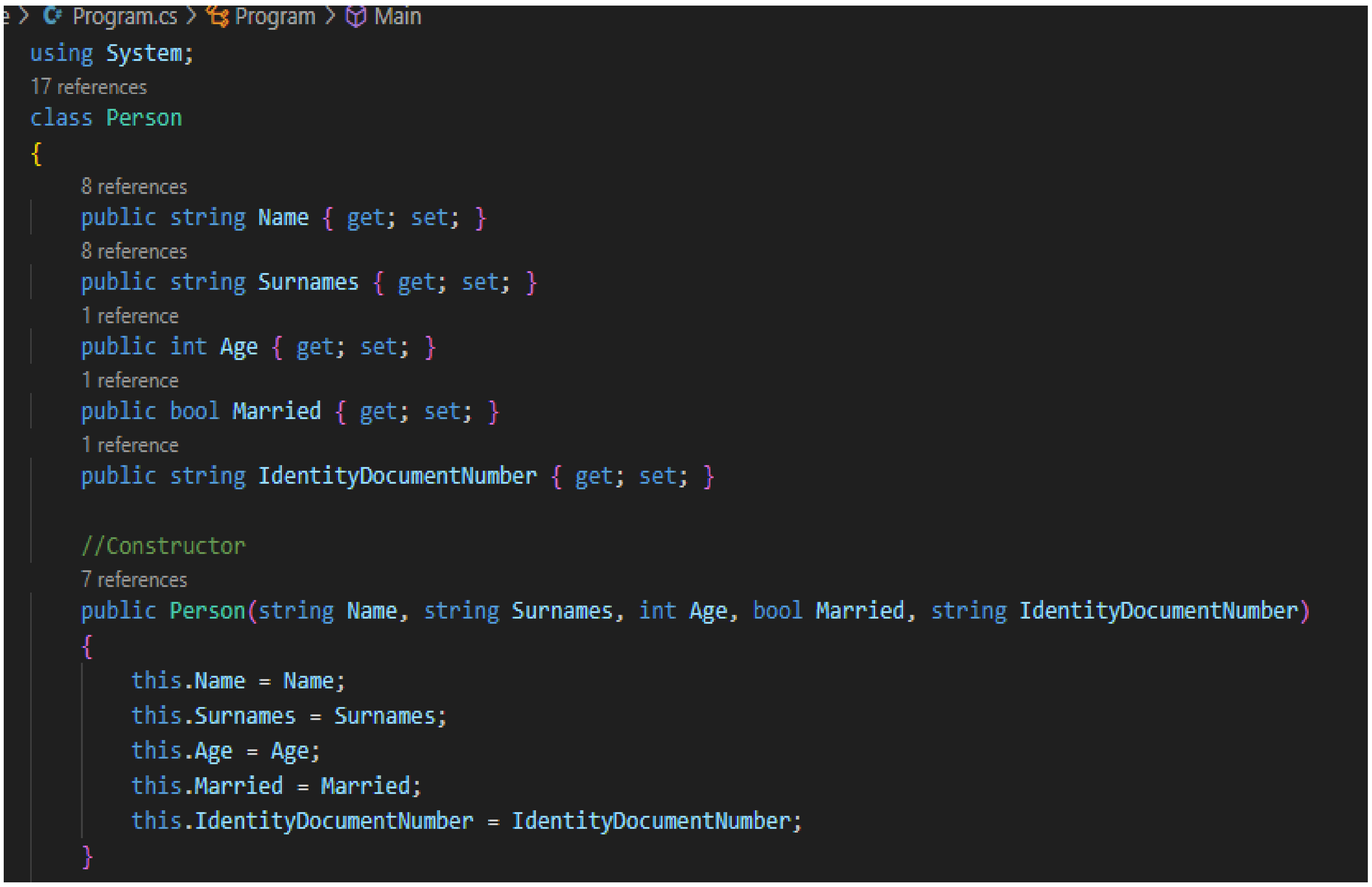
Programación #1

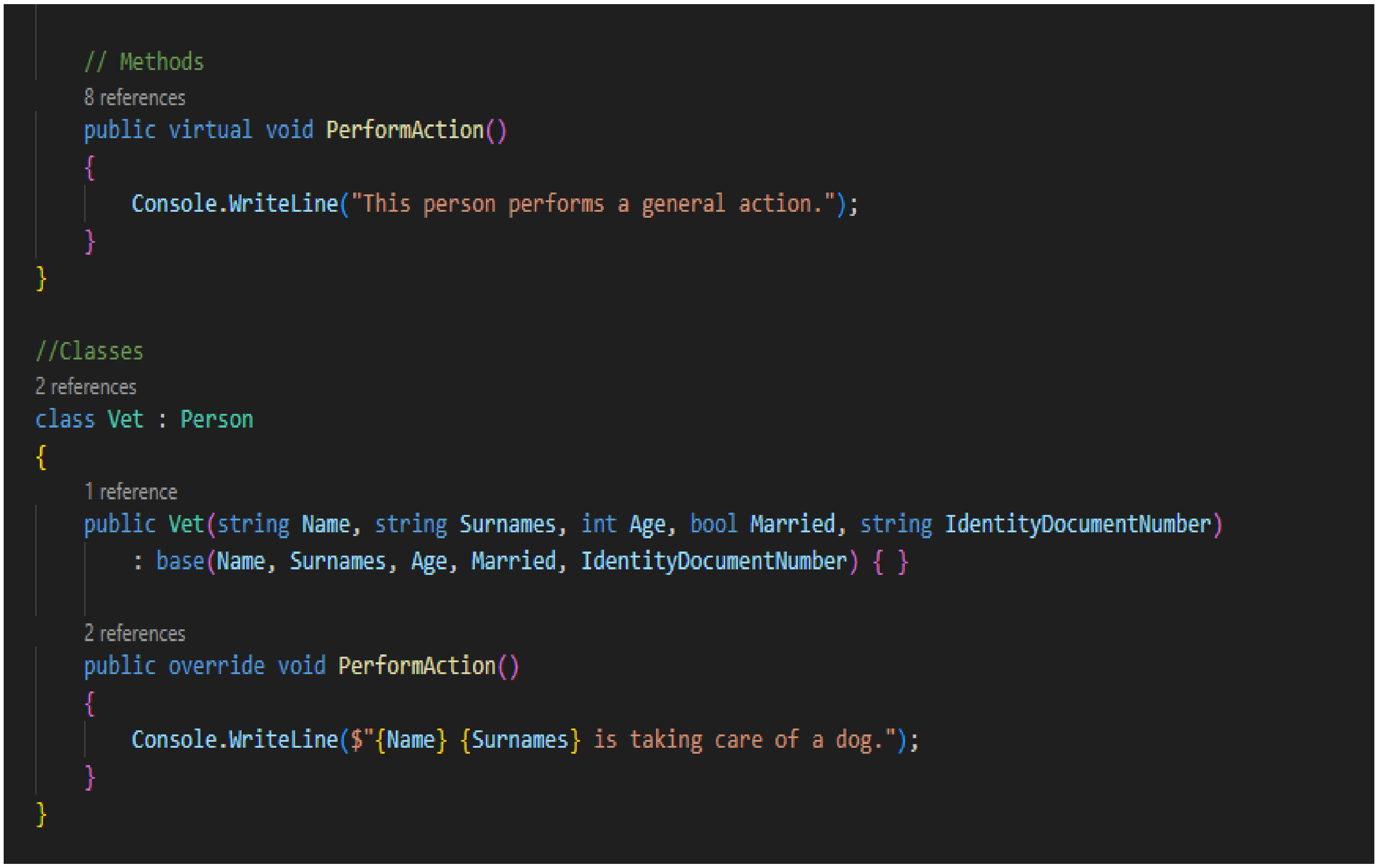
Nombre: Lisbel Martínez **Matrícula:** 2024 – 1717 **Día de clases:** Lunes

Tarea 5.
Contestar las preguntas dentro del cuestionario, puede ser transcribiendo el código en aquellos casos que corresponda, o simplemente hacer una captura del código.

