Informe
Sistema de Gestión de Torneos

Número de grupo y nombres de integrantes

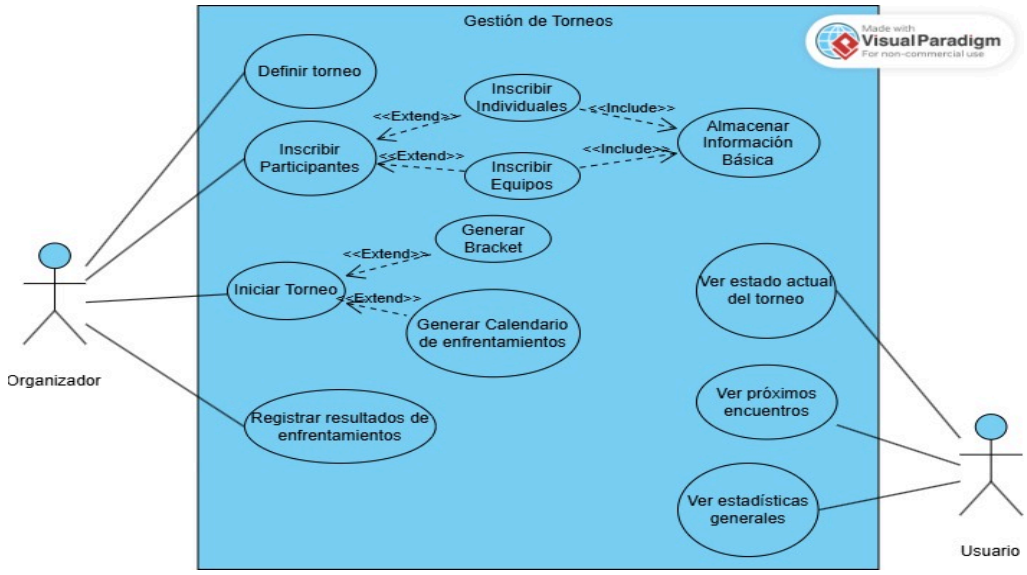
Grupo: N° 13

- Marco Enrique Liguempi Bozzano
- Joaquín Alonso Reyes Tecas
- Matias Sebastian Cuello Diban

Enunciado del proyecto

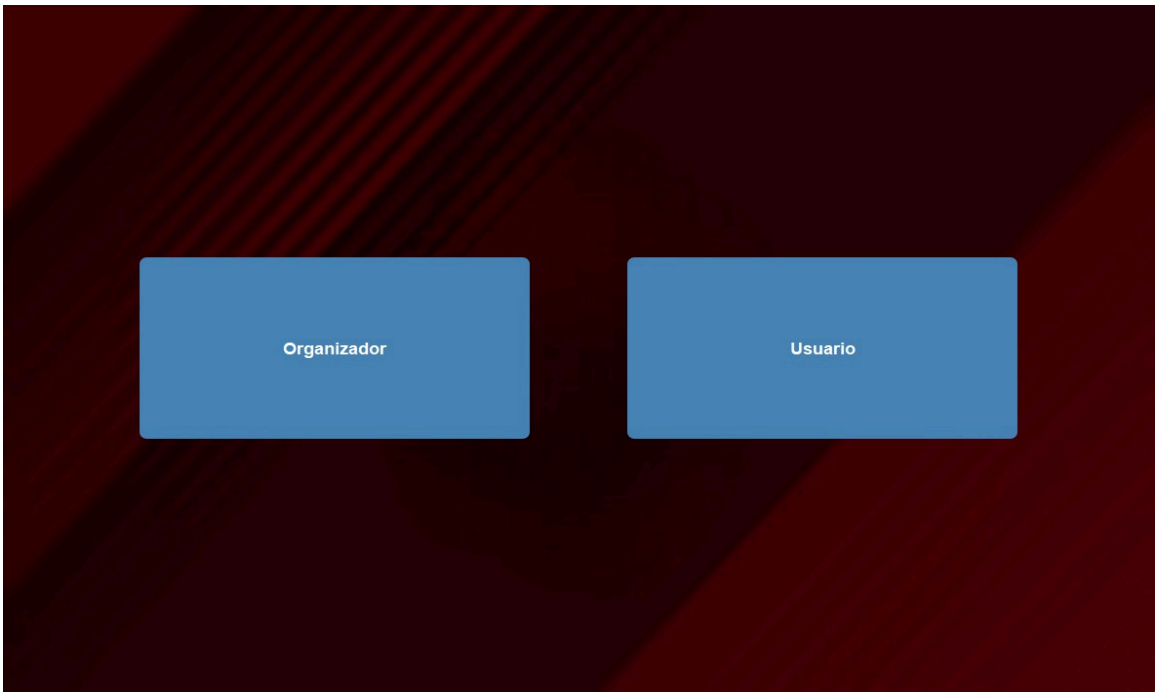
Este sistema está diseñado para facilitar la organización de torneos deportivos o de juegos. Permitirá a un organizador definir las características del torneo, como el nombre, la disciplina (ej. fútbol, ajedrez, videojuegos), las fechas y un formato principal (En nuestro caso la eliminatoria directa). Se podrán inscribir participantes, ya sean jugadores individuales o equipos, almacenando información básica como nombres y datos de contacto. El sistema deberá ser capaz de generar un calendario de enfrentamientos o un bracket inicial basado en los inscritos y el formato. Durante el torneo, se registran los resultados de cada enfrentamiento, lo que actualizará automáticamente las posiciones, el avance en el bracket o las tablas de clasificación. Los usuarios podrán visualizar el estado actual del torneo, los próximos encuentros y las estadísticas generales.

