**CARRERA DESARROLLO DE SOFTWARE**

**PROGRAMACION ORIENTADA A OBJETOS - [DSOF] - 6268**

**Mini Sistema Concurso**

**Chuncho Vera, Jorge Javier**

[jorge.chuncho@cenestur.edu.ec](mailto:jorge.chuncho@cenestur.edu.ec)

**Franco Rocha, Yadira Guissela**

**Quito, Ecuador**

**2025**

El siguiente documento describe un sistema de inscripción para un concurso de natación siguiendo los principios de la Programación Orientada a Objetos, para el mismo se dan las siguientes instrucciones generales:

1. Crea una carpeta llamada concurso\_natacion.
2. Copia cada clase en su propio archivo .java con el nombre correspondiente.
3. El Main.java se usará para el menú de ejecución.
4. Archivos a crear:
   * Participante.java
   * ParticipanteColegio.java
   * ParticipanteUniversidad.java
   * ParticipanteEmpresa.java
   * ParticipanteSindicato.java
   * Main.java

Todos los participantes deben registrarse con los siguientes atributos comunes:

* Nombre
* Edad
* Género
* Condición de salud (buena o no)
* Tipo de entidad (colegio, universidad, empresa pública, empresa privada o sindicato)

Solo podrán participar personas que cumplan con los siguientes requisitos:

* Tener entre 18 y 25 años
* Estar en buena condición de salud

A continuación, se mencionan los requisitos del sistema:

* Crear una clase base Participante que contenga los atributos comunes y un método puedeParticipar() que valide la edad y condición de salud.
* Crear al menos cuatro subclases que hereden de Participante:
  + ParticipanteColegio
  + ParticipanteUniversidad
  + ParticipanteEmpresa (debe indicar si es pública o privada)
  + ParticipanteSindicato
* Cada subclase debe incluir su atributo adicional (ej. nombre del colegio o sindicato) y sobrescribir un método getInfo() para mostrar los datos completos.
* Implementar getters y setters en las clases cuando sea necesario.
* Crear un menú interactivo por consola que permita:
  + Registrar nuevos participantes
  + Ver la lista de participantes registrados
  + Ver solo los que sí pueden participar (según edad y salud)
* Validar correctamente:
  + Que la edad esté entre 18 y 25 años
  + Que la condición de salud sea “buena” (puede ser un booleano o un texto)

En las siguientes páginas, se comparte el código de cada clase:

|  |
| --- |
| **Particpante.java** |
| public class Participante {  private String nombre;  private int edad;  private String genero;  private boolean buenaSalud;  private String tipoEntidad;   //Constructor  public Participante (String nombre, int edad, String genero, boolean saludBuena, String tipoEntidad) {  this.nombre=nombre;  this.edad=edad;  this.genero=genero;  this.buenaSalud = saludBuena;  this.tipoEntidad=tipoEntidad;  }   //Métodos principales  //Verificar si la persona puede o no participar  public boolean puedeParticipar() {  return (edad >=18 && edad <=25) && buenaSalud;  }   public String getInfo() {  return "Nombre: " + nombre +  ", edad: "+ edad +  ", Género: " + genero +  ", Salud: " + (buenaSalud ? "Buena" : "No apta") +  ", Entidad: " + tipoEntidad;  }   public boolean isBuenaSalud() {  return buenaSalud;  }   public int getEdad() {  return edad;  } } |

|  |
| --- |
| **ParticipanteColegio.java** |
| public class ParticipanteColegio extends Participante{  private String nombreColegio;   //Constructor  public ParticipanteColegio(String nombre, int edad, String genero, boolean buenaSalud, String nombreColegio) {  super (nombre, edad, genero, buenaSalud, "Colegio");  this.nombreColegio=nombreColegio;  }   @Override  public String getInfo() {  return super.getInfo() + ", Colegio: " + nombreColegio;  }  } |

|  |
| --- |
| **ParticipanteEmpresa.java** |
| public class ParticipanteEmpresa extends Participante {  private String tipoEmpresa; // "Pública" o "Privada"   //Constructor  public ParticipanteEmpresa(String nombre, int edad, String genero, boolean buenaSalud, String tipoEmpresa) {  super (nombre, edad, genero, buenaSalud, "Empresa");  this.tipoEmpresa = tipoEmpresa;  }  //Metodo  @Override  public String getInfo() {  return super.getInfo() + ", Tipo Empresa: " + tipoEmpresa;  }  } |

