Evaluación Diagnóstica.

Para esta actividad deberás enviar un documento PDF, en el cuál, se conteste la presente evaluación diagnóstica. Recuerda que esta actividad solamente determina el nivel de conocimiento que tienes al inicio de la materia, por lo tanto, representa sólo un diagnóstico, sin embargo, es una actividad obligatoria para iniciar el cuatrimestre.

Conteste las siguientes preguntas, para el ámbito de la programación de software y el paradigma de la Programación Orientada a Objetos (POO):

1. **¿Qué es un error?**

- Un error es un problema en el código de un programa que impide su ejecución normal.

2. **¿Qué es una excepción?**

- Una excepción es un evento inusual o error que ocurre durante la ejecución de un programa y puede ser manejado para evitar que el programa se bloquee.

3. **¿Cuáles son las características de los errores?**

- Los errores son problemas en el código que causan fallos en la ejecución del programa. Pueden ser de sintaxis, lógicos o de tiempo de ejecución.

4. **¿Cuáles son las características de las excepciones?**

- Las excepciones son eventos inesperados durante la ejecución del programa. Pueden ser manejadas mediante bloques try-except para evitar que el programa se detenga.

5. **¿Cuáles son los tipos de errores?**

- Errores de sintaxis, errores de lógica y errores de tiempo de ejecución.

6. **¿Cuáles son las jerarquías de las excepciones?**

- Las excepciones en muchos lenguajes de programación tienen una jerarquía de clases, donde las excepciones más generales están en la parte superior y las más específicas están en la parte inferior.

7. **¿Cuáles son las excepciones implícitas?**

- Las excepciones implícitas son generadas automáticamente por el sistema cuando ocurren errores, como DivisionByZeroError.

8. **¿Cuáles son las excepciones explícitas?**

- Las excepciones explícitas son creadas por el programador para manejar situaciones específicas.

9. **¿Cuál es la relación de los tipos de errores y excepciones que se presentan en la ejecución de un programa?**

- Los errores pueden desencadenar excepciones. Las excepciones son manejadas para evitar que los errores interrumpan la ejecución del programa.

10. **En las excepciones, ¿qué es un bloque de manejo?**

- Un bloque de manejo es una sección de código que se ejecuta cuando se captura una excepción. Generalmente se usa try-except para definirlo.

11. **En las excepciones, ¿qué significa la propagación?**

- La propagación de una excepción es el proceso de pasar la excepción de un bloque de manejo a otro o, en última instancia, al sistema si no se maneja.

12. **En las excepciones, ¿qué significa la captura?**

- La captura es el proceso de atrapar y manejar una excepción utilizando un bloque try-except.

13. **En las excepciones, ¿qué significa el análisis de la pila de errores?**

- El análisis de la pila de errores implica rastrear la secuencia de llamadas de funciones para encontrar dónde ocurrió la excepción.

14. **¿Cómo se crea una excepción?**

- Se crea una excepción personalizada mediante la definición de una nueva clase que hereda de Exception o una clase relacionada.

15. **¿Cómo se lanza una excepción?**

- Una excepción se lanza utilizando la palabra clave "raise" seguida del nombre de la excepción.

16. **En la POO, ¿Qué es una GUI?**

- Una GUI (Interfaz Gráfica de Usuario) es una representación visual de un programa que permite a los usuarios interactuar con él a través de elementos gráficos como botones y ventanas.

17. **¿Qué es un componente en una GUI?**

- Un componente en una GUI es un elemento gráfico, como un botón o una etiqueta, que se utiliza para construir la interfaz de usuario.

18. **Cuáles son las características de los componentes en una GUI?**

- Los componentes tienen propiedades visuales y funcionales, como texto, color, eventos y comportamientos específicos.

19. **¿Qué es la jerarquía de componentes de una GUI?**

- La jerarquía de componentes es la estructura en árbol que describe cómo se organizan y agrupan los componentes en una GUI.

20. **¿Cuál es el proceso de diseño de una GUI?**

- El proceso de diseño de una GUI implica planificar la disposición de los componentes, definir su apariencia y comportamiento, y crear un diseño visual atractivo.

21. **¿Cuál es el proceso de codificación de componentes de GUI?**

- El proceso de codificación de componentes de GUI implica crear y configurar objetos gráficos, establecer eventos y diseñar la interacción con el usuario.

22. **¿Qué es una librería?**

- Una librería es un conjunto de funciones y clases predefinidas que se pueden usar en un programa para realizar tareas específicas.

23. **¿Cuál librería se ocupa en Java para el manejo de objetos gráficos?**

- En Java, se utiliza la librería Swing para el manejo de objetos gráficos y la creación de GUI.