Diagrama de Casos de Uso

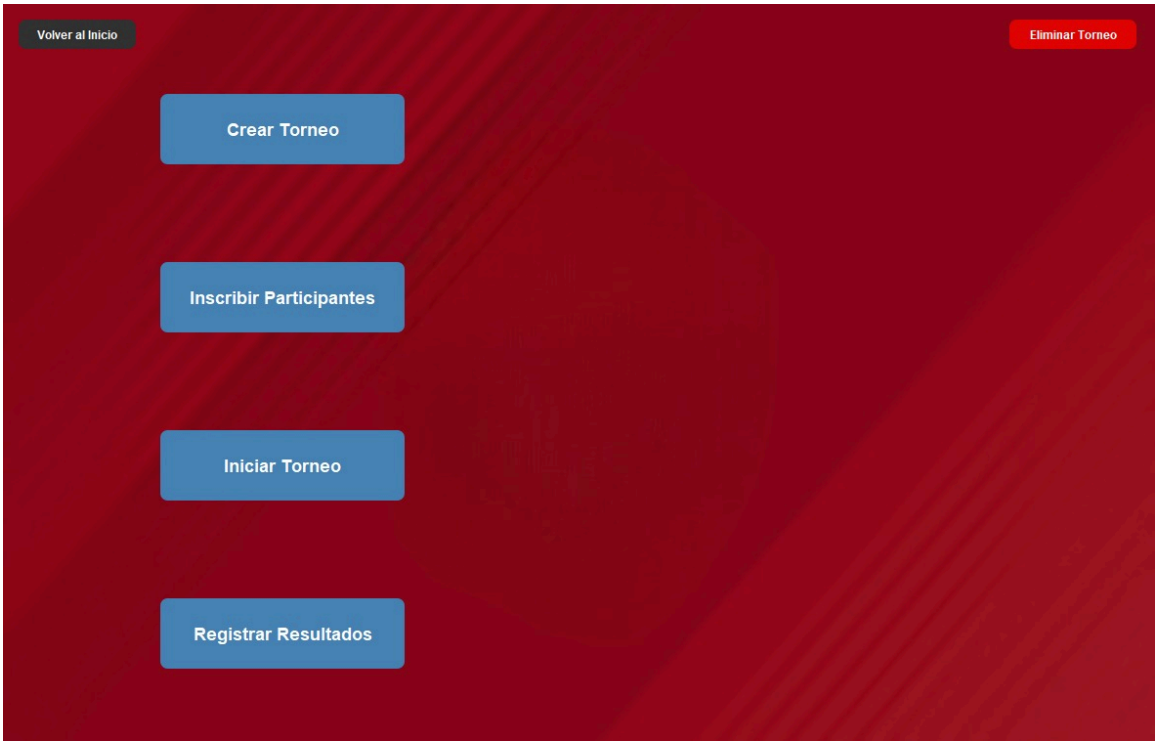


Captura de pantalla de la interfaz

Panel Inicial



Panel Organizador



