**PREGUNTA 1**

**ANALICE EL CÓDIGO QUE SE MUESTRA A CONTINUACIÓN - 10 minutos**

class Direccion{

private String calle1;

private int numero;

private String calle2;

public Direccion(String calle1, int numero, String calle2) {

this.calle1 = calle1;

this.numero = numero;

this.calle2 = calle2;

}

}

public class Cliente {

private String nombre;

private Direccion direccion;

private String password;

public Cliente(String nombre, Direccion direccion, String password) {

this.nombre = nombre;

this.direccion = direccion;

this.password = password;

}

}

Indique brevemente qué cambios debe realizarse en el código mostrado para que se pueda serializar un objeto de la clase Cliente preservando todas su variables menos el password.

**PREGUNTA 2**

**ANALICE EL CÓDIGO QUE SE MUESTRA A CONTINUACIÓN - 10 MINUTOS**

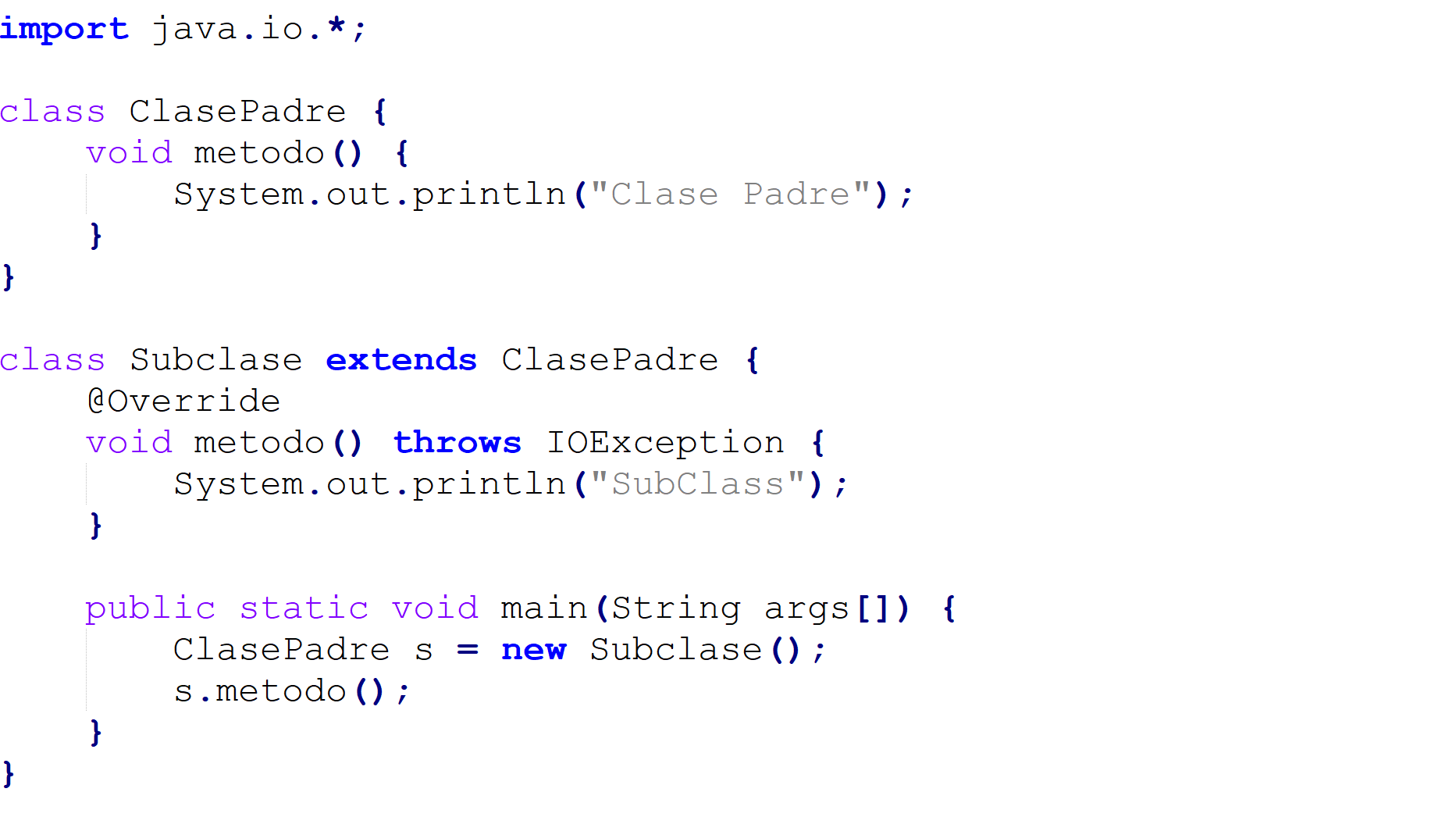
|  |
| --- |
| public interface Editable {  int obtenerNumero();  default String obtenerNombre() {  return "Nombre por defecto";  }  static int obtenerCodigoPais() {  return 0;  }  }  class Contactotelefonico implements Editable {  public int obtenerNumero() {  return 987354723;  }  public String obtenerNombre() {  return "Carlos Marín";  }  static int obtenerCodigoPais() {  return 593;  }  }  class AgendaTelefonica {  public static void main(String args[]) {  Editable obj = new Contactotelefonico();  //sentencias de la tabla  }  } |

