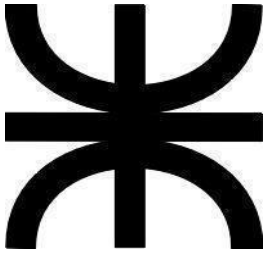
Universidad Tecnológica Nacional
Facultad Regional Córdoba



Ingeniería en Sistemas de Información
Cátedra: Ingeniería y Calidad de Software

Trabajo Práctico Nº13 - TDD

Curso: 4K4

Grupo N°2:

85250 - Paloma Candelaria Corcoba

85530 - Joaquin Miranda Oliveros

85308 - Juan Ignacio Camargo Mano

85832 - Consuelo Cordoba Oyhamburu

90522 - Pedro Placci

78493 - Genoves Micheli

93050 - Mateo Ugarte Torres

89335 - Delfina Brenda Glavas

89302 - Gianella Bryanna Magliano

85518 - Marcio Joel Saravia

Docentes:

Ing. Laura Covaro

Ing. Mickaela Crespo

Ing. Constanza Garnero

Grupo número par (2-dos), US “Inscribirme a actividad”

- **Test 1: Inscribirse correctamente a una actividad con cupo disponible (pasa)**

🔴 RED:

```
func test_inscripcion_exitosa() {
  actividad = "Safari"
  fechaElegida = "29/06/2025"
  horario = "10:00"
  visitante = {nombre: "Ana", dni: "12345678", edad: 25, talle: "M"}
  acepta_terminos = true
  resultado = null

  try {
    resultado = inscribirseActividad(actividad, horario, visitantes,
fechaElegida, acepta_terminos)
  } catch (Exception ex) {}

  assert.equals(resultado, "Inscripción exitosa")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, fechaElegida,
acepta_terminos) {
    if (actividad == "Safari" && horario == "10:00" && fechaElegida ==
    "29/06/2025" && acepta_terminos) {
         return "Inscripción exitosa"
    }
}
```

🔵 REFACTOR:

```
 func inscribirseActividad(actividad, horario, visitante, fechaElegida,
 acepta_terminos) {
   if (!actividadesDisponibles.includes(actividad))
     throw ActividadInvalidaException("Actividad no disponible.")

   if (!fechasDisponibles[actividad].includes(fechaElegida))
     throw CupoNoDisponibleException("Fecha no disponible.")

   if (!horariosDisponibles[actividad].includes(horario))
     throw CupoNoDisponibleException("No hay cupo en el horario.")
```

```
  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  // Si la actividad requiere talle y no se ingresó
  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 2: Intentar inscribirse sin cupo en ese horario (falla)**

🔴 RED:

```
func test_sin_cupo_en_horario() {
  actividad = "Palestra"
  horario = "14:00"  // no disponible
  visitante = {...}
  acepta_terminos = true
  try {
    inscribirseActividad(actividad, horario, visitante, acepta_terminos)
  } catch (Exception ex) {}

  assert.equals(ex, "No hay cupo en el horario")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (actividad == "Palestra" && horario == "14:00")
    throw Exception("No hay cupo en el horario")
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!cuposDisponibles(actividad, horario))
    throw Exception("No hay cupo en el horario")
}
```

- **Test 3: Actividad no requiere talle y no se ingresa (pasa)**

🔴 RED:

```
func test_sin_talle_si_no_se_requiere() {
  actividad = "Palestra"  // no requiere talle
  horario = "10:00"
  visitante = {nombre: "Juan", dni: "87654321", edad: 30, talle: null}
  acepta_terminos = true

  resultado = null
  try {
    resultado = inscribirseActividad(actividad, horario, visitante,
acepta_terminos)
  } catch (Exception ex) {}

  assert.equals(resultado, "Inscripción exitosa")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (actividad == "Palestra" && horario == "10:00" && acepta_terminos &&
visitante.talle == null)
    return "Inscripción exitosa"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!acepta_terminos)
    throw Exception("Debe aceptar los términos")

  if (!cuposDisponibles(actividad, horario))
    throw Exception("No hay cupo en el horario")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw Exception("Falta el talle requerido")

