

Polinomio — TDA de polinomios de una variable

Polinomios con coeficientes enteros: $p(x) = a_0 + a_1x + \dots + a_nx^n$, $n \in \mathbb{N}$, $a_i \in \mathbb{Z}$.

Validación: 0 errores, 0 warnings. Esta versión `IMPORTS Integer`, lo que mueve la exponenciación entera potencia fuera de `Polinomio` — resolviendo el único warning V1 que tenía la versión anterior (V1 le pedía a potencia axiomas sobre los constructores de `Polinomio`, lo cual no tenía sentido).

Cambio respecto a la versión anterior — `IMPORTS Integer`:

- `potencia(x, n) = x^n` es exponenciación entera: una operación que *pertenece al tipo Integer*, no a `Polinomio`.
- Antes estaba declarada y axiomatizada localmente como `auxiliar` — V1 emitía un warning porque era una "operación de `Polinomio`" sin axiomas sobre `cero/anadir`.
- Ahora `Polinomio` declara `IMPORTS Integer` y **usa** potencia como operación importada — sin declararla ni re-axiomatizarla. Separación correcta de responsabilidades: `Polinomio` conoce solo sus propias operaciones; la aritmética entera vive en `Integer`.

Diseño algebraico: dos constructores básicos y tres operaciones derivadas.

- **cero** — el polinomio nulo $p(x) = 0$.
- **anadir(p, coef, grado)** — añade el monomio $\text{coef} \cdot x^{\text{grado}}$ a p . Precondición $\text{grado} \geq 0$ (los grados son naturales; además garantiza que potencia se invoque siempre con exponente ≥ 0).
- **evaluar, coeficiente, suma** — operaciones derivadas.

Todo polinomio se representa como una cadena `anadir(anadir(...anadir(cero(), c1, g1)..., cn, gn))`.

La representación NO es canónica — y es deliberado. `anadir` admite agregar varios monomios del mismo grado; coeficiente los **acumula** (suma los coeficientes de igual grado). Por eso `anadir(anadir(cero(), 2, 1), 3, 1)` y `anadir(cero(), 5, 1)` son **observacionalmente iguales**: ambos dan coeficiente 5 en grado 1. La igualdad de polinomios es observacional (vía coeficiente / evaluar), no sintáctica.

1. Especificación

```
CLASS Polinomio
IMPORTS Integer
BASIC CONSTRUCTORS cero, anadir
EFFECTIVE
TYPE Polinomio
```

OPERATIONS

```
-- — Constructores básicos —————
```

```
-- cero: el polinomio nulo  $p(x) = 0$ .
cero: -> Polinomio;
```

```
-- anadir(p, coef, grado): añade el monomio coef·x^grado a p.
-- Precondición: el grado debe ser un natural ( $\geq 0$ ).
-- (Esto garantiza que `potencia` se invoque solo con  $n \geq 0$ .)
anadir: Polinomio(p) * Integer(coef) * Integer(grado) -> Polinomio
  pre: grado >= 0;
```

```
-- — Operaciones derivadas —————
```

```
-- evaluar(p, x): valor numérico de p evaluado en x.
```

```
evaluar: Polinomio * Integer -> Integer;
```

```
-- coeficiente(p, g): coeficiente del monomio de grado g.
```

```
coeficiente: Polinomio * Integer -> Integer;
```

```
-- suma(p, q): polinomio suma de p y q.
```

```
suma: Polinomio * Polinomio -> Polinomio;
```

```
-- Nota: `potencia: Integer * Integer -> Integer` es una
```

```
-- operación del tipo Integer (importada vía IMPORTS Integer),
```

```
-- así que NO se declara aquí.
```

```
AXIOMS p, q: Polinomio; x, c, g, g1: Integer;
```

```
-- —————
-- evaluar(p, x): valor del polinomio en x.
-- evaluar(cero, x) = 0
-- evaluar(anadir(p,c,g), x) = c·x^g + evaluar(p, x)
-- Cada monomio aporta c·x^g (usando `potencia` de Integer);
-- el resto se evalúa recursivamente.
-- —————
```

```
evaluar(cero(), x) = 0;
```

```
evaluar(anadir(p, c, g), x) =
  (c * potencia(x, g)) + evaluar(p, x);
```

```
-- —————
-- coeficiente(p, g): coeficiente del monomio de grado g.
-- coeficiente(cero, g) = 0
-- coeficiente(anadir(p,c,g1), g) =
--   si g1 = g → c + coeficiente(p, g) (acumula)
--   si g1 ≠ g → coeficiente(p, g)
-- La acumulación garantiza que anadir varios monomios del
-- mismo grado equivale a anadir uno con la suma de coefs.
-- —————
```

```
coeficiente(cero(), g) = 0;
```

```
(g1 = g) implies
  coeficiente(anadir(p, c, g1), g) = c + coeficiente(p, g);
```

```
not (g1 = g) implies
  coeficiente(anadir(p, c, g1), g) = coeficiente(p, g);
```

```
-- —————
-- suma(p, q): suma de dos polinomios.
-- Inducción sobre el segundo argumento – cada monomio de q
-- se añade, uno por uno, sobre p.
-- suma(p, cero) = p
-- suma(p, anadir(q, c, g)) = anadir(suma(p, q), c, g)
```

```
--
-- Correctitud: coeficiente(suma(p,q), g)
--               = coeficiente(p, g) + coeficiente(q, g)  ∀ g.
-- _____
suma(p, cero()) = p;

suma(p, anadir(q, c, g)) = anadir(suma(p, q), c, g);
```