**Indique que retornan cada una de las sentencias a continuación, que son ejecutadas en el main de la clase AgendaTelefonica. (Justifique su respuesta brevemente)**

|  |  |  |
| --- | --- | --- |
|  | **sentencia** | **Valor que retorna** |
| **a)** | **obj.obtenerNombre()** |  |
| **b)** | **Editable.obtenerCodigoPais()** |  |
| **c)** | **obj.obtenerNumero()** |  |

d) Se desea definir **una nueva clase** **NO ABSTRACTA** que implemente la interfaz **ContactoTelefonicoI ¿qué métodos debería implementar la clase?**

**ANALICE EL CÓDIGO QUE SE MUESTRA A CONTINUACIÓN**



**¿Cuál es la salida? Justifique su respuesta.**

El código no compila - El método en Subclase no puede lanzar una excepción que la clase Padre no lanza.

**ANALICE EL CÓDIGO QUE SE MUESTRA A CONTINUACIÓN**

**class CalculoSalarioException extends Exception{}**

**class Person{**

**public void calcularSalario(double salario) throws CalculoSalarioException{**

**throw new CalculoSalarioException();**

**System.out.println(salario\*2);**

**}**

**}**

**class Compania{**

**public static void main(String[] args){**

**new Person().calcularSalario(200);**

**}**

**}**

**¿Cuál de los siguientes enunciados de forma independiente es verdadero?**

1.-Este código compila sin problemas, la salida es 400

2.-Este código compila si el método calcularSalario devuelve un double en vez de ser un void

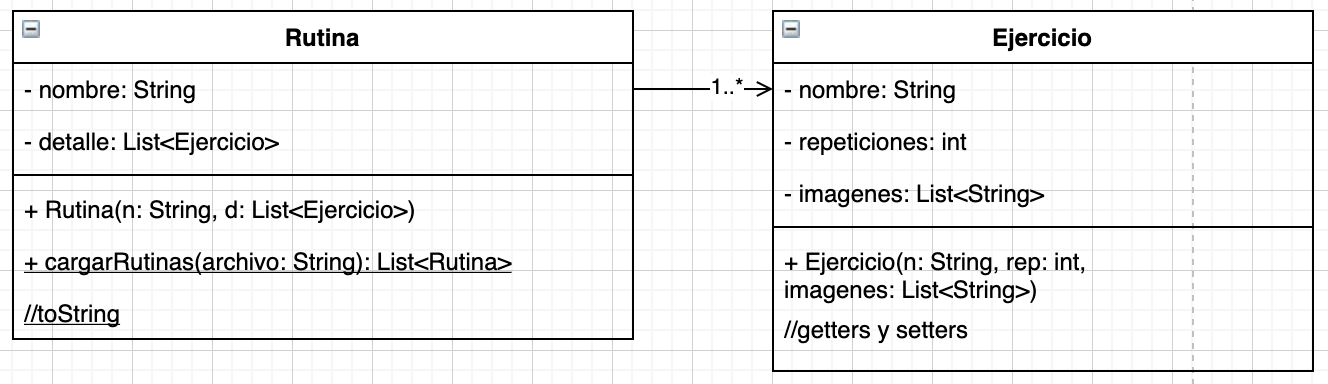
3.-Este código compila si se agrega un bloque try-catch en el método main de la clase Companía.

