Sesión 01. Programación guiada por pruebas (`JUnit 5`)

Objetivos

- Crear la estructura adecuada de proyectos Java del alumno y la asignatura y vincularlos con los repositorios que correspondan.
- Utilizar la programación guiada por pruebas con JUnit para el diseño de las clases.
- Aplicar las anteriores herramientas a ejercicios aprendidas en la asignatura Introducción a la programación.

Nota importante: Siga el esquema de nombrado de paquetes que se indicó en la sesión 01 es decir, para esta sesión cree el paquete: **org.mp.sesion0X**. En ese paquete se crearán todos los programas Java que se proponen en esta sesión dándoles un nombre apropiado en función de lo que realiza el programa en cuestión y que se indica en cada ejercicio entre paréntesis y en negrita (**NombrePrograma**).

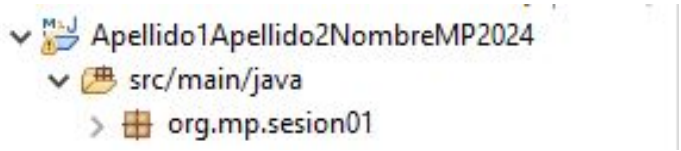


Figura 1. Nombrado de Paquetes: `org.mp.sesion0X`

1. Ejercicio: Corregir la clase `Mayor` ayudándose de un juego de pruebas

El siguiente código Java tiene un pequeño error que impide que funcione correctamente. Tu tarea es identificar y corregir el problema para asegurar que el método `elEnteroMayor` devuelva el entero más grande de un array dado. Examina el código proporcionado. Identifica el error que impide que el método `elEnteroMayor` funcione correctamente. Realiza las correcciones necesarias para asegurar que el método devuelva el entero más grande del array. Asegúrate de mantener la estructura general del código y solo realizar cambios donde sea necesario.

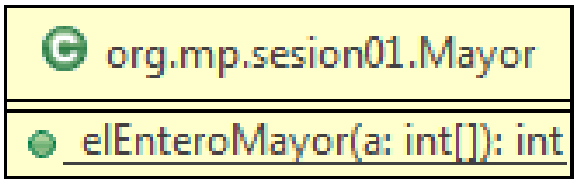


Figura 2. Diagrama de clases: `org.mp.sesion01`

Puedes poner comentarios en el código para explicar tus modificaciones si lo consideras necesario. Verifica que el código corregido sea lógico y cumpla con el propósito del método `elEnteroMayor`.

Se debe ejecutar el juego de pruebas que se proporciona, comprobar su funcionamiento y corregir a través de las pruebas los errores encontrados en el código.

```java
package org.mp.sesion01;

public class Mayor {

    public static int elEnteroMayor(int[] a) {

        int max = Integer.MAX_VALUE;
        for (int indice = 0; indice < a.length - 1; indice++) {
            if (a[indice] > max) {
                max = a[indice];
            }
        }
        return max;
    }
}
```

El `TDD` (Test-Driven Development o Desarrollo Guiado por Pruebas) es una práctica de programación que fue desarrollada por Kent Beck [1] y consiste en primero escribir pruebas unitarias (generalmente con frameworks como JUnit [2]

creado por Kent Beck [1] y Erich Gamma [3]) y luego implementar el código de producción mínimo necesario para pasar esas pruebas.

El ciclo de TDD con JUnit sería:

1. Escribir un test JUnit que falle inicialmente ya que aún no existe código de producción que implemente la funcionalidad deseada.
2. Escribir el código mínimo en producción para pasar el test JUnit.
3. Refactorizar el código escrito para mejorar su calidad y estructura, asegurándose que los tests JUnit sigan pasando después del refactor.
4. Repetir el proceso: escribir otro test JUnit para la siguiente funcionalidad deseada, escribir código para pasarlo, refactorizar, y así sucesivamente.

JUnit [2] creado por Beck [1] y Gamma [3] proporciona las anotaciones y assertivas para escribir y ejecutar tests unitarios fácilmente en Java. El TDD desarrollado por Beck [1] con JUnit produce código confiable, bien probado, facilita refactors y mejora el diseño y mantenimiento.

[1] https://es.wikipedia.org/wiki/Kent_Beck
[2] https://junit.org/junit5/
[3] https://es.wikipedia.org/wiki/Erich_Gamma

2. Ejercicio: Baraja española

Queremos ordenar una baraja española de 40 cartas en el orden natural: primero palo y en el mismo palo en orden ascendente numérico, es decir:

