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) Al diseñar la solución para el ascensor de la tarea 1, alguien dice que la "botonera del último piso" es una "botonera de piso intermedio" con sólo un botón. Al notar forma verbal "es un" argumenta que la "botonera de último piso" debe heredar de la "botonera de piso intermedio". ¿Está usted de acuerdo? Justifique.

No. Si bien se lee la relación "es un", no se cumple el principio de sustitución. Este principio no se cumple pues si al requerir una "botonera de piso intermedio" le pasáramos una "botonera de último piso" claramente no podríamos ofrecer todos sus servicios.

b) ¿Por qué al invocar un atributo no estático desde un método estático de la misma clase se genera un error de compilación?

Hay error de compilación porque los métodos estáticos pueden ser invocados sin existir previamente un objeto creado con lo cual no existe aún ese atributo. Otra razón para prohibir este acceso es que aún existiendo varios objetos previamente creados, la invocación de un método estático vía en nombre de la clase no permite identificar el objeto para el cual su atributo debe ser usado.

Nota: no es del todo completa la respuesta si usted dice: No es posible porque desde un método estático sólo se puede acceder a atributos estáticos.

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

```
class A {
    private int a;
    public boolean equals(Object object) {
        // código pedido
    }
}
```

```
class B extends A {
    private Date d;   // Date ya existente en Java
    public boolean equals(Object object) {
        // código pedido
    }
}
```

```
class A {  // si no recuerda la sintaxis con precisión,
    private int a;   // la respuesta es igualmente OK
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null) return false;
        if (getClass() != object.getClass()) return false;
        A obj = (A) object;
        return a==obj.a;
    }
}
```

```
class B extends A {
    private Date d;   // Date ya existente en Java
    public boolean equals(Object object) {
        if (!super.equals(object)) return false;
        B obj = (B) object;
        return d.equals(obj.d);
    }
}
```

Nota: usar getClass() es más estricto que usar instanceof porque instanceof también retorna verdadero cuando hay relación de subtipo (no está bien decir que una persona es igual a un empleado cuando el empleado hereda de persona).

d) En un archivo Java podemos definir varias clases. ¿Por qué el compilador pide crear un archivo por cada clase cuando las clases llevan calificador de visibilidad public?

Las clases públicas pueden ser accedidas desde fuera del archivo que las contiene. Si ponemos dos clases

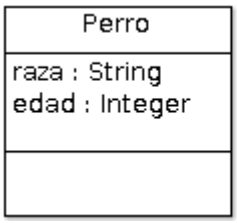
públicas en un archivo, el compilador no podrá encontrar el archivo que contiene la descripción de una de ellas cuando ésta es usada en otro archivo.

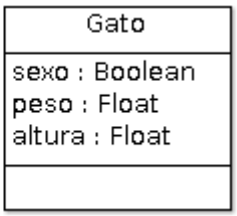
e) En el contexto de programación gráfica en Java ¿cuál es la diferencia entre el modelo y la vista? Dé un ejemplo en el cual un mismo modelo puede estar asociado a más de una vista.

El modelo es el conjunto de atributos que representan a un objeto en memoria, mientras que la vista es la representación visual de las características contenidas en el modelo.

Ej:
Un termómetro es modelado a través de su atributo "temperatura", sin embargo visualmente se puede ver como un termómetro de mercurio o un termómetro digital (u otros tipos de termómetros).

f) Considere las siguientes clases ya definidas:





¿En qué contexto se justificaría la creación de la super clase "Animales"? ¿En qué contexto no se justifica? Dé un ejemplo de cada caso (no se pide código, sólo explicar cada ejemplo.

