args) {  System.out.println("Intentamos ejecutar el bloque de instrucciones:");  System.out.println("Instrucción 1.");  int n = Integer.parseInt("M"); //error forzado en tiempo de ejecución.  System.out.println("Instrucción 2.");  System.out.println("Instrucción 3, etc.");  }  }  En mi Respuseta ejecute el siguiente código ya que cuando intente correr el programa con el error me arrogo una excepción NumberFormatExeption por eso el uso del try catch manejando esa excepción. |

public class Examen {  
 public static void main(String[] args) {  
 System.*out*.println("Intentamos ejecutar el bloque de instrucciones:");  
 System.*out*.println("Instrucción 1.");  
 try {  
 int n = Integer.*parseInt*("M"); // error forzado en tiempo de ejecución.  
 } catch (NumberFormatException e) {  
 System.*out*.println("Ocurrió un error al intentar convertir la cadena en un entero.");  
 }  
 System.*out*.println("Instrucción 2.");  
 System.*out*.println("Instrucción 3, etc.");  
  
 }  
}

* 1. Dado el siguiente método, agregar el código necesario para que el método genere una excepción personalizada en caso de error.

|  |
| --- |
| **public** **static** Date getFecha(String fecha, String formato) {  SimpleDateFormat formatter = **new** SimpleDateFormat(formato);  String dateInString = fecha;  Date fechaSalida = **null**;  **try** {  fechaSalida = formatter.parse(dateInString);  } **catch** (ParseException e) {  ***LOGGER***.error("Error" + e);  }    **return** fechaSalida;  } |

En este caso hice unos cambios porque no conosco LOWER Y hice uso de los throws

public class Main {  
 public static void main(String[] args) {  
 try {  
 Date fecha = *getFecha*("2023-08-15", "-MM-dd");  
 System.*out*.println("Fecha parseada: " + fecha);  
 } catch (Exception e) {  
 System.*err*.println("Error al obtener la fecha: " + e.getMessage());  
 }  
 }  
  
 public static Date getFecha(String fecha, String formato) throws Exception {  
 SimpleDateFormat formatter = new SimpleDateFormat(formato);  
 String dateInString = fecha;  
 Date fechaSalida = null;  
  
 try {  
 fechaSalida = formatter.parse(dateInString);  
 } catch (ParseException e) {  
 throw new Exception("Error al parsear la fecha: " + e.getMessage());  
 }  
  
 return fechaSalida;  
 }  
}

1. Que es la inyección de dependencias.

Es cuando a un objeto le agregamos instancias necesarias para su funcionamiento en lugar que el objeto lascree

Como el uso de constructores o interfaces.

En este microservicio puede aser uso de swager ya esta la dependencia agregada en el archivo pom.xml

1. Genera un microservicio utilizando el framework de spring boot y compartir el código, considere los siguientes puntos.
   1. El servicio debe tener 3 capas: Controller, Service, DAO (@Controller, @Service, @Repository)
   2. Genera un Modelo que contenga las propiedades para identificar a una persona, ejemplo: id, nombre, apellidos, rfc, curp, edad, sexo, nacionalidad.
   3. Generar un endpoint REST que reciba como parámetro el id, el parámetro es opcional.
      1. El path sería /personas
   4. Generar la lógica correspondiente para que al invocar el endpoint se recupere un listado de personas en caso de que el id no venga informado. En caso de informar el id solo debe retornar la persona con el id correspondiente.
   5. La recuperación de la información debe de utilizar las 3 capas (Controller, Service, DAO), siendo la capa DAO donde se simule la recuperación de la información.
   6. La inyección de dependencias de ser por medio de interfaces
   7. Documentar el código
   8. Separar las clases en package nombrados de acuerdo a su funcionalidad, por ejemplo:

mx.examen.service -> contiene solo clases Service

mx.examen.model -> contiene solo clases entidades TO

* 1. PUNTO ADICIONAL INCLUIR JUNIT
  2. PUNTO ADICIONAL INCLUIR Llamado a un API Publica de libre elección