  // Si la actividad no requiere talle, no importa que sea null
  return "Inscripción exitosa"
}
```

- **Test 4: Actividad que requiere talle y no se ingresa (falla)**

🔴 RED:

```
func test_falta_talle_requerido() {
  actividad = "Tirolesa"
  horario = "11:00"
  visitante = {nombre: "Carlos", dni: "11223344", edad: 22, talle: null}
  acepta_terminos = true

  try {
    inscribirseActividad(actividad, horario, visitante, acepta_terminos)
  } catch (Exception ex) {}

  assert.equal(ex, "Falta el talle requerido")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (actividad == "Tirolesa" && visitante.talle == null)
    throw Exception("Falta el talle requerido")
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!acepta_terminos)
    throw Exception("Debe aceptar los términos")

  if (!cuposDisponibles(actividad, horario))
    throw Exception("No hay cupo en el horario")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw Exception("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 5: No acepta términos (falla)**

🔴 RED:

```
func test_no_acepta_terminos() {
  actividad = "Jardinería"
  horario = "13:00"
  visitante = {nombre: "Lucía", dni: "44556677", edad: 35, talle: null}
  acepta_terminos = false

  try {
    resultado = inscribirseActividad(actividad, horario, visitante,
acepta_terminos)
  } catch (Exception ex) {}

  assert.equal(resultado, "Debe aceptar los términos")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!acepta_terminos)
    throw Exception("Debe aceptar los términos")
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!acepta_terminos)
    throw Exception("Debe aceptar los términos")

  if (!cuposDisponibles(actividad, horario))
    throw Exception("No hay cupo en el horario")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw Exception("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 6: Seleccionar horario en el que el parque está cerrado (falla)**

🔴 RED:

```
func test_horario_fuera_de_rango() {
  actividad = "Safari"
  horario = "23:00"  // horario inválido
  visitante = {nombre: "Nico", dni: "99887766", edad: 27, talle: "M"}
  acepta_terminos = true

  try {
    inscribirseActividad(actividad, horario, visitante, acepta_terminos)
  } catch (Exception ex) {}

  assert.equal(ex, "Horario no disponible")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (horario == "23:00")
    throw Exception("Horario no disponible")
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!acepta_terminos)
    throw Exception("Debe aceptar los términos")

  if (!horarioValido(horario))
    throw Exception("Horario no disponible")

  if (!cuposDisponibles(actividad, horario))
    throw Exception("No hay cupo en el horario")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw Exception("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 7: Inscribirse a una actividad con múltiples participantes (pasa)**

🔴 RED:

```
func test_inscripcion_varios_participantes() {
  actividad = "Safari"
  horario = "10:00"
  participantes = [
    {nombre: "Ana", dni: "12345678", edad: 25, talle: "M"},
    {nombre: "Luis", dni: "23456789", edad: 28, talle: "L"}
  ]
  acepta_terminos = true

  resultado = inscribirseActividad(actividad, horario, participantes,
acepta_terminos)

  assert.equals(resultado, "Inscripción exitosa")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, participantes,
acepta_terminos) {
  if (actividad == "Safari" && horario == "10:00" && acepta_terminos) {
    return "Inscripción exitosa"
  }
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, participantes,
acepta_terminos) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  for (visitante in participantes) {
```

```
    if (requiereTalle(actividad) && visitante.talle == null)
      throw TalleFaltanteException("Falta el talle requerido")
  }

  return "Inscripción exitosa"
}
```

-

- **Test 8: Intentar inscribirse con edad negativa (falla)**

🔴 RED:

```
func test_edad_invalida() {
  actividad = "Jardinería"
  horario = "11:00"
  visitante = {nombre: "Pepe", dni: "11112222", edad: -5, talle: null}
  acepta_terminos = true

  try {
    inscribirseActividad(actividad, horario, visitante, acepta_terminos)
  } catch (Exception ex) {}

  assert.equals(ex, "Edad inválida")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (visitante.edad < 0)
    throw EdadInvalidaException("Edad inválida")

  return "Inscripción exitosa"
}
```

