Cp 5 - Clases
Curso 2024-2025

1. Fracción

Implemente una clase llamada Fraction para representar el comportamiento de una fracción. Una fracción consta de dos enteros, el numerador y el denominador, donde el denominador no puede ser cero. Dos fracciones se consideran iguales si la relación entre el numerador y el denominador es la misma (por ejemplo, $\frac{1}{2}$ y $\frac{2}{4}$ son equivalentes).

Funcionalidades a implementar

- **Obtener el numerador (simplificado)**: Devuelve el numerador de la fracción en su forma simplificada.
- **Obtener el denominador (simplificado)**: Devuelve el denominador de la fracción en su forma simplificada.
- **Representación en cadena de caracteres**: Devuelve una cadena que representa la fracción en el formato `num/den` (por ejemplo, `1/2`).
- **Suma de fracciones**: Realiza la suma de dos fracciones y devuelve el resultado como una nueva fracción.
- **Resta de fracciones**: Resta una fracción de otra y devuelve el resultado como una nueva fracción.
- **Multiplicación de fracciones**: Multiplica dos fracciones y devuelve el resultado como una nueva fracción.
- **División de fracciones**: Divide una fracción por otra y devuelve el resultado como una nueva fracción.
- **Comparación de fracciones**: Permite comparar dos fracciones para determinar si una es menor, mayor o igual a la otra.

Ejemplo de uso:

```
Fraction f1 = new Fraction(2, 4);
Fraction f2 = new Fraction(1, 3);

// Suma de fracciones
Fraction sum = f1 + f2;
Console.WriteLine(sum.ToString());
// Esperado: 5/6

// Resta de fracciones
Fraction diff = f1 - f2;
Console.WriteLine(diff.ToString());
// Esperado: 1/6

// Multiplicación de fracciones
Fraction product = f1 * f2;
Console.WriteLine(product.ToString());
// Esperado: 1/6

// División de fracciones
Fraction quotient = f1 / f2;
Console.WriteLine(quotient.ToString());
// Esperado: 3/2

// Comparación de fracciones
bool isEqual = f1 == f2;
Console.WriteLine(isEqual);
// Esperado: False
```

2. Polinomio

Implemente una clase Poly para representar el comportamiento de un polinomio. Para representar un polinomio de grado n con coeficientes enteros se puede usar un array, de tal manera que en la posición k del array esté el coeficiente de grado k del polinomio. Así, para representar al polinomio $p(x) = 2x + 1$ se puede usar el array $[1, 2]$ y para representar al polinomio $p(x) = x^3 - 5x + 1$ se puede usar el array $[1, -5, 0, 1]$.

Funcionalidades a implementar

- **Obtener el grado del polinomio**: Método que devuelve el grado del polinomio (índice más alto con coeficiente no nulo).
- **Obtener el coeficiente de una potencia específica**: Método que recibe un exponente k y devuelve el coeficiente de x^k. Si k es mayor que el grado del polinomio, debe devolver 0.
- **Obtener el string que representa al polinomio**: Método que construye una representación en texto del polinomio. Ejemplo: si el array de coeficientes es $[0, -2, 1]$, debe devolver el string $x^2 - 2x$. Los términos con coeficiente 0 no deben mostrarse, y los signos de cada término deben manejarse correctamente.
- **Comparar dos polinomios**: Métodos para compara dos polinomios $==, != , >=, >, <=, <$.
- **Sumar dos polinomios**: Método que recibe otro polinomio y devuelve un nuevo polinomio que es la suma de ambos.
- **Multiplicar dos polinomios**: Método que recibe otro polinomio y devuelve un nuevo polinomio que es el producto de ambos.