O1, O2, O3, O4, O5, O6, O7, O10, O11, O12, C1, C2,...,C12, E1,E2,...,E12, B1,B2,...B12

O → Oros, C → Copas, E → Espadas, B → Bastos

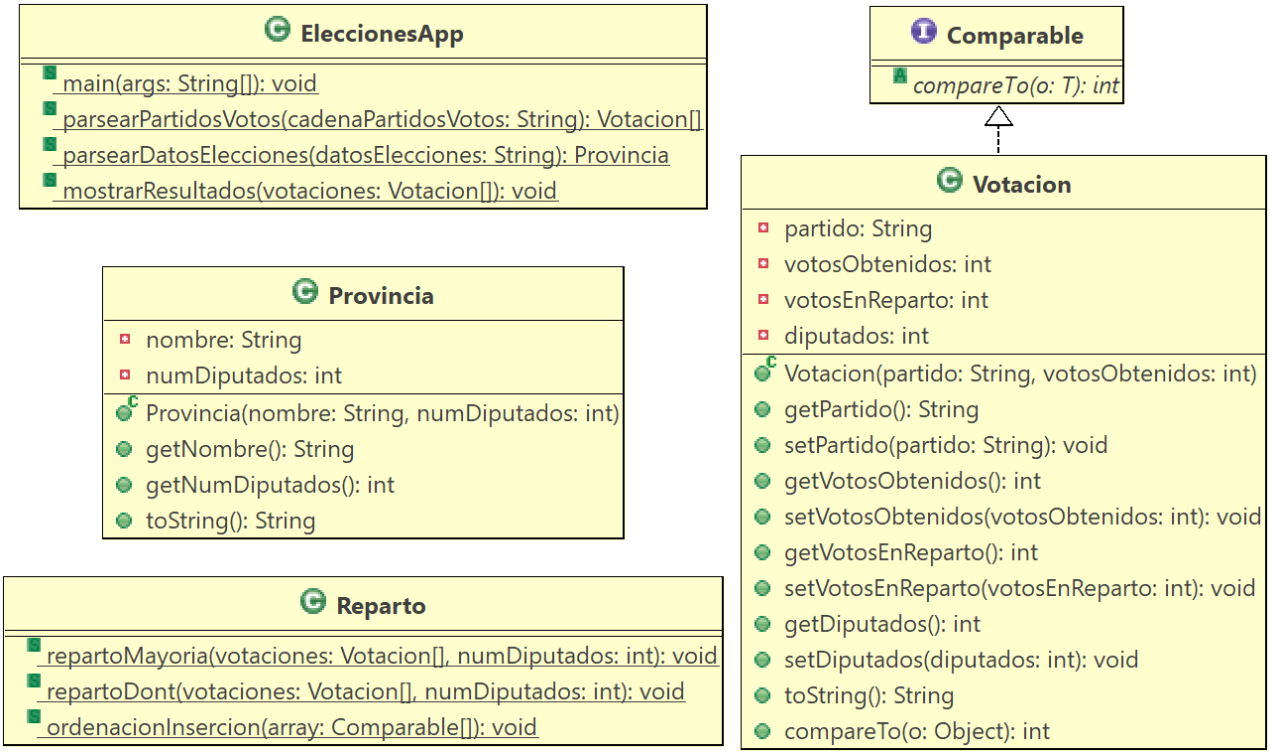
Termina de implementar las clases **Carta** y **Baraja**.Utiliza el método de ordenación por inserción haciendo uso del método **compareTo**.

Unicamente se usarán las propiedades en la clase Carta:

```
private String palo;
private int numero;
```

3. Ejercicio: Reparto proporcional D'Hondt

Desarrolle una aplicación en Java para gestionar el reparto proporcional de diputados después de unas elecciones a nivel provincial. La aplicación constará de las siguientes clases: `EleccionesApp`, `Provincia`, `Votacion`, y `Reparto`.



> Este ejercicio está basado en el examen ordinario de IP2024.
> El objetivo es implementarlo utilizando los juegos de prueba `JUnit` que se proporcionan.
> El programa se situará en el paquete `org.mp.sesion01.dont`.

1. Implemente **toda** la clase `Provincia`, sobreescribiendo los métodos: `toString`, `equals`.

2. Implemente la clase `Votacion`, con únicamente los métodos `toString` y `compareTo` comparando por el atributo `votosEnReparto`.

3. Implemente de la clase `EleccionesApp` los métodos `parsearDatosElecciones` y `parsearPartidosVotos` a partir de las cadenas y construyendo los objetos correspondientes. Utilice el método `split` sobre estas cadenas:
   ```
   String datosProvincia = "Almería,6";
   String partidosVotos = "PP,134340 PSOE,95791 Vox,75573 Sumar,21246";
   ```

4. De la clase `Reparto` implemente el método `public static void repartoMayoria(Votacion[] votaciones, int numDiputados)`. Este método asigna el valor del parámetro `numDiputados` a la `Votacion` cuyo valor del atributo `votosObtenidos` sea el

mayor. Puede ayudarse del método `ordenacionInsercion(Comparable[] array)` (no se pide la implementación de este método).

```
Mayoritario: Votacion [partido = PP, votos = 130652, votosEnReparto = 130652, diputados = 6]
```

5. De la clase `Reparto` implemente el método `public static void repartoDont(Votacion[] votaciones, int numDiputados)`. Este método implementa el reparto D'Hondt, un método de representación proporcional: Un bucle itera tantas veces como `numDiputados` de la provincia.
   - En cada iteración, primero se ordenan las votaciones por `votosEnReparto` (usar `ordenacionInsercion(Comparable[] array)`).
   - Luego se asigna un diputado a la `votacion` con más `votosEnReparto` (siempre estará en la posición del array `votaciones.length - 1`).
   - Y por último, se recalculan los `votosEnReparto` de la `votación` que acaba de recibir un diputado, dividiendo los `votosObtenidos` entre el número de diputados (ya asignados) mas 1: `diputados + 1`.

```
Estado Final: Diputados repartidos 6.
Votacion [partido = Sumar, votos = 21127, votosEnReparto = 21127, diputados = 0]
Votacion [partido = PSOE, votos = 92221, votosEnReparto = 30740, diputados = 2]
Votacion [partido = Vox, votos = 67963, votosEnReparto = 33981, diputados = 1]
Votacion [partido = PP, votos = 130652, votosEnReparto = 32663, diputados = 3]
```

> *keep your bar green, keep your code clean.*
>
> — *Anónimo*

> *El verdadero progreso ocurre cuando los errores son admitidos y corregidos, no cuando se niegan o se ignoran.*
>
> — *Anónimo*