Panel Usuario

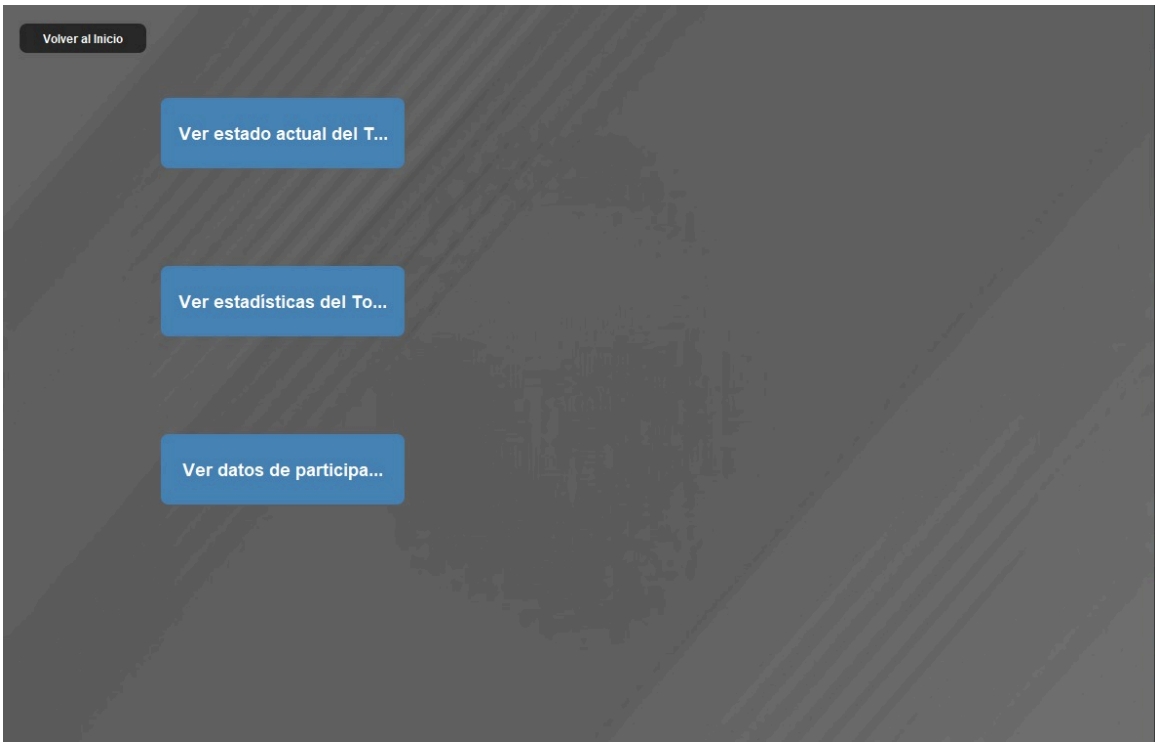
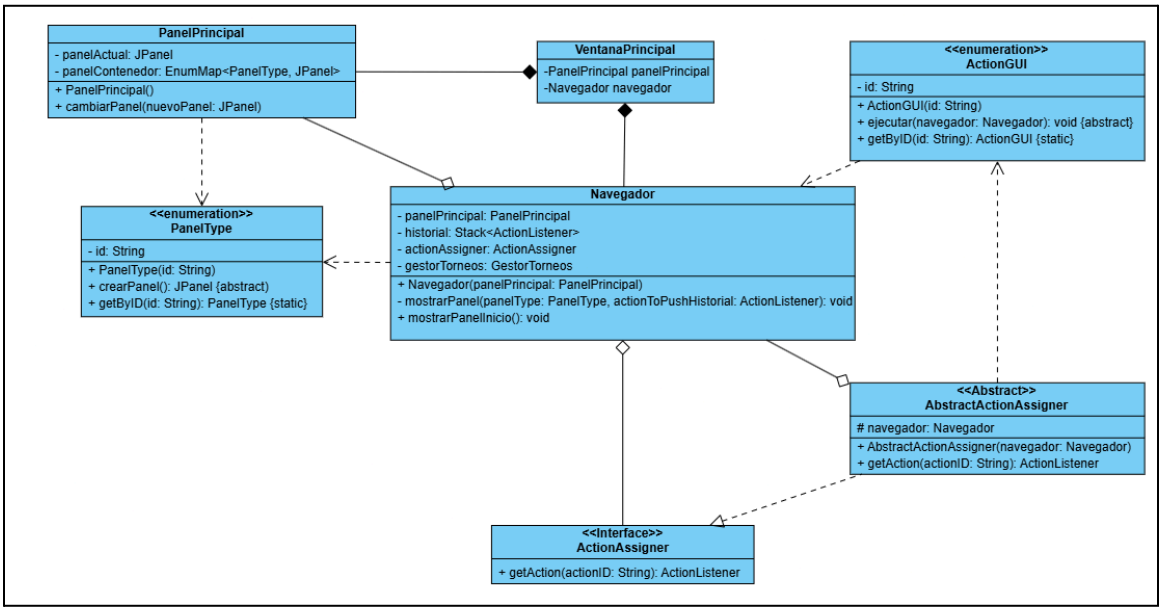
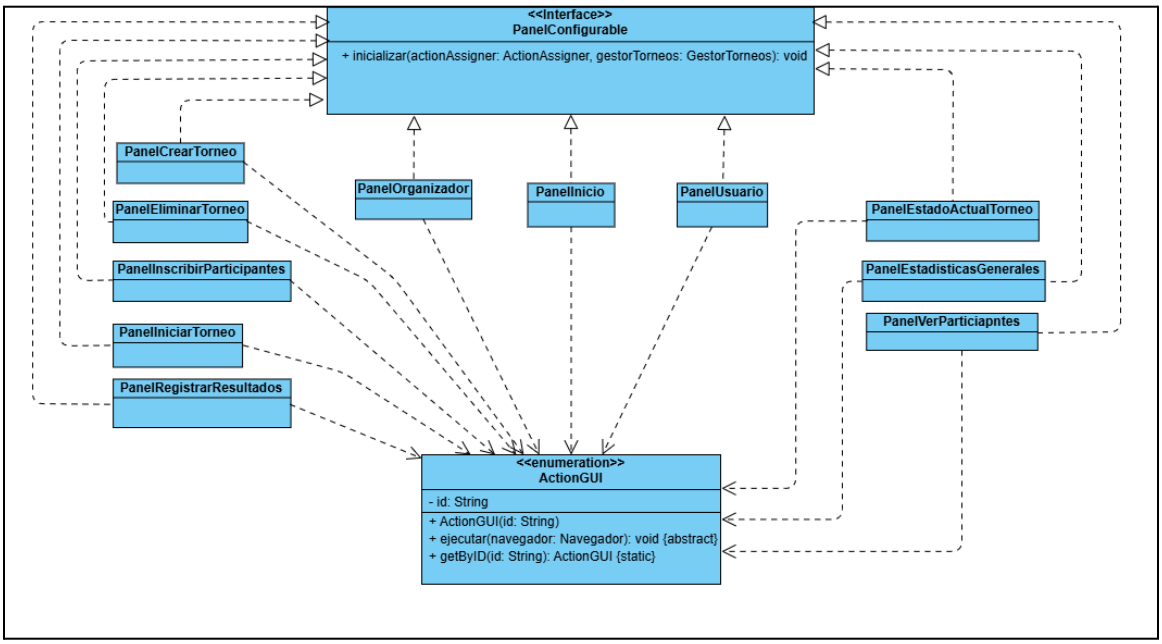