🔵 REFACTOR (con validación integrada):

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitante.edad < 0 || visitante.edad > 120)
    throw EdadInvalidaException("Edad inválida")
```

```
  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 9: Intentar inscribirse sin ingresar DNI (falla)**

🔴 RED:

```
func test_dni_faltante() {
  actividad = "Tirolesa"
  horario = "13:00"
  visitante = {nombre: "Sofía", dni: null, edad: 18, talle: "S"}
  acepta_terminos = true

  try {
    inscribirseActividad(actividad, horario, visitante, acepta_terminos)
  } catch (Exception ex) {}

  assert.equals(ex, "Falta DNI del visitante")
}
```

🟢 GREEN:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (visitante.dni == null)
    throw DniFaltanteException("Falta DNI del visitante")

  return "Inscripción exitosa"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")
```

```
  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitante.dni == null || visitante.dni == "")
    throw DniFaltanteException("Falta DNI del visitante")

  if (visitante.edad < 0 || visitante.edad > 120)
    throw EdadInvalidaException("Edad inválida")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")

  return "Inscripción exitosa"
}
```

- **Test 10: Inscribirse con fecha anterior a la actual (falla)**

🔴 RED:

```
func test_fecha_anterior() {
  actividad = "Safari"
  horario = "10:00"
  fecha = "2023-01-01"
  visitante = {nombre: "Ana", dni: "12345678", edad: 25, talle: "M"}
  acepta_terminos = true

  try {
    inscribirseActividad(actividad, horario, [visitante], acepta_terminos,
fecha)
  } catch (Exception ex) {}

  assert.equals(ex, "Fecha inválida")
}
```

🟢 GREEN:

```
func inscribirseActividad(fecha){
   if (fecha < hoy())
     throw FechaInvalidaException("Fecha inválida")

   return "Inscripción exitosa"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos,
fecha) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitante.dni == null || visitante.dni == "")
    throw DniFaltanteException("Falta DNI del visitante")

  if (visitante.edad < 0 || visitante.edad > 120)
    throw EdadInvalidaException("Edad inválida")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")


  if (fecha < hoy())
    throw FechaInvalidaException("Fecha inválida")


  return "Inscripción exitosa"
}
```

- **Test 11: Inscribirse con DNI en formato no válido (falla)**

🔴 RED:

```
func test_dni_invalido() {
  visitante = {nombre: "Ana", dni: "ABC123", edad: 25, talle: "M"}
  try {
    inscribirseActividad("Safari", "10:00", visitante, true)
  } catch (Exception ex) {}

  assert.equals(ex, "DNI inválido")
}
```

🟢 GREEN:

```
func inscribirseActividad(visitante){
  if (!dniValido(visitante.dni))
    throw DniInvalidoException("DNI inválido")

  return "Inscripción exitosa"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos,
fecha, dni) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitante.dni == null || visitante.dni == "")
    throw DniFaltanteException("Falta DNI del visitante")

  if (visitante.edad < 0 || visitante.edad > 120)
    throw EdadInvalidaException("Edad inválida")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")


  if (fecha < hoy())
    throw FechaInvalidaException("Fecha inválida")


 if (!dniValido(dni))
   throw DniInvalidoException("Dni en formato no válido")

  return "Inscripción exitosa"
}
```

- **Test 12: Inscribirse sin seleccionar horario (falla)**

🔴 RED:

```
func test_sin_horario() {
   visitante = {nombre: "Ana", dni: "12345678", edad: 25, talle: "M"}
   try {
     inscribirseActividad("Safari", null, visitante, true)
   } catch (Exception ex) {}

   assert.equals(ex, "Debe seleccionar un horario")
 }
```

🟢 GREEN:

