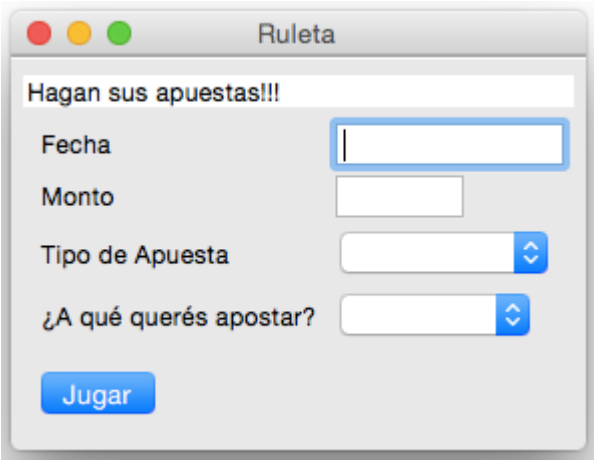
Punto 1)

Dada la siguiente interfaz generada por el programa:



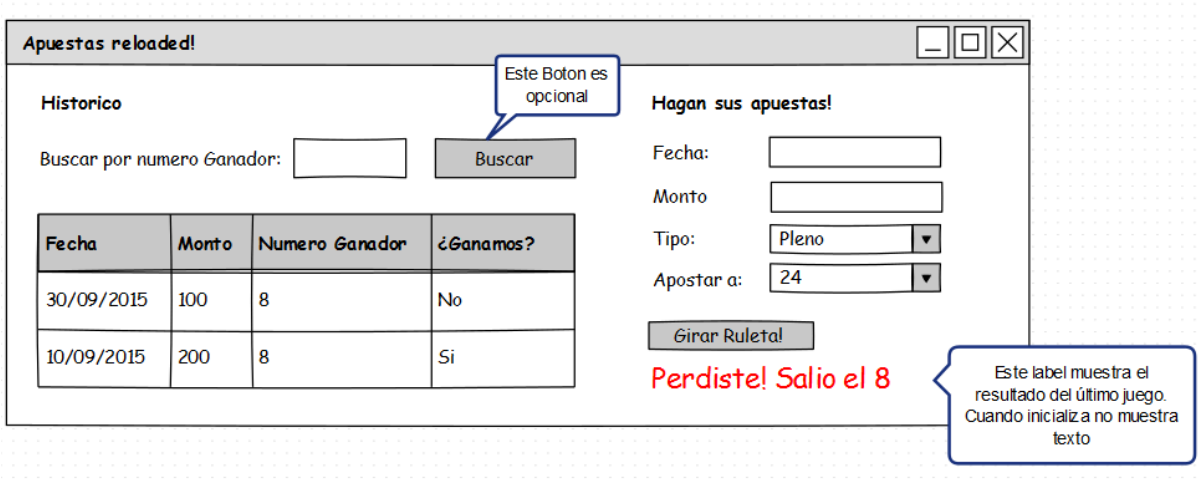

```
@Observable
@Accessors
class Apuesta {
	Date fecha
	BigDecimal monto
	TipoApuesta tipo
	Object valorApostado
	Resultado resultado

	val hoy = new Date()
	def isPuedeJugar() { fecha != null
&& monto != null && monto > BigDecimal.ZERO }

	def void setFecha(Date unaFecha) {
		this.fecha = unaFecha
		firePropertyChanged(this, "puedeJugar", puedeJugar)
	}

	def void setMonto(BigDecimal unMonto) {
		if (unMonto == null || unMonto <= BigDecimal.ZERO)
			throw new UserException("El monto debe ser positivo.")
		this.monto = unMonto
		firePropertyChanged(this, "puedeJugar", puedeJugar)
	}

	def jugar() {
		tipo.validarMontoMinimo(monto)
		val ganador = new Random().nextInt(37)
		resultado = tipo.chequearApuesta(ganador, this)
	}
	def getTiposPosibles() { #[new ApuestaPleno, new ApuestaDocena] }
}

@Accessors
abstract class Resultado {
	int ganador
	new(int ganador) { this.ganador = ganador }
}

class Ganador extends Resultado {
	BigDecimal montoGanado
	new(int ganador, BigDecimal montoGanado) {
		super(ganador)
		this.montoGanado = montoGanado
	}
	override toString() {'''Ganaste $«montoGanado»'''  }
}
```

```
class Perdedor extends Resultado {
	new(int ganador) {super(ganador)}
	override toString() { '''Perdiste, salió el «ganador»'''}
}

@Observable
abstract class TipoApuesta {
	override toString() {this.class.simpleName.substring(7)}
	def List<Object> getValoresPosibles()
	def validarMontoMinimo(BigDecimal monto) {
		if (monto < new BigDecimal(montoMinimo)) {
			throw new UserException('''El monto minimo para una apuesta «this» es
«montoMinimo»''')
		}
	}
	def int montoMinimo()
	def boolean esGanador(int ganador, Object valorApostado)
	def int ganancia()
	def Resultado chequearApuesta(int ganador, Apuesta apuesta) {
		if (esGanador(ganador, apuesta.valorApostado))
			new Ganador(ganador, apuesta.monto * new BigDecimal(ganancia))
		else new Perdedor(ganador)
	}

	def asObjects(List<?> list) { list.map[it as Object]}
}

class ApuestaPleno extends TipoApuesta {
	override getValoresPosibles() { (1 .. 36).toList.asObjects }
	override montoMinimo() {10}
	override ganancia() {35}
	override esGanador(int ganador, Object valorApostado) { ganador == valorApostado }
}

class ApuestaDocena extends TipoApuesta {
	val docenas = #["Primera", "Segunda", "Tercera"]
	override getValoresPosibles() { docenas.asObjects }
	override montoMinimo() {50}
	override ganancia() {11}
	override esGanador(int ganador, Object valorApostado) {
		val docena = docenas.indexOf(valorApostado)
		val min = docena * 12 + 1
		val max = (docena + 1) * 12
		(min .. max).contains(ganador)
	}
}

class CrearApuestaWindow extends SimpleWindow<Apuesta> {
	new(WindowOwner owner, Apuesta apuesta) {
		super(owner, apuesta)
		title = "Ruleta"
		taskDescription = "Hagan sus apuestas!!!"
	}
```

```
	override createFormPanel(Panel mainPanel) {
		val editorPanel = new Panel(mainPanel)
		editorPanel.layout = new ColumnLayout(2)

		new Label(editorPanel).text="Fecha"
		new TextBox(editorPanel) => [
			withFilter(new DateTextFilter)
			bindValueToProperty("fecha").transformer = new DateAdapter
		]

		new Label(editorPanel).text="Monto"
		new TextBox(editorPanel) => [
			val bindingMonto = bindValueToProperty("monto")
			bindingMonto.transformer = new BigDecimalTransformer
		]

		new Label(editorPanel).setText("Tipo de Apuesta")
		new Selector(editorPanel) => [
			allowNull = false
			bindItemsToProperty("tiposPosibles")
			bindValueToProperty("tipo")
		]

		new Label(editorPanel).setText("¿A qué querés apostar?")
		new Selector(editorPanel) => [
			bindItemsToProperty("tipo.valoresPosibles")
			bindValueToProperty("valorApostado")
		]
	}

	override addActions(Panel actionsPanel) {
		new Button(actionsPanel) => [
			caption = "Jugar"
			onClick [ | jugar ]
			bindEnabledToProperty("puedeJugar")
		]
		new Label(actionsPanel).bindValueToProperty("resultado")

	def jugar() {
		modelObject.jugar
		showInfo(modelObject.resultado.toString)
	}
}

class DateBox extends TextBox {
	new(Panel container) {super(container)}

override bindValueToProperty(String propertyName) {
		val binding = super.bindValueToProperty(propertyName)
		this.withFilter(new DateTextFilter)
		binding
	}
}
```