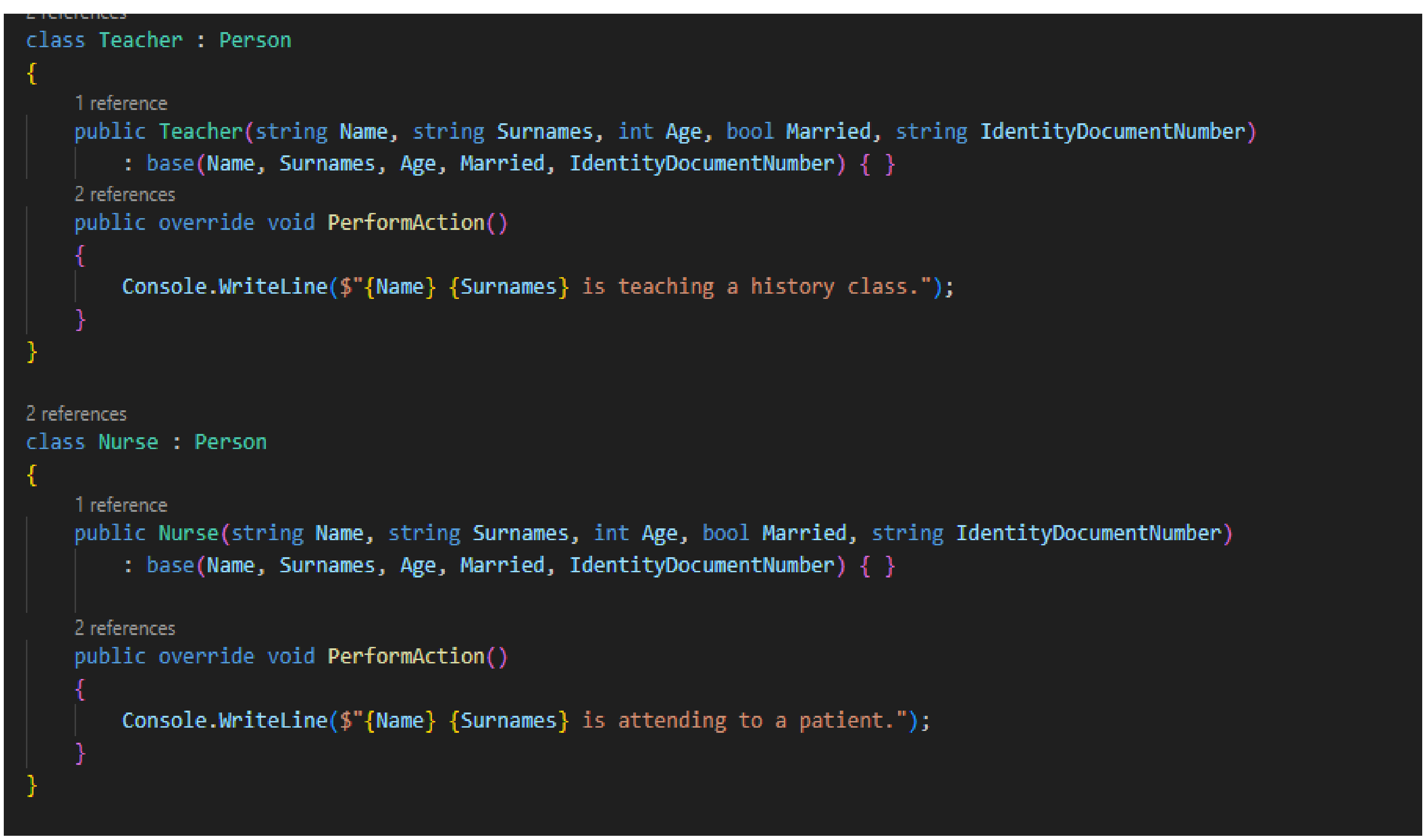
1. Considera estás desarrollando un programa donde necesitas trabajar con objetos de tipo Persona. Define una clase Persona, pero en este caso considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), numeroDocumentoIdentidad (String) y 3 métodos como acciones diferentes por persona de acuerdo a una profesión. Define un constructor y los métodos para poder establecer y obtener los valores de los atributos. Mínimo 7 personas diferentes con acciones diferentes.