24. **Explica el proceso de implementación de librerías de objetos gráficos (letras, colores, dibujo de formas)**

- La implementación de librerías de objetos gráficos implica importar las clases y métodos necesarios, configurar objetos gráficos como ventanas y componentes, establecer propiedades como colores y fuentes, y definir la lógica de dibujo y manipulación de objetos.

25. **¿Qué es un evento?**

- Un evento es una acción o suceso que ocurre en una aplicación, como hacer clic en un botón o presionar una tecla.

26. **¿Cuáles son las características de los eventos?**

- Los eventos tienen un tipo, un origen, y pueden estar asociados con acciones específicas, como clics, teclas presionadas o movimientos del ratón.

27. **¿Qué es un escuchador de eventos?**

- Un escuchador de eventos es un objeto que "escucha" o detecta eventos y ejecuta un código específico en respuesta a ellos.

28. **¿Qué es una jerarquía de eventos?**

- La jerarquía de eventos es una estructura que organiza los diferentes tipos de eventos en un programa, desde eventos generales hasta eventos más específicos.

29. **¿Define el concepto de hilo?**

- Un hilo es una unidad de ejecución ligera dentro de un programa que puede realizar tareas de manera concurrente.

30. **Describe las características del ciclo de vida de los hilos**

- Los hilos tienen estados como "listo", "ejecutando" y "terminado". Su ciclo de vida incluye la creación, ejecución y finalización.

31. **Describe cómo se crea un hilo**

- Un hilo se crea instanciando

una clase que implementa la interfaz Runnable o extendiendo la clase Thread y luego llamando al método start().

32. **Describe cómo se ejecuta un hilo**

- Un hilo se ejecuta llamando al método start(). El sistema operativo decide cuándo y cómo se ejecuta.

33. **Describe cómo se manipula un hilo**

- Los hilos se pueden manipular mediante métodos como join() para esperar a que un hilo termine y yield() para ceder el control a otros hilos.

34. **Define el concepto de concurrencia**

- La concurrencia es la ejecución simultánea de múltiples hilos o tareas en un sistema para mejorar la eficiencia y la respuesta.

35. **Define el concepto de sincronización de los hilos**

- La sincronización de hilos es el control de acceso a recursos compartidos para evitar problemas como condiciones de carrera y asegurar la coherencia de datos.

36. **Describe las directivas de sincronización a nivel método**

- Las directivas de sincronización a nivel método implican el uso de palabras clave como "synchronized" en Java para bloquear métodos y asegurar la exclusión mutua.

37. **Describe las directivas de sincronización a nivel instrucción**

- Las directivas de sincronización a nivel instrucción implican el uso de mecanismos como semáforos o mutex para controlar el acceso a secciones críticas del código.

38. **Explica el proceso de sincronización de hilos de ejecución**

- El proceso de sincronización de hilos implica el uso de técnicas como bloqueos y señales para garantizar que los hilos accedan a recursos compartidos de manera ordenada y segura.

39. **Define el concepto de socket**

- Un socket es un punto de conexión que permite la comunicación entre dos programas en una red, utilizando protocolos como TCP o UDP.

40. **Define el modelo de comunicación de los sockets punto a punto**

- El modelo de comunicación de sockets punto a punto implica la comunicación directa entre dos programas, donde uno actúa como servidor y el otro como cliente.

41. **Define el modelo de comunicación de los sockets cliente servidor**

- En el modelo de comunicación cliente-servidor, un servidor escucha conexiones de múltiples clientes y les proporciona servicios.

42. **Describa el tipo de socket datagrama**

- Un socket datagrama utiliza el protocolo UDP para la comunicación, donde los datos se envían en paquetes independientes sin garantía de entrega.

43. **Describa el tipo de socket stream**

- Un socket de flujo utiliza el protocolo TCP para la comunicación, garantizando la entrega de datos en orden y sin pérdida.

44. **Describa el tipo de socket raw**

- Un socket crudo o raw permite un acceso de bajo nivel a los paquetes de red sin procesamiento adicional del sistema operativo.

45. **Describa la operación de apertura entre sockets**

- La operación de apertura de sockets implica la creación de un socket, la especificación de la dirección y puerto, y la conexión a otro socket.

46. **Describa la operación de lectura/escritura entre sockets**

- La operación de lectura implica recibir datos desde un socket, mientras que la operación de escritura implica enviar datos a un socket.

47. **Describa la operación de cierre entre sockets**

- La operación de cierre entre sockets implica liberar los recursos asociados con el socket y finalizar la comunicación.

48. **Describa el proceso de codificación de sockets**

- El proceso de codificación de sockets involucra la creación de sockets, la configuración de direcciones y puertos, la lectura y escritura de datos, y la gestión de conexiones y errores.

49. **Defina el concepto de conexión a base de datos**

- La conexión a base de datos es el proceso de establecer una conexión entre una aplicación y una base de datos para realizar operaciones de lectura y escritura en la misma.

