Primer Certamen

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Primera parte, **sin apuntes** (32 minutos; 32 puntos):

1.- Responda brevemente y entregue esta hoja con su nombre. (Cuide su caligrafía, 4 puntos cada respuesta)

a) Sea la clase **Circle**, la cual contiene el atributo *estático double* **PI**:

```
class Circle {
        …
        public static double PI = 3.1416;
        …
}
```

Durante la ejecución de un programa, el objeto **circulo1**, instancia de **Circle**, modifica **PI**:

```
        …
        circulo1.PI = 3.1;
        …
```

i) ¿Afecta esto al resto de las instancias de la clase **Circle**?

Sí. El uso de static hace que el atributo PI sea parte de la clase **circle** y no de una instancia particular. Por lo tanto, si una instancia hace cambios a este atributo, afectará a todas las instancias por igual.

ii) ¿Cómo cambia la situación previa si **PI** es declarado como **public static final double PI = 3.1416** ?

Si un atributo o método de una clase es declarado como **final**, este no puede ser re-definido o sobre-escrito. Por lo que este cambio generaría un **error de compilación** al momento de intentar modificar el valor de PI.

b) Suponga que cuenta con la clase **Vector2D**

```
class Vector2D {
        public Vector2D(double Xin, double Yin) {
                x = Xin;
                y = Yin;
        }
        public double x = 0.0;
        public double y = 0.0;
}
```

Para los siguientes casos, indique la salida por pantalla, de haber diferencia explique el motivo:

```
…
    Vector2D v1(1.1 , 2.2);
    Vector2D v2 = v1;
     System.out.println("Eval1: "+(v1==v2));
     System.out.println("Eval2: "+v1.equals(v2));
…
```

```
…
    Vector2D v1(1.1 , 2.2);
    Vector2D v2(v1.x , v1.y);
     System.out.println("Eval1: "+(v1==v2));
     System.out.println("Eval2: "+v1.equals(v2));
…
```

Suponiendo que el método equals se encuentra bien definido, para el caso de la izquierda el programa arroja:

Eval1: true
Eval2: true

- Las referencias a las que apunta v1 es igual a v2. Además, su contenido es el mismo.

Para el caso de la derecha el programa arroja:

Eval1: false
Eval2: true

- Las referencias a las que apunta v1 es diferente a v2. Sin embargo, su contenido es el mismo.

c) Como parte del código de un software de manejo de gestión de empleados, se han definido las clases **Employee** y **Manager**, siendo esta última una extensión de la clase **Employee**. Esto permite contar con una instancia que además de ser **Employee**, se le puede asignar y solicitar funciones propias de un cargo de gerencia, como ejecutar un control de proyectos a través del método ***projectControl()***.

i. ¿Cuál de las siguientes asignaciones son correctas? Comente
Employee e = New Manager();
Manager m = New Employee();

La primera asignación es correcta. Es posible asignar una instancia de tipo **Manager** a un objeto **Employee**, ya que un **Manager** puede realizar todo lo que un **Employee** realiza (**Manager** es una *extensión* de **Employee**).
La segunda asignación no es correcta. No es posible asignar una instancia de **Employee** a un objeto de tipo **Manager**, ya que es posible que se le exijan tareas específicas de **Manager**, para las que **Employee** no cuenta con una implementación.

ii. Se cuenta con una lista dinámica de **Employee.**
ArrayList<Employee> e = New ArrayList<Employee>;
e.add((Employee) new Manager("buenaJefa")); // único elemento del arrayList.

// Complete el código para invocar el método projectControl del mánager ingresado.

```
Employee tmpE = e.get(0); // Se accede al empleado
if (tmpE instanceof Manager) { // Nos aseguramos de que sea de tipo Manager
        Manager m = (Manager) tmpE; // Casteamos Employee a Manager
        m.projectControl();  // Ahora podemos acceder a los métodos de Manager
}
```

d) El uso de métodos que arrojan excepciones requiere que éstas se manejen.

i. Comente cuál es la función de los bloques utilizados al capturar excepciones: ***try***, ***catch***, y ***finally***.

try { *// Bloque try*
Ejecución de código de forma segura. Si algo falla, se generará un evento de error que puede ser capturado con **catch**.
} catch (e-clase1 e) { *// Bloque catch*
El bloque **catch** permite realizar una serie de instrucciones específicas para abordar una situación de error. Para cada tipo de error, existirá alguna clase **exception** (en este bloque sería **e-clase1**), que incluye los errores que se han de manejar, por ejemplo IOException para errores relacionados al uso de entradas/salidas..

} catch (e-clase2 e) { *// Bloque catch*
Aquí se manejarían excepciones de tipo e-clase2.

