Laboratorios A.E.D.: Laboratorio 2

Guillermo Román
guillermo.roman@upm.es

Julio García
juliomanuel.garcia@upm.es

Lars-Åke Fredlund
larsake.fredlund@upm.es

Tonghong Li
tonghong.li@upm.es

Manuel Carro
manuel.carro@upm.es

Marina Álvarez
marina.alvarez@upm.es

Normas

- Fechas de entrega y penalización:

Hasta el Martes 4 de Octubre, 12:00 horas (**mediodía**)	0 %
Hasta el Miércoles 5 de Octubre, 12:00 horas (**mediodía**)	20 %
Hasta el Jueves 6 de Octubre, 12:00 horas (**mediodía**)	40 %
Hasta el Viernes 7 de Octubre, 12:00 horas (**mediodía**)	60 %

  Después la puntuación máxima será 0
- Se comprobará plagio y se actuará sobre los detectados.
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender.

Entrega

- Todos los ejercicios de laboratorio se deben entregar a través de

`http://deliverit.fi.upm.es`

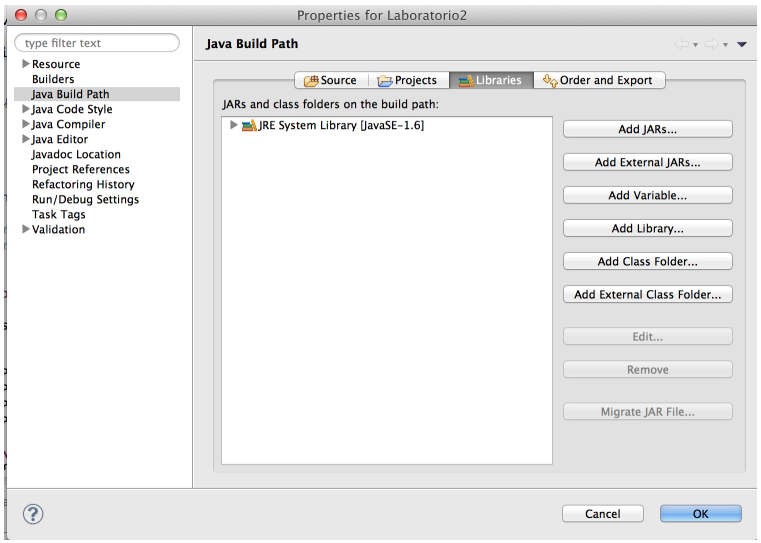
- **Nota: La dirección ha cambiado con respecto con al año pasado.**
- El fichero que hay que subir es `Polinomio.java`.

Configuración previa

- Arrancad Eclipse
- Si trabajáis en portátil, podéis utilizar cualquier versión reciente de Eclipse. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*.
- Cambiad a "Java Perspective".
- Debéis tener instalado al menos Java JDK 8.
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios.
  - **No debéis elegir la opción de crear el fichero** `module-info.java`
- Cread un *package* `aed.polinomios` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios → Laboratorio 2 → `Laboratorio2.zip`; descomprimidlo
- Contenido de Laboratorio2.zip:
  - `Polinomio.java, Monomio.java, TesterLab2.java`

Configuración previa

- Importad al paquete `aed.polinomios` los fuentes que habéis descargado
  (`Polinomio.java, Monomio.java, TesterLab2.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios).

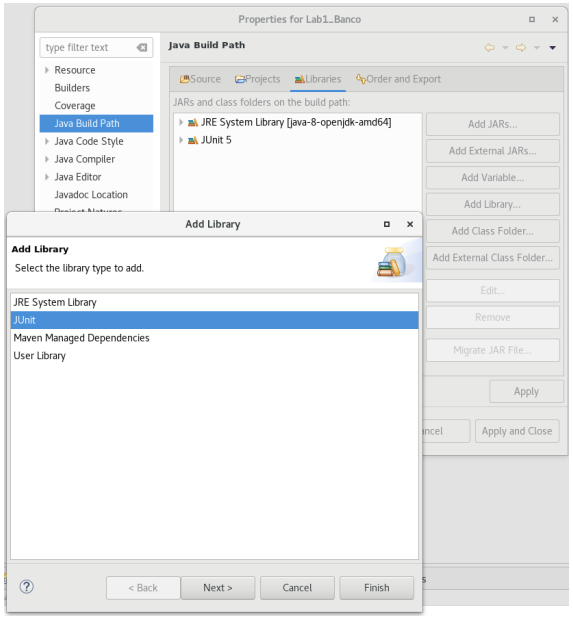


Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda
- Usad la opción “Add External JARs...”.
- Si vuestra instalación distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