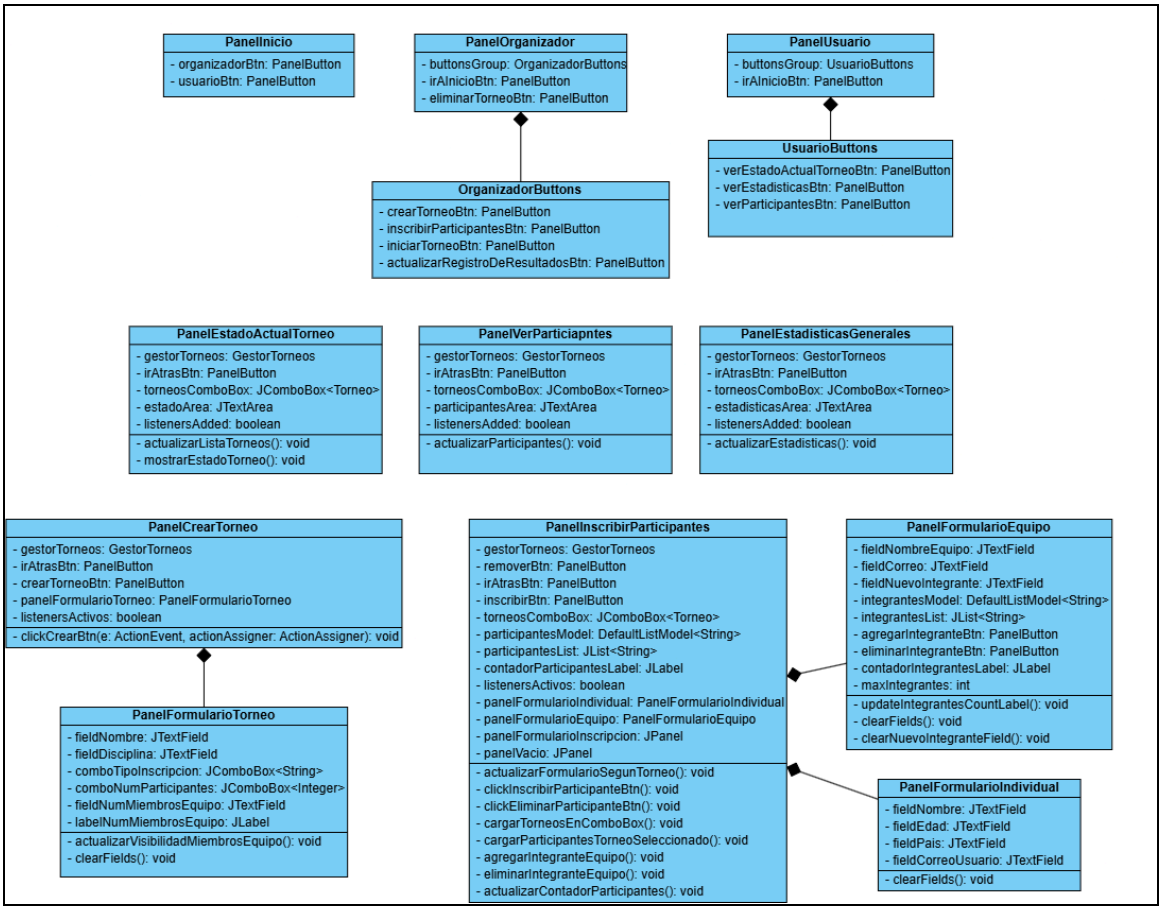
Diagrama de Clases UML

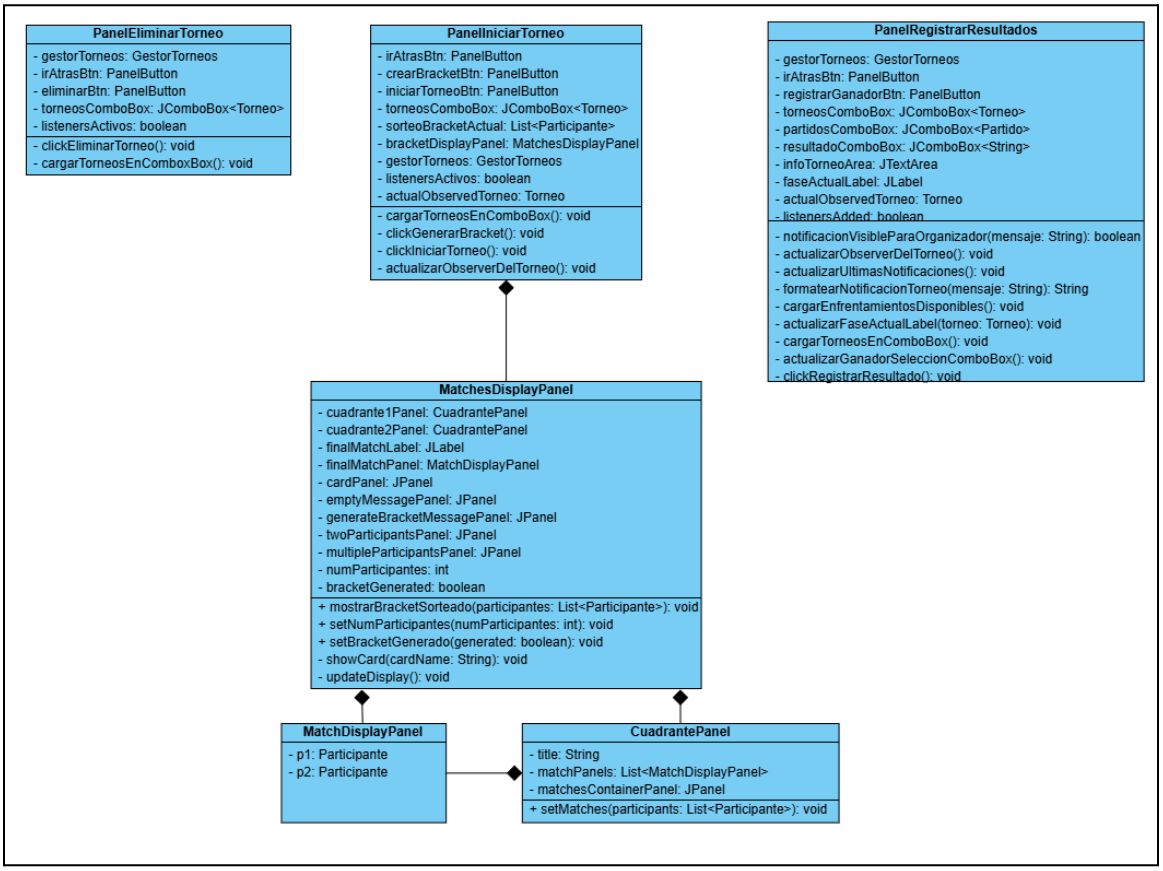
Nota: el diagrama UML de la GUI fue separado en 4 partes para mayor practicidad, ya que si se unía completa no era posible entenderla debido a la cantidad de elementos.