} …
finally { *// Bloque finally*
Permite ejecutar instrucciones al finalizar el bloque try, luego de haber generado y respondido a alguna (o ninguna) excepción.
}

ii. ¿Qué significa relanzar una excepción?

La sección de código (método) que generó la excepción tiene la posibilidad de manejarla o relanzarla. Al relanzarla, se deberá encargar de manejar la excepción la sección de código que llamó al método donde se generó la excepción. Eso permite entregar la responsabilidad de manejar los problemas a un nivel de jerarquía superior, de forma más general, en comparación a sólo manejarlo en la posición donde se generó el problema. La excepción a ser manejada puede subir hasta el mismo método **main**.

Para esto, el método debe indicar **throws exceptionType** en su declaración, por ejemplo:

public static void readFile() throws IOException { … }

e) Liste los calificadores de visibilidad (= modificador de acceso) de métodos y atributos. Dejando fuera el más restrictivo y el menos restrictivo, explique el uso de los restantes.
Modificadores de acceso: private, protected, public y la omisión del calificador de visibilidad.
private y public son el más y el menos restrictivo respectivamente.
protected: se usa cuando se desea dar acceso a una clase heredada de aquella que define el método o atributo protected o bien alguna clase del mismo paquete.
Omisión del modificador de acceso: corresponde a acceso permitido a todas las clases que forman parte de un mismo paquete.

f) Un estudiante dice: “Un RobotView es un Robot con apariencia visual”. Basado en esto el estudiante diseña la clase RobotView como clase hija de Robot. Mencione una ventaja y una desventaja de este diseño.
Ventaja: RobotView tiene acceso directo a todos los métodos de Robot necesarios para actualizar su vista. De otro modo RobotView debería tener una referencia al Robot del cual es vista.
Desventaja: Un mismo Robot no podría cambiar su apariencia visual (cuadrado, circular, con orugas, etc) o tener varias vistas.

g) Re-escriba el siguiente loop usando el lazo “for mejorado”

ArrayList<Employee> workers;
:
for (int i=0; i < workers.size(); i++)
  System.out.println(workers.get(i).toString());
:
	ArrayList<Employee> workers;
	:
	for (Employee e: workers)
	  System.out.println(e.toString());
	:

h) Para las dos clases dadas implemente el método clone, el cual redefine el de la clase Object.

```
class A implements Cloneable {
    private int a;
    public A clone() {
        // código pedido
       try {   // ausencia no descuenta en respuesta
         return (A) super.clone();
       } catch (CloneNotSupportedException e) {
          return null;
       }
    }
}
```

```
class B extends A implements Cloneable{
    private Date d;   // Date ya existente en Java
    public B clone() {
        // código pedido
       try {   // ausencia no descuenta en respuesta
         B b = (B) super.clone();
         b.d = d.clone();
         return b;
       } catch (CloneNotSupportedException e) {
          return null;
       }
    }
}
```

Segunda Parte, con apuntes (68 minutos) **Responda en hojas separadas.**

2.- (34 puntos) Usted cuenta con la clase **Maze** detallada a continuación. Se le pide completar la clase Maze para consultar si hay colisiones o contacto con la pared en un conjunto de puntos. Además se pide incluir un método para obtener la distancia mínima entre un punto y alguna pared dentro de un arreglo de posiciones.

(a) Los prototipos para estos métodos son:

```
// Retorna true si hay pared en el conjunto de puntos pointArea.
public boolean isThere_a_wall(ArrayList<Vector2D> pointArea);

// Debe retornar la distancia mínima entre center y alguna pared en pointArea.
// Si no hay pared, debe retornar -1
public double isThere_a_wall(Vector2D center, ArrayList<Vector2D> pointArea);
```

Considere diponible la clase Vector2D con los siguientes métodos:

```
public double getX() {…}
public double getY() {…}
public Vector2D getUnitary() {…}
public double getModule() {…}
public void setTo(double x, double y) {…}
public Vector2D plus(double scalar) {…}
public Vector2D times(double scalar) {…}
public Vector2D minus(Vector2D v) {…}
public double distanceTo(Vector2D p) {…}
```

```
public class Maze {
   protected Maze(){}
   public Maze(Scanner sc) {
      read(sc);
   }
   public void read(Scanner sc){
      String s;
      while(!sc.hasNextInt())
```