Configuración previa

- Añadid al proyecto `aed` la librería `JUnit 5`



Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda;
- Usad la opción "Add Library..." → Seleccionad "Junit" → Seleccionad "JUnit 5"
- Si vuestra instalacion distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

- En la clase `TesterLab2` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab2`, pulsando el botón derecho sobre el editor, seleccionar "Run as..." → "JUnit Test"
- NOTA: Si al ejecutar, no aparece la vista "JUnit", podéis incluirla en "Window" → "Show View" → "Java" → "JUnit"

Documentación de la librería aedlib.jar

- La documentación de la API de `aedlib.jar` esta disponible en `http://costa.ls.fi.upm.es/teaching/aed/docs/aedlib/`
- También se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*):
  - En el "Package Explorer": "Referenced Libraries" → aedlib.jar y elige la opción "Properties". Se abre una ventana donde se puede elegir "Javadoc Location" y ahí se pone como "javadoc location path:"

`http://costa.ls.fi.upm.es/teaching/aed/docs/aedlib/`

y presionar el buton "Apply and Close"

Tarea: implementar la clase `Polinomio`

Se quiere desarrollar un librería para poder manipular y evaluar polinomios sobre una variable y con coeficientes y exponentes enteros.

Recordatorio:

- Un *monomio* es una expresion algebraica, por ejemplo $5x^2$, compuesto por un coeficiente (5), una variable o parte literal (x), y un exponente o grado (2).
- Un *polinomio* es una suma de monomios, por ejemplo $5x^2 + 3x + 81$.
- La librería trabajará con polinomios de una variable (x) y con coeficientes y exponentes enteros y ≥ 0.
- Dado que solo usamos una variable, no necesitamos representarla y solo es preciso guardar el coeficiente y exponente de cada monomio.

Representación de un polinomio

- Dentro la clase representamos un polinomio como una lista de monomios (de tipo `PositionList`) guardada en el atributo `monomios` dentro `Polinomio`:

```
PositionList<Monomio> terms;
```

- Condiciones a cumplir:
  - Los monomios deben estar ordenados en orden decreciente de su grado. Por ejemplo, el polinomio $x^5 + -2x^3 + 8$ se representa como

    `[Monomio(1,5), Monomio(-2,3), Monomio(8,0)]`
  - No puede haber varios monomios del mismo grado en la lista. P.e., la representación `[Monomio(1,5), Monomio(2,5)]` no es válida.
  - No puede haber ningún monomio con coeficiente 0 en la lista. El polinomio 0 (o polinomio nulo) se representa como la lista vacía.

Representación de un monomio

- Un monomio se representa por un par de un coeficiente y un exponente:

```
public class Monomio extends Pair<Integer,Integer> {

  public Monomio(int coeficiente, int exponente) {
    super(coeficiente, exponente);
  }

  public int getCoeficiente() { return getLeft(); }
  public int getExponente() { return getRight(); }
}
```

Operaciones a implementar

- Se pide implementar los siguientes métodos dentro la clase `Polinomio`:

```
// suma dos polinomios, subtraccion y multiplicacion
static Polinomio suma(Polinomio p1, Polinomio p2);
static Polinomio resta(Polinomio p1, Polinomio p2);
static Polinomio multiplica(Monomio m, Polinomio p);
static Polinomio multiplica(Polinomio p1, Polinomio p2);

// calcula el valor de un polinomio cuando x = valor
long evaluar(int valor);

// devuelve el grado del polinomio
int grado();

// comprueba si dos polinomios son iguales
boolean equals(Object obj);
```

- Documentación detallada en

https://costa.ls.fi.upm.es/teaching/aed/docs/practicas/polinomios/

Tarea opcional: implementar un *parser* para polinomios

- Podéis obtener 2 puntos adicionales (para un máximo de 12 puntos) implementando correctamente el constructor

  ```
  public Polinomio(String polinomio) { ... }
  ```

  que inicializa el atributo `monomios` dentro `Polinomio` con el polinomio representado por el `String polinomio` .
- Textualmente un polinomio esta representado por un suma de monomios separados por '+' .
- Cada monomio puede tener una de las formas:

  `Kx^N` `x^N` `Kx` `x` `K`

  con `N` y `K` enteros, $\texttt{N} > 1$ y $\texttt{K} \neq 0$ **excepto** si es el único monomio en el polinomio, en cuyo caso puede ser $\texttt{K} = 0$.
- Pueden aparecer espacios ' ' entre las diferentes partes.
- Suponed que los monomios en el `String` aparecen en orden descendente de exponentes y que nunca hay dos monomios con exponentes iguales.