4.-Este código compila si agregamos un throws CalculoSalarioException en la firma del método main de la clase Companía.

**(70 puntos) DESARROLLO -**

Se le ha pedido que desarrolle una aplicación para un Gimnasio que permita a los asistentes ver información de rutinas de ejercicios.

A continuación se muestra el diagrama de la capa modelo de la aplicación



* Una **rutina** tiene un nombre y una lista de ejercicios de esa rutina**.**
* Un Ejercicio tiene un **nombre,** el número de repeticiones de ese ejercicioy una lista con nombres de las imágenes que muestran cómo hacer el ejercicio.

Existe un archivo llamado **rutinas.txt** que contiene la información de las rutinas y tiene la siguiente estructura:

nombrerutina;nombreeje1,rep1,img1,img2;nombreje2,rep2,img1,img2;……

|  |
| --- |
| abdomen;crisscross,10,crisscross1.png,crisscross2.png;rollouts,15,rollouts1.png,rollouts2.png;..  piernas;puente cadera,20,puentecadera1.png,puentecadera2.png;patadas,15,patadas1.png,patadas2.png |

**PARTE 1**

1. **(5 puntos)** Cree una clase llamada **GymException** que sea de tipo unchecked. La clase debe tener un constructor que reciba el mensaje de la excepción.

En la clase **Ejercicio**

1. **(5 puntos)** Implemente la interfaz **Comparable** para que el orden natural sea por nombre. Si el nombre es igual, entonces considere el número de repeticiones.

En la clase **Rutina**

1. **(15 puntos)** Implemente el método **cargarRutinas(String nombreArchivo)** para que retorne una lista de objetos de tipo **Rutina**.

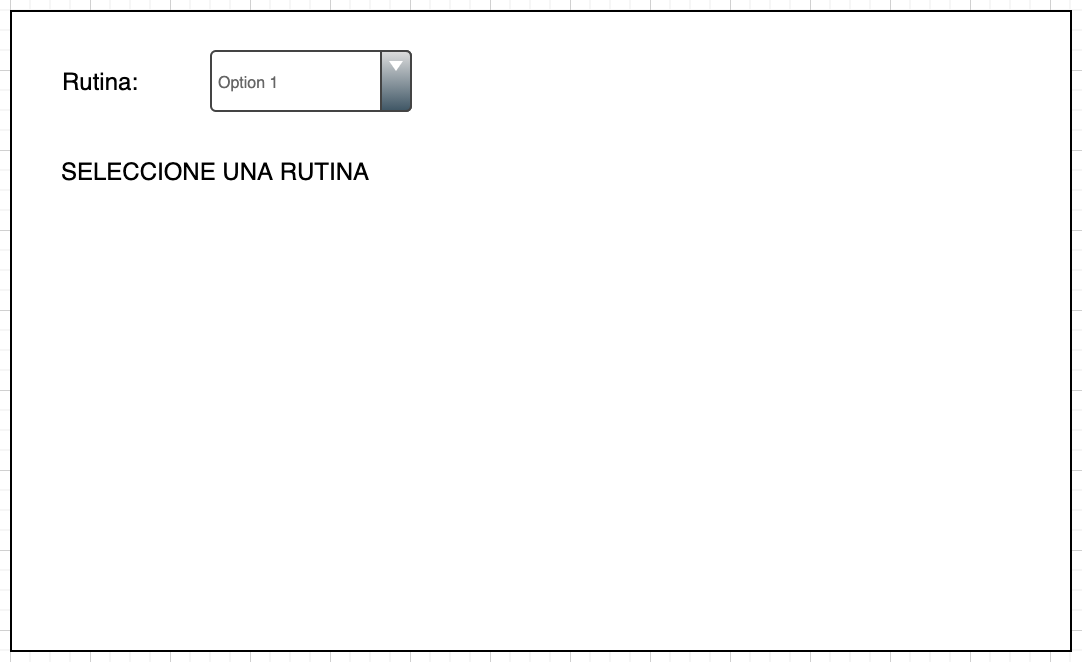
Si se produce una excepción de tipo IOException debe capturarla y lanzar una nueva excepción de tipo **GymException** con un mensaje que indique el tipo de excepción que ocurrió