Una de las ventajas de crear súper clases es evitar la duplicidad de código y favorecer el mantenimiento. Sin embargo, en este caso, si bien, se podría argumentar que ambas clases tienen en común que son animales, en cuanto a su implementación no tienen elementos en común.
Considerando esto como base, no se justificaría la creación de una clase animales si estas clases efectivamente son utilizadas en contextos dentro del código distintos.
En caso de que sí sean utilizadas en un mismo contexto dentro del código y que además sea necesaria la agrupación de estas clases, entonces sí se justificaría la creación de la clase padre animales.

Algunos ejs. de creación clase animales:

- Se necesita agrupar tanto perros como gatos en un arreglo de animales
- Existe un método que recibe animales como entrada, pudiendo ser tanto perros como gatos

Alugnos ejs. de no creación de clase animales:

- Existen métodos separados para recibir gatos y perros
- No sea necesaria ningún tipo de agrupación de animales en el programa

g) En el contexto de programación genérica, cree un método estático **insertar(pos, a, arr)** que inserte un elemento **a** en la posición **pos** de un arreglo java **arr**, siempre y cuando **pos** sea una posición válida. Si la posición es válida, luego de insertar el elemento se retorna **true**, en caso contrario, se retorna **false**. El arreglo **arr** puede contener elementos de **cualquier clase**. Solo debe escribir el código asociado al método, no es necesario escribir la definición de clase.

Ej. de uso: /* Uso: GenericClass.insertar(pos,a,arr); */

```
Integer[] arr = {1,2,3,4,5,6,7,8};          // arr = [1, 2, 3, 4, 5, 6, 7, 8]
int a = 10;
System.out.print(GenericClass.insertar( 0, a, arr));  // arr = [10, 2, 3, 4, 5, 6, 7, 8], imprime true
System.out.print(GenericClass.insertar(15, a, arr)); // arr = [10, 2, 3, 4, 5, 6, 7, 8], imprime false
```

```
public  static boolean insertar(int pos, Object a, Object[] arr){
        if(pos >= 0 && pos < arr.length){
            arr[pos] = a;
            return true;
        }
        return false;
}
```

h) ¿Puede una clase abstracta heredar de otra clase abstracta? ¿Puede una interfaz implementar otra interfaz? Justifique.

Una clase abstracta SÍ puede heredar de otra clase abstracta. Una clase abstracta es una clase cuya única diferencia con el resto es que nunca será instanciada y que tiene al menos un método no implementado. Respetando esta definición, cumple con las mismas características de cualquier clase Java, por lo que no hay ninguna restricción para que se cumpla herencia entre ellas.

Una interfaz NO puede implementar otra interfaz. Una interfaz es la descripción de uno o más métodos que alguna clase puede ofrecer. Utilizar una interfaz implica la implementación de la misma, y por tanto, de los métodos definidos en ella. Una interfaz no sería capaz de implementar los métodos que ofrezca otra interfaz (por definición, una interfaz no tiene ningún método implementado), por lo que es incompatible implementar una en la otra.

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

2.- (34 puntos) 1. Desarrolle el programa DragMe.java. Cuando se ejecute este programa, aparecerá una ventana con un elemento gráfico de tipo rectángulo en la esquina superior izquierda de la misma.
Este rectángulo puede ser arrastrado libremente usando el mouse.
Para arrastrarlo, se debe hacer click con el botón del mouse en cualquier punto interior del rectángulo y, sin soltarlo, mover el mouse con el rectángulo a la nueva posición deseada para luego soltar el botón del mouse. La posición del centro del rectángulo deberá coincidir con la última posición del mouse antes de soltar el botón.
El programa finaliza al cerrar la ventana principal.