GUI:

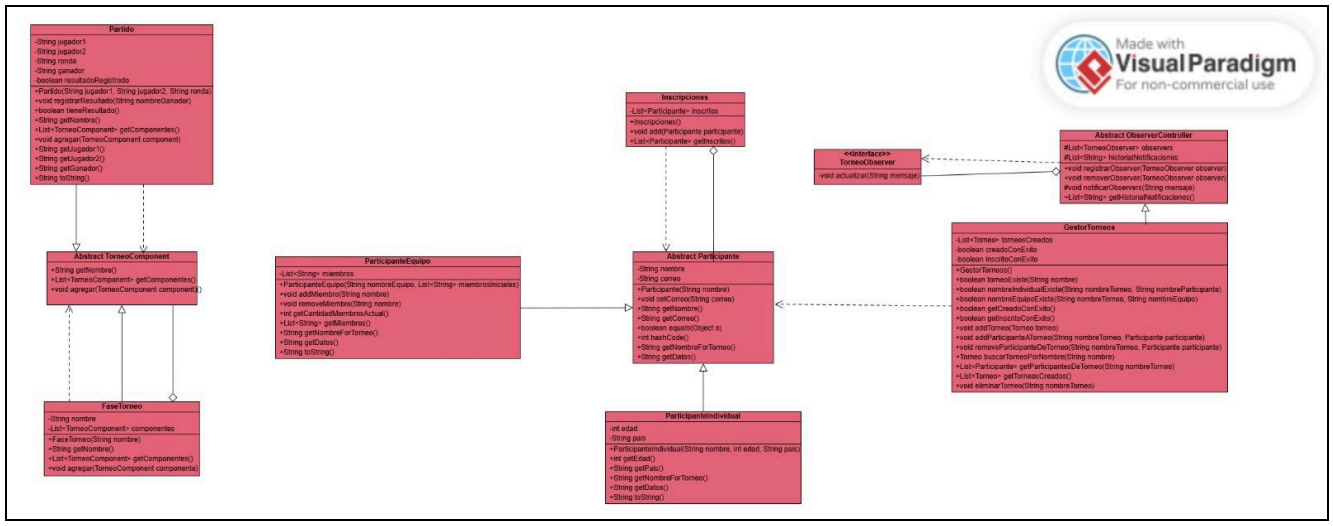





PanelInicio
- organizadorBtn: PanelButton
- usuarioBtn: PanelButton
PanelOrganizador
- buttonsGroup: OrganizadorButtons
- irAInicioBtn: PanelButton
- eliminarTorneoBtn: PanelButton
PanelUsuario
- buttonsGroup: UsuarioButtons
- irAInicioBtn: PanelButton
OrganizadorButtons
- crearTorneoBtn: PanelButton
- inscribirParticipantesBtn: PanelButton
- iniciarTorneoBtn: PanelButton
- actualizarRegistroDeResultadosBtn: PanelButton
UsuarioButtons
- verEstadoActualTorneoBtn: PanelButton
- verEstadisticasBtn: PanelButton
- verParticipantesBtn: PanelButton
PanelEstadoActualTorneo
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- estadoArea: JTextArea
- listenersAdded: boolean
- actualizarListaTorneos(): void
- mostrarEstadoTorneo(): void
PanelVerParticiapntes
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- participantesArea: JTextArea
- listenersAdded: boolean
- actualizarParticipantes(): void
PanelEstadisticasGenerales
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- estadisticasArea: JTextArea
- listenersAdded: boolean
- actualizarEstadisticas(): void
PanelCrearTorneo
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- crearTorneoBtn: PanelButton
- panelFormularioTorneo: PanelFormularioTorneo
- listenersActivos: boolean
- clickCrearBtn(e: ActionEvent, actionAssigner: ActionAssigner): void
PanelFormularioTorneo
- fieldNombre: JTextField
- fieldDisciplina: JTextField
- comboTipoInscripcion: JComboBox<String>
- comboNumParticipantes: JComboBox<Integer>
- fieldNumMiembrosEquipo: JTextField
- labelNumMiembrosEquipo: JLabel
- actualizarVisibilidadMiembrosEquipo(): void
- clearFields(): void
PanelInscribirParticipantes
- gestorTorneos: GestorTorneos
- removerBtn: PanelButton
- irAtrasBtn: PanelButton
- inscribirBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- participantesModel: DefaultListModel<String>
- participantesList: JList<String>
- contadorParticipantesLabel: JLabel
- listenersActivos: boolean
- panelFormularioIndividual: PanelFormularioIndividual
- panelFormularioEquipo: PanelFormularioEquipo
- panelFormularioInscripcion: JPanel
- panelVacio: JPanel
- actualizarFormularioSegunTorneo(): void
- clickInscribirParticipanteBtn(): void
- clickEliminarParticipanteBtn(): void
- cargarTorneosEnComboBox(): void
- cargarParticipantesTorneoSeleccionado(): void
- agregarIntegranteEquipo(): void
- eliminarIntegranteEquipo(): void
- actualizarContadorParticipantes(): void
PanelFormularioEquipo
- fieldNombreEquipo: JTextField
- fieldCorreo: JTextField
- fieldNuevoIntegrante: JTextField
- integrantesModel: DefaultListModel<String>
- integrantesList: JList<String>
- agregarIntegranteBtn: PanelButton
- eliminarIntegranteBtn: PanelButton
- contadorIntegrantesLabel: JLabel
- maxIntegrantes: int
- updateIntegrantesCountLabel(): void
- clearFields(): void
- clearNuevoIntegranteField(): void
PanelFormularioIndividual
- fieldNombre: JTextField
- fieldEdad: JTextField
- fieldPais: JTextField
- fieldCorreoUsuario: JTextField
- clearFields(): void