- **Multiplicar un polinomio por un escalar** e: Método que recibe un entero e y devuelve un nuevo polinomio resultante de multiplicar todos los coeficientes por e.
- * **Dividir dos polinomios**: Método que realiza la división entre dos polinomios, devolviendo el cociente como polinomio.
- * **Resto de dividir dos polinomios**: Método que realiza la división entre dos polinomios, devolviendo el residuo como polinomio.
- **Evaluar el polinomio en** x: Método que recibe un valor x y devuelve el resultado de evaluar el polinomio en ese valor.
- **Derivar el polinomio**: Método que devuelve un nuevo polinomio resultante de la derivada del polinomio actual, aplicando la regla de derivación $\frac{d(x^k)}{dx} = k \cdot x^{k-1}$.

Ejemplo de uso:

```
Poly p1 = new Poly([ 1, -5, 0, 1 ]); // x^3 - 5x + 1
Poly p2 = new Poly([ 2, 3 ]); // 2x + 3

// Obtener grado del polinomio
Console.WriteLine($"Grado de p1: {p1.Degree}");
// Esperado: Grado de p1: 3

// Obtener coeficiente de una potencia específica
Console.WriteLine($"Coeficiente de x^2 en p1: {p1[2]}");
// Esperado: Coeficiente de x^2 en p1: 0

// Suma de polinomios
Poly sum = p1 + p2;
Console.WriteLine(sum.ToString());
// Esperado: x^3 - 3x + 4

// Resta de polinomios
Poly diff = p1 - p2;
Console.WriteLine(diff.ToString());
// Esperado: x^3 - 7x - 2

// Multiplicación de polinomios
Poly product = p1 * p2;
Console.WriteLine(product.ToString());
// Esperado: 2x^4 - 7x^3 - 15x^2 + 3x + 3

// Evaluación del polinomio en un valor específico
int result = p1.Evaluate(2);
Console.WriteLine(result);
// Esperado: 9

// Derivada del polinomio
Poly derivative = p1.Derivative();
Console.WriteLine(derivative.ToString());
// Esperado: 3x^2 - 5
```

3. Conjunto

Implemente la clase Set para representar el comportamiento de un conjunto de enteros. Un conjunto es una colección de elementos únicos, es decir, no repetidos.

Funcionalidades a implementar

- **Obtener la cardinalidad**: Método que devuelve el número de elementos en el conjunto.
- **Añadir un elemento**: Método que agrega un nuevo elemento al conjunto. Si el elemento ya existe, no debe ser agregado de nuevo.
- **Eliminar un elemento**: Método que elimina un elemento específico del conjunto. Si el elemento no existe en el conjunto, no se realiza ninguna acción.
- **Vaciar el conjunto**: Método que elimina todos los elementos del conjunto, dejándolo vacío.
- **Pertenencia**: Método que verifica si un elemento específico está presente en el conjunto ($x \in B$).
- **Intersección**: Método que devuelve un nuevo conjunto con los elementos comunes entre dos conjuntos ($A \cap B$).
- **Unión**: Método que devuelve un nuevo conjunto con todos los elementos de dos conjuntos ($A \cup B$),
- **Diferencia**: Método que devuelve un nuevo conjunto con los elementos de A que no están en B ($A - B$).
- **Subconjunto**: Método que determina si un conjunto está contenido dentro de otro ($A \subseteq B$).
- **Igualdad**: Método que compara si dos conjuntos son iguales ($A = B$), es decir, si tienen exactamente los mismos elementos.