50. **Describa las características de conexión a base de datos (conectores y el proceso completo)**

- La conexión a una base de datos implica el uso de conectores específicos del sistema de gestión de bases de datos (DBMS) y el establecimiento de una conexión, creación de consultas y recuperación de resultados.

51. **Implemente una conexión a base de datos con clases y métodos.**

-package proyecto;

```
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import reportes.reportAdministrativo;
import reportes.reportCitas;
```

```
import reportes.reportDepartamentos;
import reportes.reportDoctores;
import reportes.reportEnfermeros;
import reportes.reportPacientes;
import reportes.reportParamedicos;
import reportes.reportTecnicos;

public class MENU {
    JFrame fr1;
    JMenuBar mb1;
    JMenu menu1, menu2, menu3, menu4, menu5;
    JMenuItem menu11, menu12, menu13, menu14, menu15,menu16,menu17,menu18,menu19,
    menu21, menu22,
    menu31, menu32, menu33, menu34, menu35,
    menu36, menu37, menu38,
    menu41,
    menu51;
    JSeparator js1;

    public MENU() {
        fr1 = new JFrame("Menu Principal");
        menu1 = new JMenu("Catalogo");
        menu2 = new JMenu("Proceso");
        menu3 = new JMenu("Reportes");
        menu4 = new JMenu("Consultas");
        menu5 = new JMenu("Ayuda");


        menu11 = new JMenuItem("Citas");
```

```
    menu12 = new JMenuItem("Departamentos");
    menu13 = new JMenuItem("Pacientes");
    menu14 = new JMenuItem("Doctores");
    menu15 = new JMenuItem("Paramedicos");
    menu16 = new JMenuItem("Enfermeros");
    menu17 = new JMenuItem("Administrativos");
    menu18 = new JMenuItem("Tecnicos");
    menu19 = new JMenuItem("Salir");

    menu21 = new JMenuItem("Pagos");
    menu22 = new JMenuItem("Control de Horario");

    //menu31 = new JMenuItem("Enfermeros");
    //menu32 = new JMenuItem("Administrativo");
    //menu33 = new JMenuItem("Departamento");
    menu34 = new JMenuItem("Doctores");
    //menu35 = new JMenuItem("Pacientes");
    //menu36 = new JMenuItem("Citas");
    menu37 = new JMenuItem("Paramedicos");
    //menu38 = new JMenuItem("Tecnicos");

    menu41 = new JMenuItem("Consultas Genericas");

    menu51 = new JMenuItem("Manual de Usuario");

    js1 = new JSeparator();

    mb1=new JMenuBar();
}
```

```
    public void usar() {
        // Configurar imagen de fondo
        ImageIcon imagenFondo = new
ImageIcon(getClass().getResource("/proyecto/imagenes/menu.png"));
        JLabel etiquetaFondo = new JLabel(imagenFondo);
        etiquetaFondo.setBounds(0, 0, imagenFondo.getIconWidth(), imagenFondo.getIconHeight());
        fr1.getLayeredPane().add(etiquetaFondo, Integer.valueOf(Integer.MIN_VALUE));

        // Asegurar que el contenido del JFrame sea transparente para mostrar la imagen
        ((JPanel) fr1.getContentPane()).setOpaque(false);

        // Añadir componentes del menú

        menu1.add(menu11);
        menu1.add(menu12);
        menu1.add(menu13);
        menu1.add(js1);
        menu1.add(menu14);
        menu1.add(menu15);
        menu1.add(menu16);
        menu1.add(menu17);
        menu1.add(menu18);

        menu1.add(menu19);

        menu2.add(menu21);
        menu2.add(menu22);
```

```
//menu3.add(menu31);
//menu3.add(menu32);
//menu3.add(menu33);
  menu3.add(menu34);
//menu3.add(menu35);
//menu3.add(menu36);
  menu3.add(menu37);
//menu3.add(menu38);

menu4.add(menu41);

menu5.add(menu51);

mb1.add(menu1);
mb1.add(menu2);
mb1.add(menu3);
mb1.add(menu4);
mb1.add(menu5);

fr1.setSize(600,800);
fr1.setJMenuBar(mb1);
fr1.setVisible(true);
fr1.setLocation(650, 350);
fr1.setSize(imagenFondo.getIconWidth(), imagenFondo.getIconHeight());
fr1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


//// mandar a llamar tablas////////
```

```
menu11.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
      // de cada clase que tenga que ver con tu menú
      fr1.dispose();
      citas myC=new citas();
      myC.usar();
   }
});
 menu12.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
      // de cada clase que tenga que ver con tu menú
      fr1.dispose();
      departamentos mylogin=new departamentos();
      mylogin.usar();
   }
});
  menu13.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
      // de cada clase que tenga que ver con tu menú
      fr1.dispose();
      pacientes mylogin = new pacientes();
      mylogin.usar();
   }
});
menu14.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
```

```
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
        doctores mylogin = new doctores();
        mylogin.usar();
    }
  });
  menu15.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
        Paramedicos myParam=new Paramedicos();
        myParam.usar();
    }
  });