PanelEliminarTorneo
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- eliminarBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- listenersActivos: boolean
- clickEliminarTorneo(): void
- cargarTorneosEnComboxBox(): void
PanelIniciarTorneo
- irAtrasBtn: PanelButton
- crearBracketBtn: PanelButton
- iniciarTorneoBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- sorteoBracketActual: List<Participante>
- bracketDisplayPanel: MatchesDisplayPanel
- gestorTorneos: GestorTorneos
- listenersActivos: boolean
- actualObservedTorneo: Torneo
- cargarTorneosEnComboBox(): void
- clickGenerarBracket(): void
- clickIniciarTorneo(): void
- actualizarObserverDelTorneo(): void
PanelRegistrarResultados
- gestorTorneos: GestorTorneos
- irAtrasBtn: PanelButton
- registrarGanadorBtn: PanelButton
- torneosComboBox: JComboBox<Torneo>
- partidosComboBox: JComboBox<Partido>
- resultadoComboBox: JComboBox<String>
- infoTorneoArea: JTextArea
- faseActualLabel: JLabel
- actualObservedTorneo: Torneo
- listenersAdded: boolean
- notificacionVisibleParaOrganizador(mensaje: String): boolean
- actualizarObserverDelTorneo(): void
- actualizarUltimasNotificaciones(): void
- formatearNotificacionTorneo(mensaje: String): String
- cargarEnfrentamientosDisponibles(): void
- actualizarFaseActualLabel(torneo: Torneo): void
- cargarTorneosEnComboBox(): void
- actualizarGanadorSeleccionComboBox(): void
- clickRegistrarResultado(): void
MatchesDisplayPanel
- cuadrante1Panel: CuadrantePanel
- cuadrante2Panel: CuadrantePanel
- finalMatchLabel: JLabel
- finalMatchPanel: MatchDisplayPanel
- cardPanel: JPanel
- emptyMessagePanel: JPanel
- generateBracketMessagePanel: JPanel
- twoParticipantsPanel: JPanel
- multipleParticipantsPanel: JPanel
- numParticipantes: int
- bracketGenerated: boolean
+ mostrarBracketSorteado(participantes: List<Participante>): void
+ setNumParticipantes(numParticipantes: int): void
+ setBracketGenerado(generated: boolean): void
- showCard(cardName: String): void
- updateDisplay(): void
MatchDisplayPanel
- p1: Participante
- p2: Participante
CuadrantePanel
- title: String
- matchPanels: List<MatchDisplayPanel>
- matchesContainerPanel: JPanel
+ setMatches(participants: List<Participante>): void

Lógica:



Patrones de diseño implementados

Patron Observer:

Propósito: Este patrón fue implementado para mantener una comunicación eficiente entre los componentes del sistema (principalmente entre el modelo y la vista) sin crear un acoplamiento fuerte. Permite que múltiples paneles GUI (observers) reciban actualizaciones automáticas cuando cambia el estado de los torneos.

Clases involucradas:

- ***`ObserverController`***
- ***`TorneoObserver`***
- ***`Torneo`***
- ***`GestorTorneos`***
- ***`PanelInscribirParticipantes`***
- ***`PanelEstadoActualTorneo`***
- ***`PanelEliminarTorneo`***
- ***`PanelRegistrarResultados`***
- ***`PanelEstadisticasGenerales`***
- ***`PanelVerParticipantes`***
- ***`PanelCrearTorneo`***

- ***PanelIniciarTorneo***
- ***TorneoObserver***

Patron Composite:

Propósito: Aplicado para representar la estructura jerárquica de los torneos, fases y partidos, permitiendo tratar objetos individuales y composiciones de objetos de manera uniforme.