Ejemplo de uso

```
Set s1 = new Set();
s1.Add(1);
s1.Add(2);
s1.Add(3);

Set s2 = new Set([2, 3, 4]);

// Obtener cardinalidad
Console.WriteLine($"Cardinalidad de s1: {s1.Cardinality}");
// Esperado: Cardinalidad de s1: 3

// Agregar un elemento
s1.Add(4);
Console.WriteLine($"Cardinalidad de s1 después de agregar 4: {s1.Cardinality}");
// Esperado: Cardinalidad de s1 después de agregar 4: 4

// Eliminar un elemento
s1.Remove(2);
Console.WriteLine($"Cardinalidad de s1 después de eliminar 2: {s1.Cardinality}");
// Esperado: Cardinalidad de s1 después de eliminar 2: 3

// Pertenencia
bool contains = s2.Contains(3);
Console.WriteLine($"¿s2 contiene 3?: {contains}");
// Esperado: ¿s2 contiene 3?: True

// Intersección
Set intersection = Set.Intersection(s1, s2);
```

```
Console.WriteLine($"Intersección de s1 y s2: [{string.Join(", ", intersection.Elements)}]");
// Esperado: Intersección de s1 y s2: [3, 4]

// Unión
Set union = Set.Union(s1, s2);
Console.WriteLine($"Unión de s1 y s2: [{string.Join(", ", union.Elements)}]");
// Esperado: Unión de s1 y s2: [1, 2, 3, 4]

// Diferencia
Set difference = Set.Difference(s2, s1);
Console.WriteLine($"Diferencia de s2 y s1: [{string.Join(", ", difference.Elements)}]");
// Esperado: Diferencia de s2 y s1: [2]

// Vaciar
s1.Clear()
Console.WriteLine($"Cardinalidad de s1 después de vaciar: {s1.Cardinality}");
// Esperado: Cardinalidad de s1 después de vaciar: 0

// Subconjunto
bool isSubset = Set.IsSubset(s1, s2);
Console.WriteLine($"¿s1 es subconjunto de s2?: {isSubset}");
// Esperado: ¿s1 es subconjunto de s2?: True

// Igualdad
bool isEqual = s1 == s2;
Console.WriteLine($"¿s1 es igual a s2?: {isEqual}");
// Esperado: ¿s1 es igual a s2?: False
```

4. Entero grande

Implemente una clase BigInt para representar números enteros no negativos potencialmente grandes (pueden tener más de 1000 cifras). Esta clase debe permitir realizar operaciones aritméticas básicas de forma precisa, sin limitarse al rango de tipos numéricos tradicionales.

Funcionalidades a implementar

- **Obtener representación en cadena de caracteres**: Devuelve el número como un `string` que representa el valor completo del entero grande.
- **Suma**: Implementa la suma de dos objetos BigInt.
- **Resta**: Calcula la diferencia entre dos BigInt. **Nota:** asegúrese de que no se permitan resultados negativos.
- **Multiplicación**: Realiza la multiplicación entre dos objetos BigInt.
- * **División**: Implementa la división entre dos BigInt, devolviendo solo la parte entera del cociente.
- * **Resto de la división**: Calcula el residuo de la división entre dos BigInt.
- **Potenciación**: Permite elevar el número a una potencia `b` de tipo `int`.
- **Comparación**: Métodos para compara dos objetos BigInt $==, !=, >=, >, <=, <$.

Ejemplo de uso:

```
BigInt bigInt1 = new BigInt("100000000000000000000");
BigInt bigInt2 = new BigInt("100000000000000000000");

// Suma de enteros grandes
BigInt sum = bigInt1 + bigInt2;
Console.WriteLine(sum.ToString());
// Esperado: 110000000000000000000

// Resta de enteros grandes
BigInt diff = bigInt1 - bigInt2;
Console.WriteLine(diff.ToString());
// Esperado: 90000000000000000000

// Multiplicación de enteros grandes
BigInt product = bigInt1 * bigInt2;
Console.WriteLine(product.ToString());
// Esperado: 10000000000000000000000000000000000000000

// Potenciación
int exponent = 2;
BigInt pow = BigInt.Pow(bigInt2, exponent);
Console.WriteLine(pow.ToString());
// Esperado: 10000000000000000000000000000000000000000

// Comparación de enteros grandes
Console.WriteLine(bigInt1 < bigInt2);
// Esperado: False
```

5. Fecha

Implemente una clase llamada Date para representar una fecha, que incluya día, mes y año. Esta clase debe permitir realizar operaciones y consultas básicas sobre fechas.