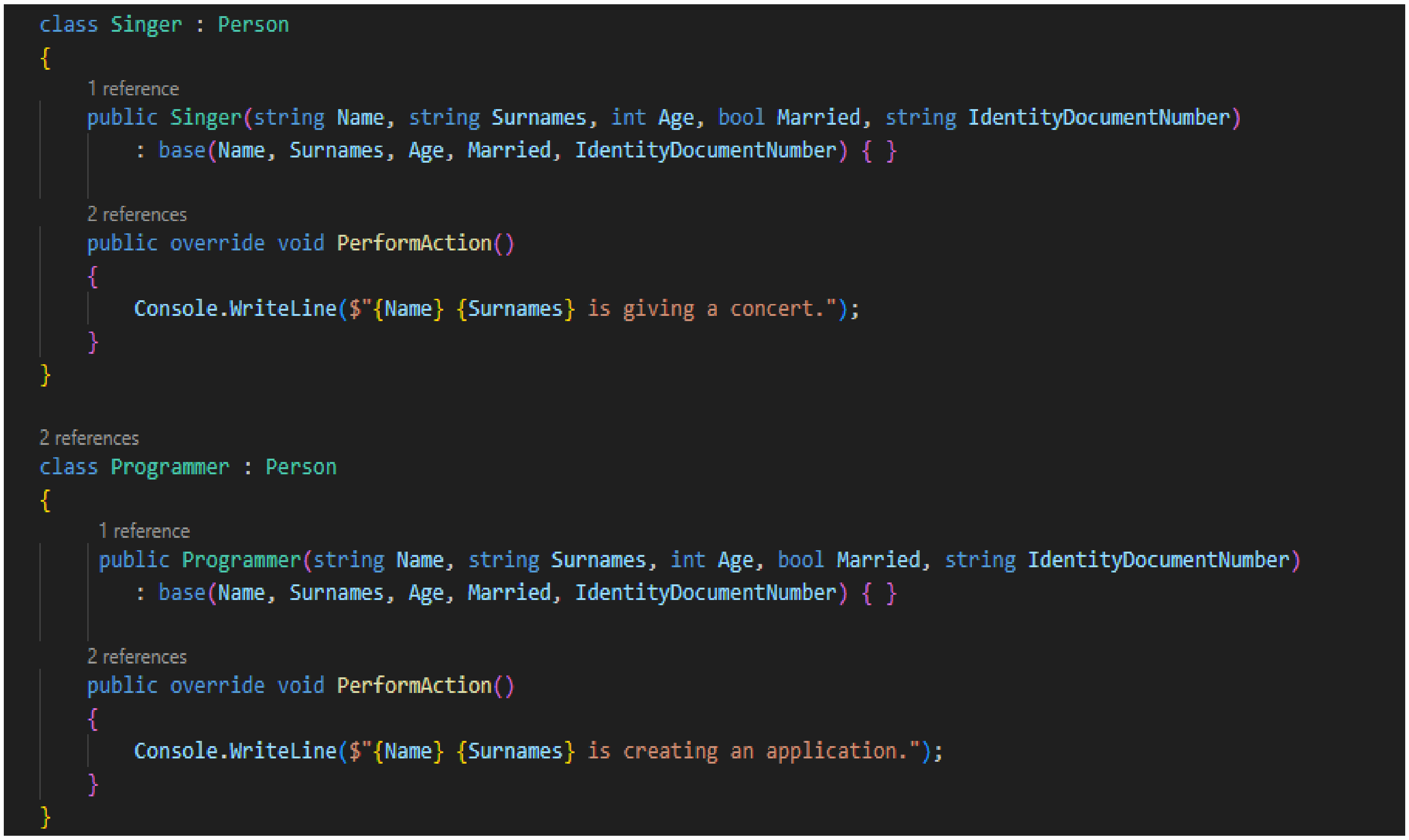
```
using System;
17 references
class Person
{
    8 references
    public string Name { get; set; }
    8 references
    public string Surnames { get; set; }
    1 reference
    public int Age { get; set; }
    1 reference
    public bool Married { get; set; }
    1 reference
    public string IdentityDocumentNumber { get; set; }

    //Constructor
    7 references
    public Person(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
    {