END-CLASS

2. Operaciones y axiomas

Operación	Origen / tipo	Axiomas
cero	constructor básico de Polinomio	— (valor primitivo)
anadir	constructor básico <i>pre: grado ≥ 0</i>	— (constructor)
evaluar	derivada	2 — base sobre cero (= 0) + recursiva sobre anadir ($c \cdot x^g + \text{evaluar}(p, x)$)
coeficiente	derivada	3 — base sobre cero (= 0) + 2 implies sobre anadir según $g_1 = g$
suma	derivada	2 — inducción sobre el 2.º argumento: base $\text{suma}(p, \text{cero}) = p$ + recursiva
potencia	importada — operación del tipo Integer (IMPORTS Integer)	— (definida y axiomatizada en Integer, no aquí)

3. Ejemplo: el polinomio $p(x) = 3 + 2x^2$

Representación como término:

```
p = anadir(anadir(cero(), 3, 0), 2, 2)
    └─ monomio 3·x0    └─ monomio 2·x2
```

Traza de evaluar(p, 5)

Paso	Reescritura
1	evaluar(anadir(anadir(cero(),3,0), 2,2), 5) → $2 * \text{potencia}(5, 2) + \text{evaluar}(\text{anadir}(\text{cero}(),3,0), 5)$
2	$\text{potencia}(5, 2) = 25$ (operación de Integer, importada)
3	evaluar(anadir(cero(),3,0), 5) → $3 * \text{potencia}(5, 0) + \text{evaluar}(\text{cero}(), 5) = 3 * 1 + 0 = 3$
4	Total: $2 * 25 + 3 = 50 + 3 = 53$

Verificación: $p(5) = 3 + 2 \cdot 5^2 = 3 + 50 = 53 \checkmark$

Traza de `coeficiente(p, 2)`

Paso	Reescritura
1	<code>coeficiente(anadir(anadir(cero(),3,0), 2,2), 2)</code> <code>g1 = 2, g = 2 → g1 = g ✓ → 2 + coeficiente(anadir(cero(),3,0), 2)</code>
2	<code>coeficiente(anadir(cero(),3,0), 2)</code> <code>g1 = 0, g = 2 → g1 ≠ g → coeficiente(cero(), 2)</code>
3	<code>coeficiente(cero(), 2) = 0</code>
4	Total: $2 + 0 = 2 \checkmark$ (coeficiente de x^2)

Traza de `suma(p, q)` con $q = 7x$

Sea $q = \text{anadir}(\text{cero}(), 7, 1)$ (el monomio $7 \cdot x^1$):

```
suma(p, anadir(cero(), 7, 1))
= anadir(suma(p, cero()), 7, 1)      -- axioma recursivo de suma
= anadir(p, 7, 1)                    -- axioma base: suma(p, cero) = p
= anadir(anadir(anadir(cero(),3,0), 2,2), 7, 1)
```

El resultado representa $3 + 7x + 2x^2$. Verificación vía coeficiente:

Grado	<code>coeficiente(suma(p,q), g)</code>	<code>coef(p,g) + coef(q,g)</code>
0	3	$3 + 0 = 3 \checkmark$
1	7	$0 + 7 = 7 \checkmark$
2	2	$2 + 0 = 2 \checkmark$

4. Propiedades garantizadas

Correctitud de suma: para todo grado g se cumple $\text{coeficiente}(\text{suma}(p, q), g) = \text{coeficiente}(p, g) + \text{coeficiente}(q, g)$. Demostrable por inducción sobre q :

- Base** ($q = \text{cero}$): $\text{coeficiente}(\text{suma}(p, \text{cero}), g) = \text{coeficiente}(p, g) = \text{coeficiente}(p, g) + 0 = \text{coeficiente}(p, g) + \text{coeficiente}(\text{cero}, g)$.
- Paso** ($q = \text{anadir}(q', c, g')$): $\text{coeficiente}(\text{anadir}(\text{suma}(p, q'), c, g'), g)$ distribuye en los dos casos de `coeficiente`; por hipótesis inductiva $\text{coeficiente}(\text{suma}(p, q'), g) = \text{coeficiente}(p, g) + \text{coeficiente}(q', g)$, y el monomio c se suma exactamente cuando $g' = g$ — igual que en $\text{coeficiente}(\text{anadir}(q', c, g'), g)$.

Consistencia evaluar ↔ coeficiente: `evaluar(p, x)` coincide con la suma $\sum \text{coeficiente}(p, g) \cdot x^g$ sobre todos los grados g presentes — porque ambos recorren la misma estructura de monomios.

Esto valida que las dos "vistas observacionales" del polinomio son coherentes.

Acumulación en coeficiente: el axioma del caso $g1 = g$ usa $c + \text{coeficiente}(p, g)$ (no simplemente c). Esto hace que agregar dos monomios del mismo grado — `anadir(anadir(cero(), 2, 1), 3, 1)` — sea observacionalmente igual a agregar uno solo con el coeficiente sumado — `anadir(cero(), 5, 1)`. La representación no canónica queda "normalizada" a nivel observacional.

Separación de responsabilidades vía IMPORTS: `potencia` no figura en `OPERATIONS` ni en `AXIOMS` de `Polinomio`. Es una operación del tipo `Integer`, importada. `Polinomio` la *usa* en el axioma de evaluar pero su definición y sus axiomas ($\text{potencia}(x, 0) = 1$, $\text{potencia}(x, n) = x \cdot \text{potencia}(x, n-1)$) son responsabilidad de `Integer`. Esta es la forma correcta de modularizar: cada TDA conoce solo sus propias operaciones.

Spec validada con el analyzer del workspace: 0 errores, 0 warnings. Nota: el parser de Nereus no acepta ñ en identificadores, por eso `añadir` quedó como `anadir`.