|  |
| --- |
| **ParticipanteSindicato.java** |
| public class ParticipanteSindicato extends Participante {  private String tipoSindicato;   //Constructor  public ParticipanteSindicato(String nombre, int edad, String genero, boolean buenaSalud, String tipoSindicato) {  super (nombre, edad, genero, buenaSalud, "Sindicato");  this.tipoSindicato=tipoSindicato;  }   //Metodo  @Override  public String getInfo () {  return super.getInfo() + ", Sindicato: " + tipoSindicato;  } } |

|  |
| --- |
| **ParticipanteUniversidad.java** |
| public class ParticipanteUniversidad extends Participante {  private String nombreUniversidad;   //Constructor  public ParticipanteUniversidad(String nombre, int edad, String genero, boolean buenaSalud, String nombreUniversidad) {  super (nombre, edad, genero, buenaSalud, "Universidad");  this.nombreUniversidad=nombreUniversidad;  }  //Metodo  @Override  public String getInfo() {  return super.getInfo() + ", Universidad: " + nombreUniversidad;  }  } |

|  |
| --- |
| **Main.java** |
| // Importamos las clases necesarias import java.util.ArrayList; // Para manejar listas dinámicas de participantes import java.util.Scanner; // Para capturar entradas del usuario por consola  // Clase principal que contiene el mé*todo main* public class Main {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.*in*); // Inicializamos el lector de datos desde consola  ArrayList<Participante> participantes = new ArrayList<>(); // Lista donde se almacenarán todos los participantes registrados   int opcion;  // Ciclo do-while para mostrar el menú hasta que el usuario decida salir (opción 0)  do {  // Mostramos el menú principal  System.*out*.println("\n--- MENÚ CONCURSO DE NATACIÓN ---");  System.*out*.println("1. Registrar Participante");  System.*out*.println("2. Ver lista de participantes");  System.*out*.println("3. Ver participantes que pueden concursar");  System.*out*.println("0. Salir");  System.*out*.print("Ingrese opción: ");  opcion = scanner.nextInt(); // Leemos la opción elegida  scanner.nextLine(); // Limpieza del buffer para leer Strings correctamente   switch (opcion) {  case 1:  // Registro de un nuevo participante  System.*out*.print("Nombre: ");  String nombre = scanner.nextLine();   System.*out*.print("Edad: ");  int edad = scanner.nextInt();  scanner.nextLine();   System.*out*.print("Género: ");  String genero = scanner.nextLine();   System.*out*.print("Salud buena (true/false): ");  boolean salud = scanner.nextBoolean();  scanner.nextLine();   // Selección del tipo de entidad  System.*out*.println("Tipo de entidad (1.Colegio 2.Universidad 3.Empresa 4.Sindicato): ");  int tipo = scanner.nextInt();  scanner.nextLine();   Participante p = null; // Se declara la variable tipo Participante para instanciar según el tipo   // Submenú que determina la subclase a instanciar según tipo de entidad  switch (tipo) {  case 1:  System.*out*.print("Nombre del Colegio: ");  String colegio = scanner.nextLine();  p = new ParticipanteColegio(nombre, edad, genero, salud, colegio); // Instancia de subclase Colegio  break;  case 2:  System.*out*.print("Nombre de la Universidad: ");  String universidad = scanner.nextLine();  p = new ParticipanteUniversidad(nombre, edad, genero, salud, universidad); // Subclase Universidad  break;  case 3:  System.*out*.print("Tipo de Empresa (Pública o Privada): ");  String empresa = scanner.nextLine();  p = new ParticipanteEmpresa(nombre, edad, genero, salud, empresa); // Subclase Empresa  break;  case 4:  System.*out*.print("Tipo de Sindicato: ");  String sindicato = scanner.nextLine();  p = new ParticipanteSindicato(nombre, edad, genero, salud, sindicato); // Subclase Sindicato  break;  default:  System.*out*.println("Opción inválida."); // Si el tipo no es válido  }   // Si se pudo crear correctamente el objeto Participante (no es nulo), se agrega a la lista  if (p != null) {  participantes.add(p);  System.*out*.println("Participante registrado con éxito.");  }  break;   case 2:  // Mostramos todos los participantes registrados  System.*out*.println("\n--- Lista de Participantes Registrados ---");  for (Participante participante : participantes) {  System.*out*.println(participante.getInfo()); // Mostramos su información usando polimorfismo  }  break;   case 3:  // Mostramos solo los que cumplen los requisitos para participar  System.*out*.println("\n--- Participantes Aptos para el Concurso ---");  for (Participante participante : participantes) {  if (participante.puedeParticipar()) { // Se valida usando el mé*todo de la clase base* System.*out*.println(participante.getInfo()); // Solo si puede participar  }  }  break;   case 0:  System.*out*.println("Saliendo del sistema..."); // Salida del programa  break;   default:  System.*out*.println("Opción no válida."); // Si el usuario ingresó una opción no reconocida  }  } while (opcion != 0); // El ciclo se repite mientras la opción no sea salir  scanner.close(); // Cerramos el lector de entrada para liberar recursos  } } |