**PARTE 2**

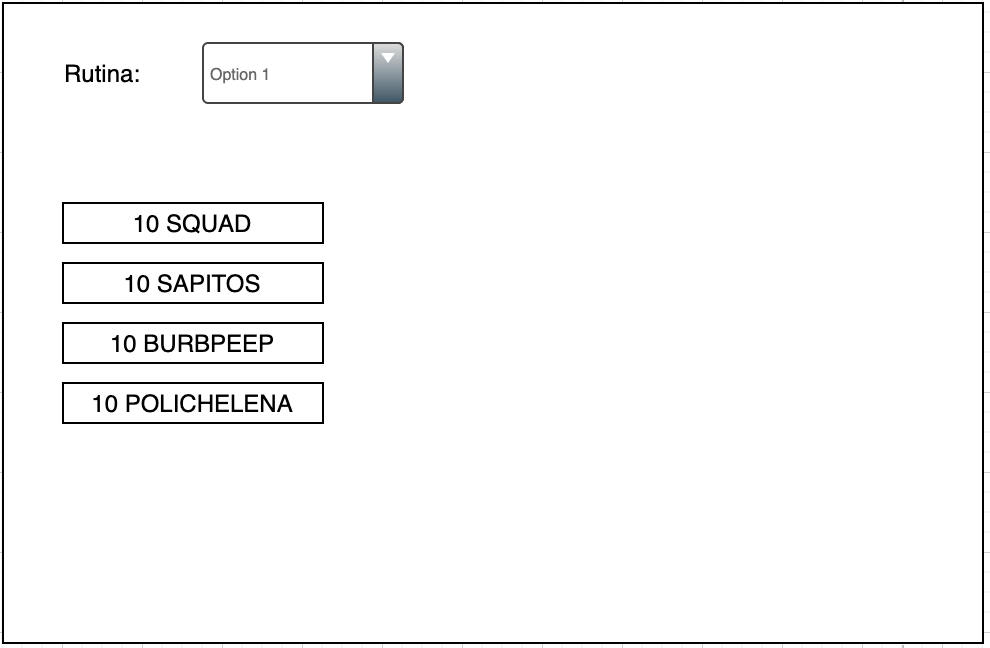
Al iniciar la aplicación aparece una ventana donde el usuario puede escoger de una lista desplegable la rutina que prefiera **- (15 puntos)**



**Su programa debe manejar las excepciones de tipo GymException y en caso de presentarse debe mostrarse en la pantalla un mensaje apropiado al usuario**