        this.Name = Name;
        this.Surnames = Surnames;
        this.Age = Age;
        this.Married = Married;
        this.IdentityDocumentNumber = IdentityDocumentNumber;
    }
```



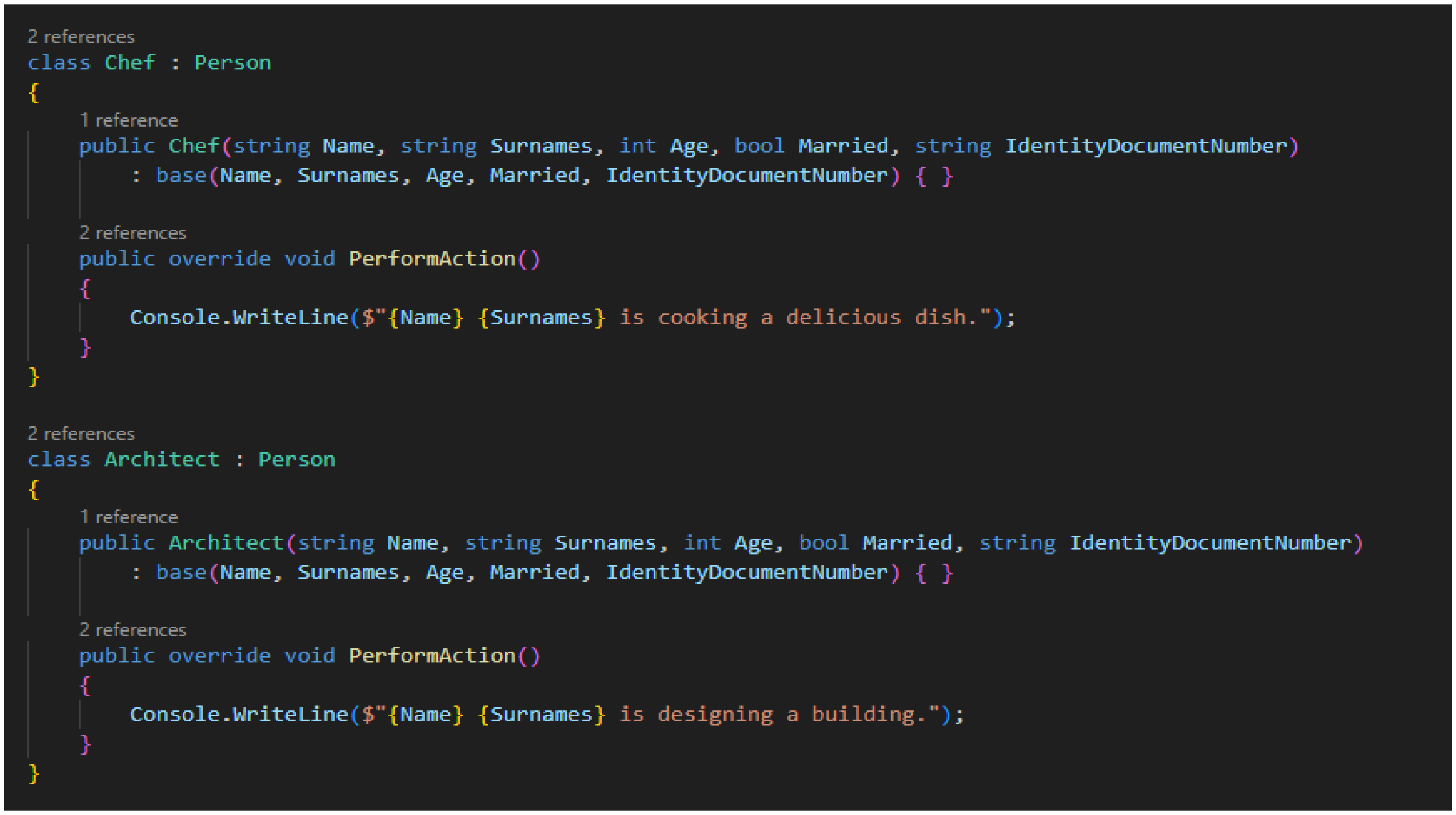
```
    // Methods
    8 references
    public virtual void PerformAction()
    {
        Console.WriteLine("This person performs a general action.");
    }
}

//Classes
2 references
class Vet : Person
{
    1 reference