  menu16.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
        Enfermeros myEnf=new Enfermeros();
        myEnf.usar();
    }
  });
menu17.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
```

```
        fr1.dispose();
        Administrativos myAdmin = new Administrativos();
        myAdmin.usar();
    }
 });
menu18.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
        Tecnicos myTec=new Tecnicos();
        myTec.usar();
    }
 });

            ///////// PROCESOS//////////////

 menu21.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
         CalcularPago myCP = new CalcularPago();
         myCP.usar();
    }
 });
   menu22.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
```

```
        // de cada clase que tenga que ver con tu menú
        fr1.dispose();
          ControlHoras controlHoras = new ControlHoras();
          controlHoras.usar();
     }
 });

  ////////////////REPORTES/////////////////

  //menu31.addActionListener(new ActionListener() {
    //public void actionPerformed(ActionEvent e) {
     //     reportEnfermeros  myreporte=new reportEnfermeros ();
     //     myreporte.REPORTE();
    //}
// });
  //menu32.addActionListener(new ActionListener() {
  //  public void actionPerformed(ActionEvent e) {
  //        reportAdministrativo  myreporte=new reportAdministrativo ();
  //        myreporte.REPORTE();
  //  }
// });
  // menu33.addActionListener(new ActionListener() {
  ////  public void actionPerformed(ActionEvent e) {
  //      reportDepartamentos  myreporte=new reportDepartamentos ();
  //      myreporte.REPORTE();
  //  }
 //});
  menu34.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
```

```
        reportDoctores  myreporte=new reportDoctores ();
        myreporte.REPORTE();
    }
 });
    // menu35.addActionListener(new ActionListener() {
    // public void actionPerformed(ActionEvent e) {
    //      reportPacientes  myreporte=new reportPacientes();
    //      myreporte.REPORTE();
    // }
// });
   //  menu36.addActionListener(new ActionListener() {
   //  public void actionPerformed(ActionEvent e) {
   //      reportCitas  myreporte=new reportCitas ();
   //      myreporte.REPORTE();
   //  }
   //  });
    menu37.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
         reportParamedicos myreporte=new reportParamedicos();
         myreporte.REPORTE();
    }
    });
   //  menu38.addActionListener(new ActionListener() {
   //  public void actionPerformed(ActionEvent e) {
   //       reportTecnicos  myreporte=new reportTecnicos ();
   //       myreporte.REPORTE();
   //  }
   //  });
```

```
        ////////////CONSULTAS/////////////////
        menu41.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           fr1.dispose();
             muertos mymp = new muertos();
           mymp.usar();
        }
        });

        ///////////////AYUDAAAAA/////////////
       menu51.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           // Después del fr1.dispose manda a llamar las últimas 2 líneas de código de los main
           // de cada clase que tenga que ver con tu menú
       Abrirpdf.main(null);
        }
      });
}

   public static void main(String[] args) {
       try {
      // Establecer Look and Feel Nimbus
      for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
         if ("Nimbus".equals(info.getName())) {
            UIManager.setLookAndFeel(info.getClassName());
            break;
         }
      }
   } catch (Exception e) {
```

```
        // Si no se puede establecer el Look and Feel Nimbus, usar el predeterminado del sistema
        e.printStackTrace();
    }

        MENU pro = new MENU();
        pro.usar();
    }
}

//////////////////////////////////////conexión a base de datos//////////////////////////////////////
package DAO;
import java.sql.*;
import javax.swing.JOptionPane;

public class conexion {
    Connection con;
    PreparedStatement stmt;
    ResultSet tabla;
    String cadena, driver, sql;
    int sw=0;

    public conexion(){
        cadena="jdbc:mysql://localhost/"
         + "proyecto?user=root&password=root12345";
        driver= "com.mysql.jdbc.Driver";
    }

    public Connection conecta(){
        try{Class.forName(driver);
```

```
        con= DriverManager.getConnection(cadena);
        System.out.println("conexion con exito");
         }catch (ClassNotFoundException e){
            JOptionPane.showMessageDialog(null, e.getMessage());
         }catch (SQLException e1){
            JOptionPane.showMessageDialog(null, e1.getMessage());  }
         return con;
}
   public static void main(String[] args){
      conexion mycon=new conexion();
      mycon.conecta();
   }
}
```

Etc,etc,etc (es mucho código que tengo que copiar pero en resumen creo que esa es la respuesta) es un código que yo hice.