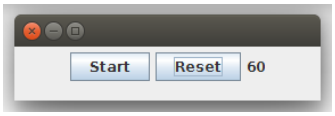
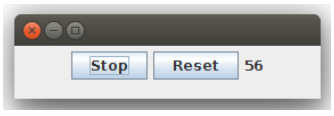
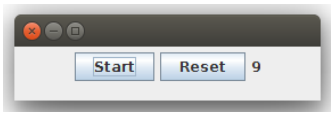
```
            sc.nextLine();
        int width = sc.nextInt();
        int hight = sc.nextInt();
        sc.nextLine();
        array = new boolean [hight][];
        for (int h=0; h<hight; h++)
           array[h] = new boolean[width];
        for (int h=0; h<hight; h++)
           for (int w=0; w<width; ){
              s = sc.findInLine(".");
              if (s==null) sc.nextLine();
              else array[h][w++] = s.charAt(0)=='1';
           }
    }

    protected Maze (Maze m){  // This is called copy constructor
        array = new boolean [m.array.length][];
        for (int h=0; h<array.length; h++)
           array[h] = new boolean[m.array[0].length];
        for (int h=0; h<array.length; h++)
           for (int w=0; w<array[0].length; w++)
              array[h][w]=m.array[h][w];
    }
    public boolean isThere_a_wall(int x, int y) {
        if ((x < array.length) && (y < array[0].length) )
           return array[x][y];
        else return false;
    }
// ****************
// IMPLEMENTED METHODS
    public boolean isThere_a_wall(ArrayList<Vector2D> pointArea) {  // 17 pts.
        boolean b = false;
        for (Vector2D p : pointArea)
           b |= isThere_a_wall((int) Math.round(p.getX()), (int)Math.round(p.getY()));
        return b;
    }
/* Si alguien interpreta el método previo como retonar true en el caso que todos los
puntos detecten pared, la implementación cambia (inicialización true y en lugar de |
= sería &= ) */

// 17 pts.
    public double isThere_a_wall(Vector2D center, ArrayList<Vector2D> pointArea) {
        double distance = -1;
        for (Vector2D p : pointArea)
           if(isThere_a_wall((int) Math.round(p.getX()),(int) Math.round(p.getY()))) {
              if (distance == -1)
                     distance = center.distanceTo(p);
               else if (center.distanceTo(p) < distance)
                   distance = center.distanceTo(p);
           }
        return distance;
    }
//
// ****************
    public void markPoint(Vector2D p){
        int x=(int)p.getX(), y=(int)p.getY();
        if ((x < array.length) && (y < array[0].length))
```

```
            array[x][y]=true;
   }
   public void write(PrintStream out){
      out.println("P1");
      out.println("#Created by "+getClass().getName()+"UTFSM ELO329");
      out.println(array[0].length + " " +array.length);
      for (int h=0; h<array.length; h++) {
         for (int w=0; w<array[0].length; w++)
            out.print(array[h][w]?"1":"0");
         out.println();
      }
   }
   private boolean [][] array;
}
```

3.- (34 puntos) Complete el código adjunto para implementar un temporizador de cuenta regresiva (Countdown Timer) como el mostrado en las siguientes figuras.

		
Situación Inicial	Pasado unos segundos de presionar Start	Luego de presionar Stop

Al presionar Reset, se regresa a la situación inicial.

```
import javax.swing.*;
import java.awt.event.*;

public class CountDownTimer {
   public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
            public void run() {
               CountDownFrame frame = new CountDownFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
    }
}
class CountDownFrame extends JFrame {
   CountDownFrame() {
      JPanel panel = new CountDownPanel();
      getContentPane().add(panel);
      pack();
   }
}
class CountDownPanel extends JPanel implements ActionListener {
   private Timer tick;                                    // 2 pts definición de Timer
   private JButton startStop = new Jbutton("Start"); // 6 pts definición e
   private JButton reset = new Jbutton("Reset"); // inicialización de 2 Botones
   private int t = 60;
   private JLabel count = new JLabel(""+t);  // 3 pts def. e inicializ. de Rótulo

   CountDownPanel () {
      startStop.addActionListener(this);          // 2 pts. registro de listener
      reset.addActionListener(this);             // 2 pts. registro de listener
```

```
      tick = new Timer(1000, new ActionListener (){  // 5 pts. inicializ. Timer e
         public void actionPerformed(ActionEvent event) {  // implement. de listener
            t--;
            count.setText(""+t);
         }
      });
      add(startStop);      // 2 pts inclusión en interfaz
      add(reset);          // 2 pts inclusión en interfaz
      add(count);          // 2 pts inclusión en interfaz
   }
   public void actionPerformed(ActionEvent event) {  // 8 pts. implementación
      String command = event.getActionCommand();     // listener botones
      if (command.equals("Start")){
         tick.start();
         startStop.setText("Stop");
      }
      if (command.equals("Stop")){
         tick.stop();
         startStop.setText("Start");
      }
      if (command.equals("Reset")) {
         tick.stop();
         startStop.setText("Start");
         t=60;
         count.setText(""+t);
      }
   }
}
```