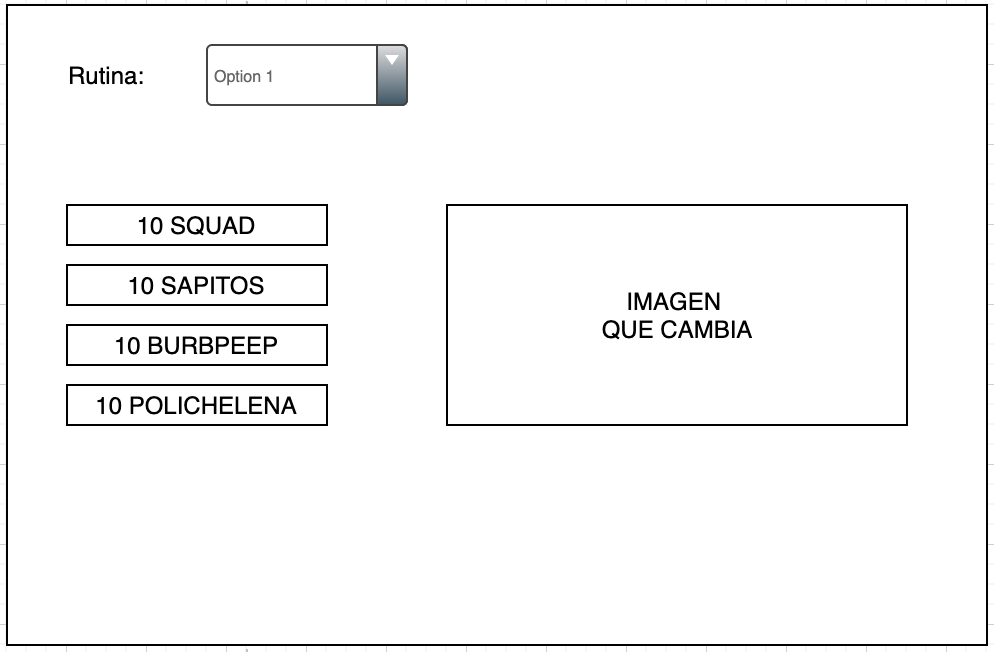
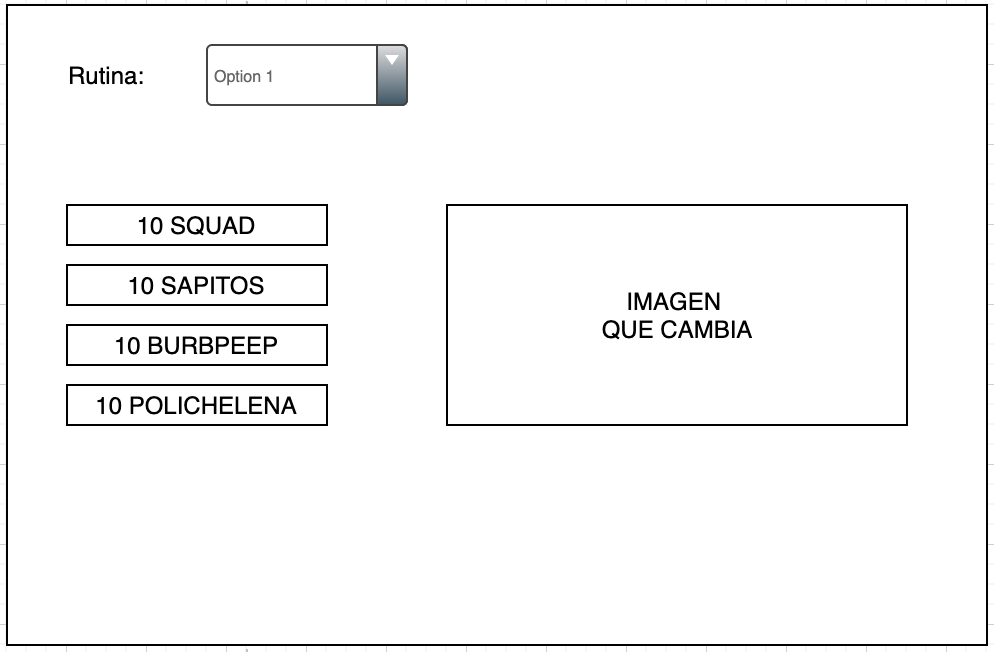


Al seleccionar la rutina se muestra en pantalla los ejercicios de la rutina como se ve a continuación **- (15 puntos)**



* Cada botón muestra el número de repeticiones y el nombre de un ejercicio.
* Al dar click a un botón se muestran las imágenes asociadas con el ejercicio en secuencia tantas veces como indica el número de repeticiones del mismo - Las imágenes se intercalan cada 0.5 segundos **(15 puntos)**

**NOTA: Asuma que no se da click en otro botón antes de acabarse las repeticiones de un ejercicio.**



**RÚBRICA**