Se pide

Modificar la solución de forma que permita guardar el historial de apuestas realizadas. Debe de lograrse la siguiente interfaz de usuario:



En el diseño de la solución que implemente debe explicar:

- Objeto elegido como modelo de la vista y por qué
- Mostrar código de creacion de la vista.

NOTAS:

- NO es obligatorio definir el código que muestre el layout tal cual lo muestra la imagen, se puede identificar sobre el mismo mock up
- Si el código que vamos a crear es similar al actual no es necesario volver a escribirlo en su totalidad, pueden hacer referencia al código del enunciado
- El botón Buscar es opcional, pueden implementar la búsqueda al escribir en el input. Pueden agregar un botón de Clear si necesitan.
- Tips!
    - Recuerden que para crear un componente Tabla que muestra datos: `new Table<TipoDeDato>(panel, typeof(TipoDeDato))`
    - Crear una Columna: `new Column<TipoDeDato>(tabla)`
    - Mensajes que bindean contenido:
        - A la tabla: `bindItemsToProperty`
        - A la columna: `bindContentsToProperty` o `bindContentsToTransformer`

Punto 2)

Opine sobre el siguiente código en términos de separación de responsabilidades dentro del patrón MVC y lo visto en clase, haga hincapié en el modelo elegido para la vista. En caso de proponer cambios enuncielos o implementelos.

```
class SeguimientoDeCarreraWindow extends SimpleWindow<Materia> {
	new(WindowOwner parent) {
		super(parent, new Materia())
	}
```

```
	override protected createFormPanel(Panel mainPanel) {
		new Label(mainPanel).text = "Seguidor de Carrera"
		this.crearListadoDeMaterias(mainPanel)
		this.crearEdicionDeMateria(mainPanel)
	}

	def crearEdicionDeMateria(Panel panel) {
		new Label(panel).text = "Nombre"
		new TextBox(panel).bindValueToProperty("seleccionada.nombreMateria")

		new Label(panel).text="Aprobo:"
		new CheckBox(panel).bindValueToProperty("seleccionada.estaAprobada")
	}

	def crearListadoDeMaterias(Panel panel) {
		new Label(panel).text = "Materias"
		new List<Materia>(panel) => [
				bindItemsToProperty("materiasRepository.all")
				bindValueToProperty("seleccionada")
			]
		}

	override protected addActions(Panel panel) {
		new Button(panel) =>[
			caption = "Nueva Materia"
			onClick = [ |
				var nueva = MateriasRepository.instance.newMateria
				MateriasRepository.instance.add(nueva)
				modelObject.seleccionada = nueva
			]
		]
	}
@Observable
@Accessors
class Materia {
	private var Materia seleccionada
	private var String nombreMateria
	private var boolean estaAprobada
	def MateriasRepository getMateriasRepository() {MateriasRepository.instance	}
}
@Observable
@Accessors
class MateriasRepository {
	var static MateriasRepository instance;
	var List<Materia> all = newArrayList
	def static getInstance(){
		if(instance == null){ instance = new MateriasRepository() }
		instance
	}
	def newMateria() {new Materia()}
	def add(Materia materia) {all.add(materia)}
}
```