Funcionalidades a implementar

- **Obtener el día**: Devuelve el día correspondiente de la fecha.
- **Obtener el mes**: Devuelve el mes correspondiente de la fecha.
- **Obtener el año**: Devuelve el año correspondiente de la fecha.
- **Obtener el día de la semana**: Calcula y devuelve el día de la semana en el que cae la fecha (por ejemplo, Lunes, Martes).
- **Comparar fechas**: Permite comparar dos fechas, indicando si una fecha es anterior, igual o posterior a otra.
- **Avanzar un día**: Ajusta la fecha actual avanzando un día, manejando correctamente los cambios de mes y año cuando sea necesario.
- **Retroceder un día**: Ajusta la fecha actual retrocediendo un día, manejando correctamente los cambios de mes y año cuando sea necesario.
- **Calcular la diferencia en días**: Calcula y devuelve la cantidad de días entre dos fechas dadas.
- **Representación en cadena de caracteres**: Devuelve la fecha en formato `día/mes/año` (por ejemplo, `12/10/2024`).

Ejemplo de uso:

```
Date date1 = new Date(12, 10, 2024);
Date date2 = new Date(15, 10, 2024);

// Obtener el día, mes, año y el día de la semana
int day = date1.Day;                // Esperado: 12
int month = date1.Month;            // Esperado: 10
int year = date1.Year;              // Esperado: 2024
string dayOfWeek = date1.DayOfWeek; // Esperado: "Saturday"

// Comparar fechas con operadores
if (date1 < date2)
{
    Console.WriteLine($"{date1} es anterior a {date2}");
}
else if (date1 > date2)
{
    Console.WriteLine($"{date1} es posterior a {date2}");
}
else
{
    Console.WriteLine($"{date1} es igual a {date2}");
}
// Esperado: "12/10/2024 es anterior a 15/10/2024"

// Avanzar un día
date1.AdvanceOneDay();
Console.WriteLine(date1.ToString()); // Esperado: 13/10/2024

// Retroceder un día
date2.RetreatOneDay();
Console.WriteLine(date2.ToString()); // Esperado: 14/10/2024

// Calcular la diferencia en días entre dos fechas
int daysDifference = Date.DaysBetween(date1, date2);
Console.WriteLine($"Días de diferencia: {daysDifference}"); // Esperado: 1
```

6. Libro con reseñas

Estamos desarrollando un sistema para gestionar una biblioteca digital que permita a los usuarios agregar reseñas y calificaciones a los libros, y necesitamos tu ayuda para implementar una clase que represente los libros en este sistema. Tu tarea es diseñar una clase Book que encapsule de manera segura toda la información de un libro.

- **Propiedades (de solo lectura):**
  - `Title` (string): el título del libro.
  - `Author` (string): el nombre del autor.
  - `Genre` (enum): el género del libro (Ficción, No Ficción, Ciencia, Historia, etc).
  - `ISBN` (string): el identificador único del libro, que debe ser inmutable tras la creación del objeto.
- **Métodos Públicos:**
  - `AddReview(string reviewText, int rating)`: agrega una reseña al libro con un texto y una calificación (de 1 a 5).
  - `GetReviews()`: devuelve todas las reseñas del libro como un solo string, mostrando el texto y la calificación de cada reseña, separadas por saltos de línea.

- `GetAverageRating()`: calcula y devuelve el promedio de todas las calificaciones de las reseñas.
- `GetSummary()`: devuelve un breve resumen del libro en el siguiente formato:

```
Title: [Título]
Author: [Autor]
Genre: [Género]
ISBN: [ISBN]
```

Ejemplo de uso:

```
Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald",
                     Genre.Fiction, "978-0743273565");
book.AddReview("Amazing classic!", 5);
book.AddReview("A timeless tale of love and loss.", 4);

Console.WriteLine(book.GetSummary());
// Esperado:
// Title: The Great Gatsby
// Author: F. Scott Fitzgerald
// Genre: Fiction
// ISBN: 978-0743273565

Console.WriteLine("Reviews:");
Console.WriteLine(book.GetReviews());
// Esperado:
// Review 1: Amazing classic! (Rating: 5)
// Review 2: A timeless tale of love and loss. (Rating: 4)"

Console.WriteLine("Average Rating: " + book.GetAverageRating());
// Esperado:
// Average Rating: 4.5
```