```
func inscribirseActividad(horario){
    if (horario == null)
      throw HorarioFaltanteException("Debe seleccionar un horario")

    return "Inscripción exitosa"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitante, acepta_terminos,
fecha, dni) {
  if (!actividadesDisponibles.includes(actividad))
    throw ActividadInvalidaException("Actividad no disponible")

  if (horario == null)
    throw HorarioFaltanteException("Debe seleccionar un horario")

  if (!horariosDisponibles[actividad].includes(horario))
    throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
    throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitante.dni == null || visitante.dni == "")
    throw DniFaltanteException("Falta DNI del visitante")

  if (visitante.edad < 0 || visitante.edad > 120)
    throw EdadInvalidaException("Edad inválida")

  if (requiereTalle(actividad) && visitante.talle == null)
    throw TalleFaltanteException("Falta el talle requerido")
```

```
  if (fecha < hoy())
    throw FechaInvalidaException("Fecha inválida")

 if (!dniValido(dni))
   throw DniInvalidoException("Dni en formato no válido")

 return "Inscripción exitosa"
}
```

- **Test 13: Inscribirse a una actividad ingresando una cantidad de visitantes mayor a 10 (falla)**

🔴 RED:

```
func test_inscripcion_mas_de_diez_participantes() {
   actividad = "Tirolesa"
   horario = "17:00"
   acepta_terminos = true
   visitantes = []

   for (i = 1; i <= 11; i++) {
       visitantes.push({nombre: "Persona" + i, dni: "1000000" + i, edad: 25,
       talle: "M"})
   }

   try {
       inscribirseActividad(actividad, horario, visitantes, acepta_terminos)
   } catch(Exception ex) { }

   assert.equals(ex, "La cantidad de visitantes no puede exceder los 10.")
}
```

🟢 GREEN:

```
func inscribirseActividad(visitantes) {
  if (visitantes.length > 10)
     throw ExcepcionCantidadExcedida("La cantidad de visitantes no puede
     exceder los 10.")

  return "Inscripción aceptada"
}
```

🔵 REFACTOR:

```
func inscribirseActividad(actividad, horario, visitantes, acepta_terminos) {
  if (!actividadesDisponibles.includes(actividad))
      throw ActividadInvalidaException("Actividad no disponible")

  if (!horariosDisponibles[actividad].includes(horario))
      throw CupoNoDisponibleException("No hay cupo en el horario")

  if (!acepta_terminos)
      throw TerminosNoAceptadosException("Debe aceptar los términos")

  if (visitantes.length > 10)
      throw ExcepcionCantidadExcedida("La cantidad de visitantes no puede
      exceder los 10.")

  return "Inscripción exitosa"
}
```

Funciones Auxiliares

```
// Devuelve true si la actividad requiere talle
func requiereTalle(actividad) {
  return actividad == "Tirolesa" || actividad == "Safari"
}

// Devuelve true si hay cupo disponible para esa actividad y horario
func cuposDisponibles(actividad, horario) {
  return !(actividad == "Palestra" && horario == "14:00")  // ejemplo
}

// Devuelve true si el horario está dentro del rango de apertura del parque
func horarioValido(horario) {
  return horario >= "09:00" && horario <= "18:00"
}

// Devuelve true si el DNI tiene un formato válido (7 u 8 dígitos numéricos)
func dniValido(dni) {
  return dni.matches("^[0-9]{7,8}$")
}

// Devuelve la fecha actual (mockeada para testing)
func hoy() {
  return "2025-06-24"  // reemplazar por la fecha real del sistema si hace
falta
}

// Devuelve true si la actividad está disponible
func actividadesDisponibles() {
  return ["Safari", "Tirolesa", "Palestra", "Jardinería"]
}

// Devuelve true si el horario está disponible para una actividad
func horariosDisponibles(actividad) {
  return {
    "Safari": ["10:00", "11:00", "17:00"],
    "Tirolesa": ["11:00", "13:00", "17:00"],
    "Palestra": ["10:00"],
    "Jardinería": ["11:00", "13:00"]
  }[actividad]
}
```

```
// Devuelve true si la fecha está habilitada para la actividad
func fechasDisponibles(actividad) {
  return {
    "Safari": ["29/06/2025"],
    "Tirolesa": ["29/06/2025"],
    "Palestra": ["29/06/2025"],
    "Jardinería": ["29/06/2025"]
  }[actividad]
}
```