Para entender mejor el funcionamiento del sistema, hemos creado el siguiente diagrama UML

Diagrama

El contenido generado por IA puede ser incorrecto.

Finalmente reforzamos el contenido con un cuestionario de 20 preguntas en base a los tópicos relacionados con la creación de este sistema expuesto.

1. **¿Qué es la encapsulación en la programación orientada a objetos?**
   1. Esconder el código fuente.
   2. Limitar el acceso directo a los atributos de una clase.
   3. Crear múltiples clases en un archivo.
   4. Crear interfaces entre clases.
2. **¿Cuál es el propósito del término *super* en Java?**
   1. Crear un nuevo método.
   2. Llamar a métodos de una clase externa.
   3. Acceder a los métodos o constructores de la clase principal.
   4. Declarar métodos como públicos.
3. **En Java, un método puede tener el mismo nombre en diferentes clases mientras pertenezcan a jerarquías distintas**
   1. Verdadero
   2. Falso
4. **Los métodos getters y setters son una buena practica para acceder a atributos privados.**
   1. Verdadero
   2. Falso
5. **¿Cuál de las siguientes afirmaciones sobre los constructores es verdadera?**
   1. Deben tener el mismo nombre que la clase
   2. Siempre deben ser públicos
   3. Solo puede haber uno por clase
   4. No pueden recibir parámetros
6. **¿Qué es la herencia en POO?**
   1. La conversión de tipos
   2. La copia de código entre métodos
   3. La capacidad de una clase de adquirir atributos y métodos de otra
   4. El uso de clases anidadas
7. **Una subclase puede acceder directamente a atributos *private* de la clase principal.**
   1. Verdadero
   2. Falso
8. **La palabra *this* se usa para hacer referencia a la clase padre.**
   1. Verdadero
   2. Falso
9. **¿Qué hace @override?**
   1. Crea un nuevo constructor
   2. Indica que un método esta siendo ocultado
   3. Obliga a una clase a ser abstracta
   4. Declara que un método sobrescribe uno de la clase principal
10. **¿Cuál modificador de acceso permite que un miembro sea visible solo dentro de la misma clase?**
    1. Public
    2. Protected
    3. Private
    4. Default
11. **Una sublcase puede acceder directamente a atributos *private* de la clase principal.**
    1. Verdadero
    2. Falso
12. **En Java, el método main() debe ser public static void para que se ejecute como punto de entrada**
    1. Verdadero
    2. Falso
13. **¿Cuál es el propósito de una clase abstracta?**
    1. Instanciar objetos de forma directa
    2. Contener solo atributos
    3. Proveer una plantilla para clases hijas
    4. Generar clases aleatorias
14. **¿Qué es un objeto en Java?**
    1. Una función especial
    2. Una variable global
    3. Una instancia de una clase
    4. Un tipo de método
15. **La herencia en Java permite reutilizar código**
    1. Verdadero
    2. Falso
16. **Una clase puede heredar de más de una clase a la vez usando *extends*.**
    1. Verdadero
    2. Falso
17. **¿Qué hace el siguiente operador lógico en Java &&?**
    1. Compara si dos números son iguales
    2. Evalúa si ambas condiciones son verdaderas
    3. Asigna dos valores a la vez
    4. Niega una condición
18. **¿Qué hace la clase Scanner en Java?**
    1. Imprime texto en consola
    2. Escanea redes
    3. Lee datos del usuario desde teclado
    4. Compara cadenas de texto
19. **Una clase final puede ser heredada por otras clases.**
    1. Verdadero
    2. Falso
20. **Si una clase no define ningún constructor, Java crea uno por defecto sin parámetros.**
    1. Verdadero
    2. Falso

Todo esto se puede encontrar en el siguiente repositorio de **GITHUB:** <https://github.com/Xavier9649/JorgeChNatacionC.git>