Ejemplos del uso del constructor opcional

```
new Polinomio("2x^3 + 9")    // == 2x^3 + 9x^0
                             // ==> [Monomio(2,3), Monomio(9,0)]
new Polinomio("2x^3 + -9")   // == 2x^3 + -9x^0
                             // ==> [Monomio(2,3), Monomio(-9,0)]
new Polinomio("x")           // == 1x^1 ==> [Monomio(1,1)]
new Polinomio("5")           // == 5x^0 ==> [Monomio(5,0)]
new Polinomio("8x")          // == 8x^1 ==> [Monomio(8,1)]
new Polinomio("0")           // representado con lista vacia
                             // ==> []
```

Ejemplos

```
Polinomio p1 = new Polinomio("2x^2 + 9x + 3");
Polinomio p2 = new Polinomio("8x^2 + -7");
Polinomio.suma(p1,p2);    // => 10x^2 + 9x^1 + -4
Polinomio.resta(p2,p1);   // => 6x^2 + -9x^1 + -10

Polinomio.multiplica(p1,new Polinomio("5")); // => 10x^2 + 45x^1 + 15
Polinomio.multiplica(p1,p2); // => 16x^4 + 72x^3 + 10x^2 + -63x^1 + -21

p1.grado();               // => 2
p2.evaluar(5);            // => 98 (sustituyendo x por 5 en p2)

p1.equals(p2);                                    // false
new Polinomio("5x").equals(new Polinomio("5x"));  // => true
```

Consejos

- Hay mucho material online sobre las operaciones basicas en polinomios, consulta por ejemplo `https://es.wikipedia.org/wiki/Polinomio` y `https://es.wikipedia.org/wiki/Monomio`.
- No sólo está permitido, sino recomendado, definir y usar métodos auxiliares para reducir la cantidad de código. Por ejemplo, las operaciones de sumar y restar son muy parecidas y es posible implementar un método auxiliar usado por `suma` y `resta`.

Consejos sobre la implementación del constructor opcional

Consulta la documentación estándar de Java para saber más sobre las operaciones sobre `String`. Métodos útiles (`s` es un `String`):

- `s.split(regexp)` devuelve un *array* de `String` donde cada elemento ha sido separado por `regexp`. Por ejemplo, si `s = "2x^3 + 1x + 2"` entonces `s.split("\\+")` devuelve el *array* `["2x^3 ", " 1x ", " 2"]`. Otros ejemplos:

```
"xA".split("x")  => ["", "A"]
"Ax".split("x")  => ["A"]
"x".split("x")   => []
"abc".split("x") => ["abc"]
```

- Un "regexp" es una expresion regular. Para esta tarea solo hace falta reconocer caracteres individuales. Por ejemplo la expresion regular `"x"` reconoce el caracter `'x'`. Sin embargo, algunas caracteres como '+' y '^' tiene signicado especial. Para reconocer estos caracteres tenemos que declarar que no nos interesa el signficado especial. Usaremos `"\\+"` y `"\\^"` para eso.

Consejos sobre la implementación del constructor opcional

- `s.contains(s1)` comprueba si `s1` aparece dentro `s`. Por ejemplo: `new String("hello").contains("lo")` devuelve `true`.
- Para convertir un string representando un número a un entero se puede llamar `Integer.parseInt(s)`. Por ejemplo `Integer.parseInt("-24")` devuelve el entero -24.
- `s.trim()` devuelve un nuevo `String` donde los espacios al principio y final de `s` están eliminados. Por ejemplo, `new String(" hola ").trim()` devuelve `"hola"`.

Evaluación

Seguir estos consejos os permitirá conseguir mejores resultados!

- Corrección
- Ausencia de código repetido con la misma funcionalidad (podéis usar métodos auxiliares para evitarlo)
- Concisión del código
- Legibilidad, incluida selección de nombres descriptivos para variables y métodos
- El código debe estar correctamente indentado y con comentarios *útiles* cuando lo veáis necesario
- Eficiencia:
  - Se valorará la complejidad computacional del código
  - Se valorará no iterar innecesariamente en los recorridos de las estructuras de datos

Notas

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar
- Debe pasar todos los test `TesterLab2` correctamente sin mensajes de error
- **Nota:** una ejecución sin mensajes de error y que pase todas las pruebas **no** significa que el método sea correcto (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se corrigen manualmente antes de dar la nota final