Investigación de conceptos

Ejercicio 9. Tic-Tac-Toe en Java

1) Clase `Character`

En la lógica de validación normalizo la ficha ingresada a mayúscula con `Character.toUpperCase(char)`, de modo que `'x'` y `'X'` se traten por igual. Además, `Character` provee utilidades para clasificar y transformar caracteres:

- Transformaciones: `toUpperCase(char)`, `toLowerCase(char)`, `toTitleCase(char)`. (Oracle Docs)
- Predicados de tipo: `isLetter`, `isDigit`, `isWhitespace`, `isUpperCase`, `isLowerCase` (útiles si decido validar entradas por consola). (Oracle Docs)

Ejemplo mínimo (usado en mi método `colocarFicha`):

```
char f = Character.toUpperCase(ficha);
if (f != 'X' && f != 'O') throw new IllegalArgumentException("La ficha debe s
er 'X' u 'O'.");
```

> Base conceptual general sobre Character: En el libro explica la diferencia entre tipos primitivos (char) y sus clases envoltorio, y el uso de métodos utilitarios para procesar caracteres (Java: Cómo Programar, 10.ª ed., secciones de tipos primitivos, clases envoltorio y cadenas). Para la firma exacta de métodos, consulto la API de Oracle. (Oracle Docs)

---

2) Arreglos (unidimensionales y bidimensionales)

En el tablero uso una **matriz** 3×3 de `char`:

```
private final int N = 3;
private final char[][] matriz = new char[N][N];
```

La **inicializo** con `' '` (espacio) como “vacío” y accedo por índices `[fila][columna]`.

2.1 Declaración y creación (ejemplos)

- **Unidimensional**

  `int[] a = new int[5];` — valores por defecto en `0`.

  `String[] nombres = {"Ana", "Luis"};` — literal.

- **Bidimensional (arreglo de arreglos)**

  `char[][] m = new char[3][3];` — cada fila es un arreglo.

En Java, los "arreglos multidimensionales" son realmente **arreglos de arreglos**, y cada fila puede tener longitudes distintas (ragged arrays), aunque aquí uso 3×3 fijo. Esta característica está documentada en los tutoriales oficiales de Oracle. (Oracle Docs)

Para operaciones comunes de inicialización y depuración:

- `Arrays.fill(matriz[i], ' ');` para rellenar rápido una fila.
- `Arrays.deepToString(matriz)` para imprimir matrices.

  Estas utilidades están en `java.util.Arrays`. (Oracle Docs)

---

3) `final` : propósito y uso en el código

Uso `final` para expresar **invariantes** del objeto:

- `private final int N = 3;` → el tamaño del tablero **no cambia**.
- `private final char[][] matriz = new char[N][N];` → **no reasigno** la referencia de la matriz (aunque sí puedo modificar su contenido: `matriz[i][j] = 'X'`).

Puntos clave (según Oracle):

- En **primitivos**, `final` impide **reasignar** el valor.
- En **referencias**, `final` impide **cambiar la referencia**; el **estado interno** del objeto/arreglo sí puede mutar.
- También puede aplicarse a **métodos** y **clases** para impedir sobreescritura o herencia, respectivamente. (Oracle Docs)

Básicamente, `final` se asegura de la inmutabilidad de valores, en este caso lo uso como constante.

---

4) Excepciones empleadas y criterio de uso

4.1 `IllegalArgumentException`

Si la ficha no es `'X'` u `'O'`. Es la excepción adecuada cuando un **parámetro** del método es inválido. (Oracle Docs).

Se refiere a que un argumento de la expresión no es aceptada por el sistema.

4.2 `IndexOutOfBoundsException`

La uso en `validarIndices(fila, columna)` si los índices están fuera de `0..2`. Representa correctamente "índice fuera de rango" en arreglos/colecciones. (Oracle Docs)

Al leer con `Scanner`, si el usuario ingresa texto cuando espero un entero, se produce esta excepción. La evito validando con `hasNextInt()`; de necesitarlo, podría capturarla. (Oracle Docs)

---

5) Verificación de ganador y consistencia del estado

El algoritmo comprueba **filas**, **columnas** y **diagonales** buscando tres iguales y distintos de `' '`; retorna `'X'` u `'O'`, o `'\0'` si no hay ganador. Además, el tablero imprime su estado actual y reporta empate con `estaLleno()`. (Fragmentos relevantes en la clase `Tablero`.)

---

6) Literal de caracter. '/0'

En **Java**, los literales de carácter aceptan **secuencias de escape**. Además de `\n`, `\t`, `\'`, `\"`, `\\`, etc., el lenguaje permite **escapes octales**: una barra invertida seguida de 1–3 dígitos octales (`0–7`). Ese formato viene heredado por compatibilidad con C. En particular, `\0` es el escape octal del **código Unicode 0** (carácter **NUL**, `\u0000`). La especificación del lenguaje lo establece explícitamente en la gramática de *EscapeSequence* y *OctalEscape*.

Es **un carácter válido** en Java (no es `null`). Equivale a `'\u0000'`.

Ejercicio 10. Cadenas

1) `Normalizer` y por qué lo uso

Qué es.

```
java.text.Normalizer
```

transforma texto Unicode a formas equivalentes “compuestas” o “descompuestas”, lo que facilita búsquedas y comparaciones. La API oficial indica que **normalize** implementa los formularios estándar del Anexo #15 de Unicode (NFC, NFD, etc.). (Oracle, s. f.).

