23 | Aplicación de estructuras: Polinomios

Meta

Que el alumno seleccione y utilice sus propias estructuras de datos para resolver un problema de cómputo.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Valorizar el uso de la estructura de datos correcta para implementar un algoritmo.
- Utilizar una interfaz de texto para interactuar dinámicamente con el usuario.
- Visualizar cómo se utiliza un intérprete de comandos para ejecutar las instrucciones dadas por el usuario.
- Pasar de notación infija a alguna de las otras dos.
- Evaluar operaciones en notación infija con precedencia, prefija y postfija.

Antecedentes

El objetivo de este proyecto es implementar y probar una clase que permita representar y operar con polinomios de **una sola variable**. La definición del tipo de dato abstracto *polinomio* se corresponde con la definición algebraica.

$$P(x) = \sum_{i=0}^{n} a_i x^i \tag{23.1}$$

Como parte del ejercicio, esta vez deberás implementar tus propias pruebas unitarias, para ir verificando que tu código funciona como lo esperas. Se incluye la estructura

para utilizar `JUnit 5`. Para programar tus pruebas se te recomienda revisar la documentación en https://junit.org/junit5/docs/current/user-guide/#writing-tests, basta con el ejemplo más sencillo para los fines de tu proyecto, aunque eres libre de explorar tanto como desees, sólo estima bien tus tiempos. En los pasos siguientes deberás ir añadiendo pruebas y probando conforme progresas. No intentes programar todo sin haber probado lo que vas construyendo.

Ejercicios

1. Redacta la definición del tipo de dato abstracto (TDA) polinomio e inclúyela en tu reporte, recuerda que parte de esta defindión incluye la especificación de las operaciones que es posible realizar con polinomios. Dado que esta definición depende de la definición de monomio incluye ésta también como TDA.

2. Completa la estructura de directorios de tu proyecto. Verifica/crea los directorios `src`, `test` y `libs`. Crea el paquete `ed.aplicaciones.algebra` tanto en `src` como en `test`. Se incluye un archivo `build.xml` que puede funcionar con `JUnit 5`, úsalo como base y, si es necesario, modifícalo para que se adapte a las necesidades de tu proyecto.

3. Dentro programa una clase `Monomio` y una clase `Polinomio`. Al determinar los atributos que debe tener, considera elegir la estructura de datos apropiada para representar polinomios de cualquier longitud (cualquier número de monomios). Para que errores en tus prácticas no afecten tu proyecto, puedes utilizar las estructuras vistas en su versión de la API de Java o las tuyas, si prefieres.

   Debes decidir si guardarás los monomios en un orden predeterminado; toma en cuenta que más adelante tendrás que operar con ellos y será importante poder ordenar sus términos de acuerdo a los exponentes. Incluye en tu reporte la especificación del criterio que utilizarás para ordenar los monomios.

4. Define el o los constructores que vas a necesitar. Agrega métodos `toString()` que permitan visualizar el contenido del polinomio regresando una cadena adecuada. Crea pruebas unitarias que te permitan verificar la creación de tus polinomios.

   - Crea un monomio, utiliza el método `toString()` y verifica que la cadena devuelta sea igual a la que esperabas, de paso imprímelo en pantalla.
   - Crea varios polinomios: compuestos por uno, dos o varios monomios. Utiliza `toString()` para verificar que quedaron escritos como esperabas. Si gustas, puedes imprimirlos en pantalla.

5. Programa el método `simplifica()`, que agrupe términos semejantes y devuelva un nuevo polinomio con el resultado. El nuevo polinomio debe estar hecho de monomios nuevos, no modifiques los del polinomio original. TIP: averigua qué es un

constructor copia. Observa que, para simplificar tu polinomio, te convendrá agregar operaciones a la clase monomio tales como `más(Monomio m)`. Ve modificando a `Monomio` conforme lo vayas requiriendo. Agrega las pruebas unitarias para verificar que el método funciona como esperabas.

6. Programa el método `más(Polinomio p)`, de tal modo que sume este polinomio con `p` y devuelva un nuevo polinomio con el resultado. Observa que el nuevo polinomio debe estar hecho de monomios nuevos y el polinomio original no debe ser afectado.

7. Programa el método `por(Polinomio p)`, de tal modo que multiplique este polinomio por `p` y devuelva un nuevo polinomio con el resultado. Observa que el nuevo polinomio debe estar hecho de monomios nuevos.

$$P_1(x) \times P_2(x) = \left(\sum_{i=0}^{n} a_i x^i\right) \times \left(\sum_{j=0}^{m} b_j x^j\right) = \sum_{i=0}^{n} a_i x^i \times \left(\sum_{j=0}^{m} b_j x^j\right) \tag{23.2}$$

   TIP: considera a la identidad.

8. No olvides documentar tu código.

9. Agrega un método `main` donde hagas una pequeña demostración (unos cinco ejemplos) de cómo funciona tu clase `Polinomio`.