Clases involucradas:

- **`TorneoComponent`**
- **`FaseTorneo`**
- **`Partido`**

Patron Singleton:

Propósito: Aplicado implícitamente en clases como **`GestorTorneos`** y **`Navegador`** donde se necesita una única instancia accesible globalmente en la aplicación.

Clases involucradas:

- **`GestorTorneos`**
- **`Navegador`**

Patron Strategy :

Propósito: Implementado para encapsular diferentes algoritmos de navegación (cambio entre paneles) y permitir que estos algoritmos varíen independientemente de los clientes que los usan.

Clases involucradas:

- **`Participante`**
- **`ParticipanteIndividual`**
- **`ParticipanteEquipo`**
- **`ActionGUI`**
- **`Navegador`**

- `ActionAssigner`

Patron Factory Method:

Propósito: Se utilizó para crear diferentes tipos de paneles de manera flexible y desacoplada. El enum `PanelType` actúa como fábrica, encapsulando la lógica de creación de cada tipo de panel.

Clases involucradas:

- `PanelType`
- `PanelInicio`
- `PanelUsuario`
- `PanelOrganizador`
- `PanelCrearTorneo`
- `PanelInscribirParticipantes`
- `PanelEstadoActualTorneo`
- `PanelIniciarTorneo`
- `PanelRegistrarResultados`
- `PanelEstadisticasGenerales`
- `PanelVerParticipantes`
- `PanelEliminarTorneo`

Patron Command:

Propósito: Se utilizó para encapsular solicitudes como objetos, permitiendo parametrizar clientes con diferentes solicitudes, hacer cola o registrar solicitudes, y soportar operaciones que pueden deshacerse.

Clases involucradas:

- `ActionGUI`
- `ActionAssigner`
- `AbstractActionAssigner`
- `Navegador`

Decisiones importantes del equipo

Durante el avance y nuevas implementaciones en la GUI, se buscó patrones de repetición DRY, los que dio lugar a la clase/patrón Navegador y a la interfaz ActionAssigner.

Este enfoque surgió de la necesidad de gestionar múltiples paneles a través de un historial, idea que nace a partir del diagrama de casos de usos y el requisito de manejar las interacciones entre organizador y usuario.

Una vez pulido el problema sobre la visibilidad entre las vistas/actores, pudimos dar paso a emplear la separación de responsabilidades entre la lógica y la GUI.

De esta forma logramos hacer que la GUI sea únicamente la “representación visual” y lógica la parte “interna” que debía funcionar con o sin GUI. Además, cualquier implementación visual que se fuese añadiendo solo debía implementarse al navegador como una “nueva pestaña” a la que se puede acceder, un ejemplo de esto fue la opción de “eliminar un torneo”, donde solo se implemento una nueva funcionalidad lógica encargada de esto y luego la GUI solo integró un nuevo panel donde navegar, el cual solo debió integrar la funcionalidad prescrita.

Problemas identificados y autocrítica

PanelConfigurable (Interfaz)

Su propósito original se fue desbordando, dando pie a la casi nula reutilización en paneles que se diferenciaban únicamente en su diseño, pues compartían mismos atributos e incluso métodos/funcionalidades. Esto fue detectado como una repetición de código, pero no se le dio la misma importancia como a otros casos(por ej. el Navegador que a criterio nuestro fue un acierto) ya que internamente sabíamos que había un límite de funcionalidades a integrar, pero si se quisieran agregar más funcionalidades, esto hubiese sido directamente insostenible.

Solución pensada: PanelConfigurable (Clase Abstracta)

De haberse implementado esta modificación en el momento en que nos percatamos del problema, habríamos garantizado desde el inicio una estructura mucho mejor definida y ordenada para todos los paneles de la aplicación.

Esta idea de refactorización surgió al momento de implementar la lógica específica en cada panel, pues cada panel accedía a clases y/o métodos de la lógica de la misma forma, solo que para cumplir distintos propósitos.

Reflexion:

A nuestro parecer logramos que la GUI funcionara puramente como la forma de presentación, algo que no fue logrado en la tarea 3 y por ello le dimos una gran importancia en este proyecto. Nos aseguramos de que cualquier regla/cambio drástico en la lógica no se viese afectado en la interfaz y viceversa, y esto lo pudimos notar conforme fuimos diseñando esta, pues surgieron ideas de mejora para la lógica y estas no influyen en ningún momento, pues solo deben ser implementadas.