Oracle Docs

Para validar palíndromos y contar vocales/consonantes de forma robusta, primero **normalizo** el texto a **NFD** y elimino diacríticos. Normalizar es transformar el texto Unicode a una forma consistente para que caracteres “equivalentes” se comparen correctamente (por ejemplo, `á` ≡ `a` + acento). La API de Java explica que `Normalizer` implementa estos formularios estándar, y el tutorial oficial ejemplifica su uso práctico en búsquedas y comparaciones. (Oracle Docs)

En mi utilidad `normaliza`, hago:

```
String s = Normalizer.normalize(s, Normalizer.Form.NFD)
                .replaceAll("\\p{M}+","")
                .toLowerCase();
```

- `Form.NFD` descompone letra+marca (ej.: `á` → `a` + ◌́).
- `\\p{M}+` borra **marcas combinantes** (clase Unicode *Mark*). La documentación de `Pattern` indica que `\p{...}` accede a propiedades Unicode; la categoría **M** corresponde a marcas combinantes según el estándar Unicode. (Oracle Docs, Unicode)

---

2) Expresiones regulares que empleo: `[^a-z0-9]` y `\\p{M}+`

- `[^a-z0-9]`: es una **clase negada**; coincide con *todo lo que no sea* una letra `a–z` o dígito `0–9`. Así filtro espacios, signos y otros símbolos antes de comparar la cadena con su reverso. (Las clases y propiedades Unicode están definidas en `Pattern`.) (Oracle Docs)

- `\\p{M}+`: captura una o más **marcas combinantes** (categoría general Unicode **M: Mark**), por eso sirve para "desacentuar" tras NFD. El estándar Unicode (UAX #44) define estas categorías y su semántica. (Unicode)

Nota: uso `[^a-z0-9]` **después** de normalizar y convertir a minúsculas.

---

3) `StringBuilder` y la inversión eficiente

`StringBuilder` provee una cadena **mutable** pensada para modificaciones eficientes. Su método `reverse()` invierte la secuencia cuidando parejas sustitutas (surrogate pairs), de modo que la inversión respeta los límites de puntos de código. (Oracle Docs)

Por eso, tanto en `invertir()` como en la verificación de palíndromo uso:

```
String cadenaReverse = new StringBuilder(cadena).reverse().toString();
```

---

4) `for (char v : VOCALES)` — *enhanced for*

Para chequear pertenencia (¿el carácter `c` es una vocal?), itero el arreglo `VOCALES` con el *enhanced for*:

```
for (char v : VOCALES) if (c == v) return true;
```

El *enhanced for* está diseñado precisamente para recorrer **arreglos y colecciones** y es la forma recomendada cuando no necesito el índice. (Oracle Docs, jcp.org)

Cómo encajan todas las piezas en mi archivo

1. **Normalizo** (NFD) → **elimino marcas** (`\\p{M}+`) → **minúsculas**.
2. **Filtro** con `[^a-z0-9]` para quedarme solo con alfanuméricos antes de la comparación de palíndromo. (Oracle Docs)
3. **Invierto** con `StringBuilder.reverse()` y comparo `s.equals(rev)`. (Oracle Docs)
4. Para contar vocales/consonantes, evalúo `Character.isLetter(c)` y hago pertenencia con `for (char v : VOCALES)`. (Oracle Docs)

Referencias

- Deitel, P. J., & Deitel, H. M. (2016). *Java: Cómo programar* (10.ª ed.). Pearson.
- Oracle. (s. f.). *Normalizing Text (The Java™ Tutorials)*. https://docs.oracle.com/javase/tutorial/i18n/text/normalizerapi.html (Oracle Docs)
- Oracle. (s. f.). *Pattern (Java Platform SE)* — Unicode categories y `\p{...}`. https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html (Oracle Docs)
- Unicode Consortium. (s. f.). *UAX #44: Unicode Character Database* (General Category, Mark). https://www.unicode.org/reports/tr44/ (Unicode)
- Oracle. (s. f.). *StringBuilder (Java Platform SE)* — `reverse()`. https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html (Oracle Docs)
- Oracle. (s. f.). *The StringBuilder Class (The Java™ Tutorials)*. https://docs.oracle.com/javase/tutorial/java/data/buffers.html (Oracle Docs)
- Oracle. (s. f.). *The for Statement (The Java™ Tutorials) — enhanced for*. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html (Oracle Docs)
- Java Community Process (JCP). (s. f.). *An enhanced for loop for the Java™ Programming Language* (JSR 201 note). https://jcp.org/aboutJava/communityprocess/jsr/tiger/enhanced-for.html (jcp.org)
- **Oracle (Documentación oficial):**
  - `Character` (transformaciones y predicados): Java SE 8 API. (Oracle Docs)
  - Arreglos (concepto de “array de arrays”): *Learning the Java Language – Arrays*. (Oracle Docs)
  - Utilidades para arreglos (`java.util.Arrays`): Java SE API. (Oracle Docs)
  - `final` en métodos/clases y variables `final`: *The Java™ Tutorials*. (Oracle Docs)
  - Excepciones: `IllegalArgumentException`, `IndexOutOfBoundsException`, `UnsupportedOperationException`, `InputMismatchException`. (Oracle Docs)