Características del programa:
- La ventana tendrá dimensiones de 300 pixeles de ancho y 200 pixeles de alto
- El rectángulo tendrá dimensiones de 100 pixeles de ancho y 30 pixeles de alto

```
package dragme;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class DragMe {
    public static void main(String[] args) {
        RectangleFrame frame = new RectangleFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class RectangleFrame extends JFrame{
    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;
    public RectangleFrame(){
        setTitle("Mueve rectangulo");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        RectanglePanel panel = new RectanglePanel();
```

```
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }
}

class RectanglePanel extends JPanel{
    private Rectangle2D rect;
    private final int ancho = 100;
    private final int alto = 30;

    public RectanglePanel(){
        double x = 0;
        double y = 0;

        rect = new Rectangle2D.Double(x,y,ancho,alto);
        repaint();
        addMouseMotionListener(new MouseMotionHandler());
    }
    public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      g2.draw(rect);

    }
    private class MouseMotionHandler implements MouseMotionListener{
        public void mouseMoved(MouseEvent event){
        }

        public void mouseDragged(MouseEvent event){
            int x = event.getX();
            int y = event.getY();
            if(rect.contains(event.getPoint())){
                rect.setFrame(
                   x-ancho/2,
                   y-alto/2,
                   ancho,
                   alto);
                repaint();
            }
        }
    }
}
```

Distribución de los 34 puntos:
- Import: 2 pts
- main: 2 pts
    - crear frame y setear visible: 2 pts
- frame: 2 pts
    - Ancho y alto de 300 x 200: 2 pts
    - Crear panel: 2 pts
    - Asociar panel con panel de contenido: 2 pts
- Panel: 3 pts
    - Ancho y alto de 100 x 300 del rectángulo: 2 pts
    - Crear rectángulo: 2 pts
    - Asociar listener de mouse con panel: 2 pts
    - Pintar rectángulo usando graphics: 2 pts
- Mouse listener: 3 pts
    - Implementar método Dragged (o equivalente): 2 pts
    - Verificar punto dentro de rectángulo: 2 pts
    - Setear posición de rectángulo en x-ancho/2,y-alto/2 y repintar: 2 pts

3.- (34 puntos) Se quiere incorporar el método estático **temblor** para poder terminar el programa de la tarea al llegar al próximo piso de subida o bajada.
Cree la clase **Emergencia** con un único **método estático temblor (ControlUnit cu)** que al invocarlo haga

que el programa se detenga la próxima vez que llegue a un piso.

a) Implemente la clase Emergencia
b) Muestre los cambios a incorporar en la clase ControlUnit. Si usted considera cambios en otras clases, inclúyalos también.

```
public class ControlUnit {
   private Motor motor;
   private Cabina cabina;
   private Sensor[] sensores;
   private Botonera[] botoneras;

   public ControlUnit(Motor m,Cabina ca, Sensor[] s, Botonera[] b){
      // .... código existente
   }
   public void elevatorRequested(int locationRequest){
      // .... código existente
   }

   private void checkAndAttendUpRequest(int floor) {
      // .... código existente
   }
   private void checkAndAttendDownRequest(int floor) {
      // .... código existente
   }
   private void checkAndAttendCabinaRequest(int floor) {
      // .... código existente
   }
   public void activateSensorAction(int currentFloor){
      // .... código existente
   }
   private boolean areThereHigherRequests(int currentFloor) {
      // .... código existente
   }

   private boolean areThereLowerRequests(int currentFloor) {
      // .... código existente
   }
}
```

a)

```
public class Emergencia {                              //
   public static void temblor (ControlUnit cu){    //   8 pts
      cu.emergencyStop();       // 4 pts.
   }
}
```

b) Sólo se muestra los métodos modificados.

```
public class ControlUnit {
   private Motor motor;
   private Cabina cabina;
   private Sensor[] sensores;
   private Botonera[] botoneras;
   private boolean emergencyState;  // atributo nuevo   6 pts

   public ControlUnit(Motor m,Cabina ca, Sensor[] s, Botonera[] b){
      // .... código existente
      emergencyState = false;    // 4 pts
   }

   public void emergencyStop() { // método nuevo   6 pts
      emergencyState = true;
```

```
    }

    public void activateSensorAction(int currentFloor){
        if (emergencyState) System.exit(0);          // 6 pts.
        // .... código existente
    }
}
```