    public Vet(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is taking care of a dog.");
    }
}
```



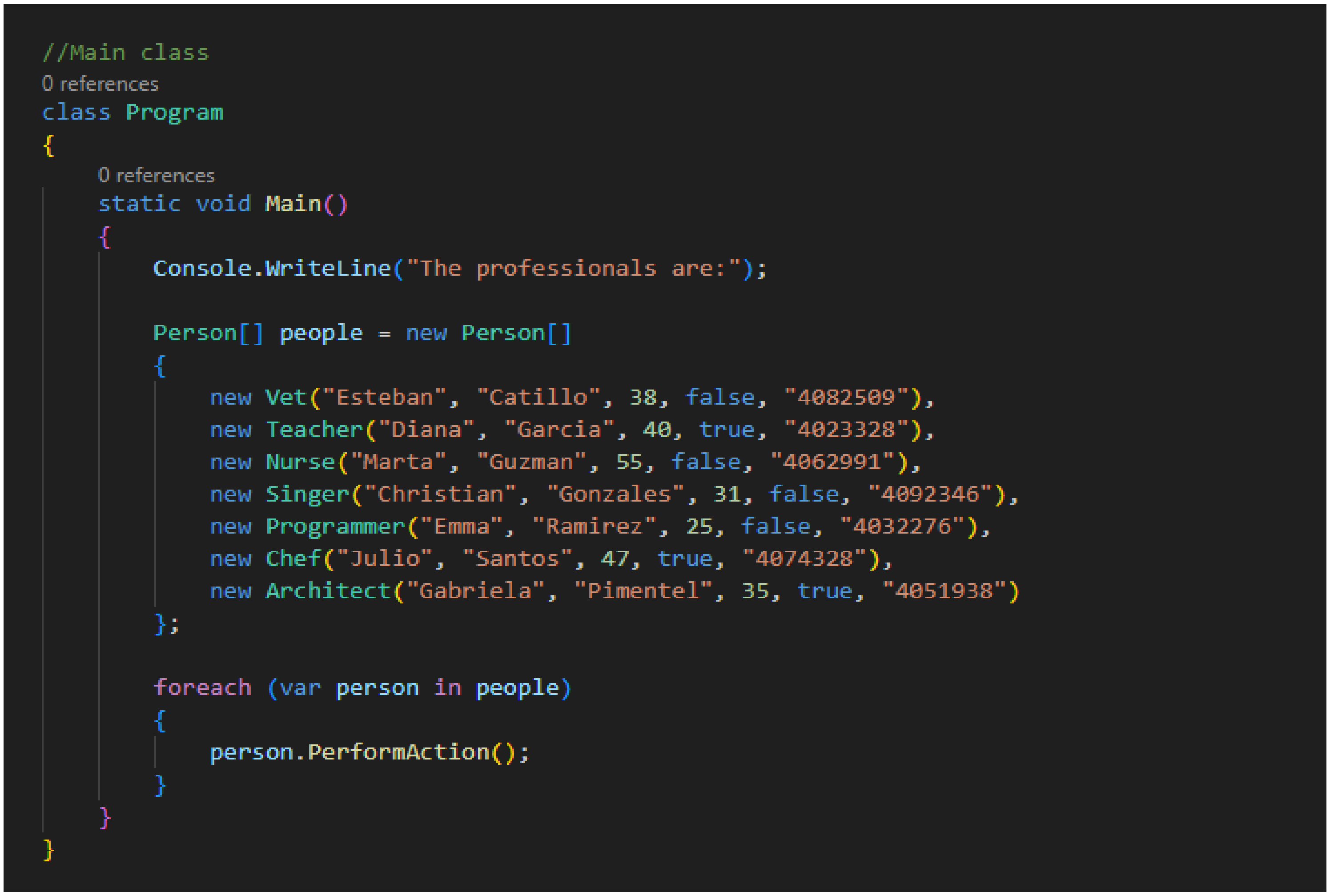
```
class Teacher : Person
{
    1 reference
    public Teacher(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }
    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is teaching a history class.");
    }
}

2 references
class Nurse : Person
{
    1 reference
    public Nurse(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is attending to a patient.");
    }
}
```



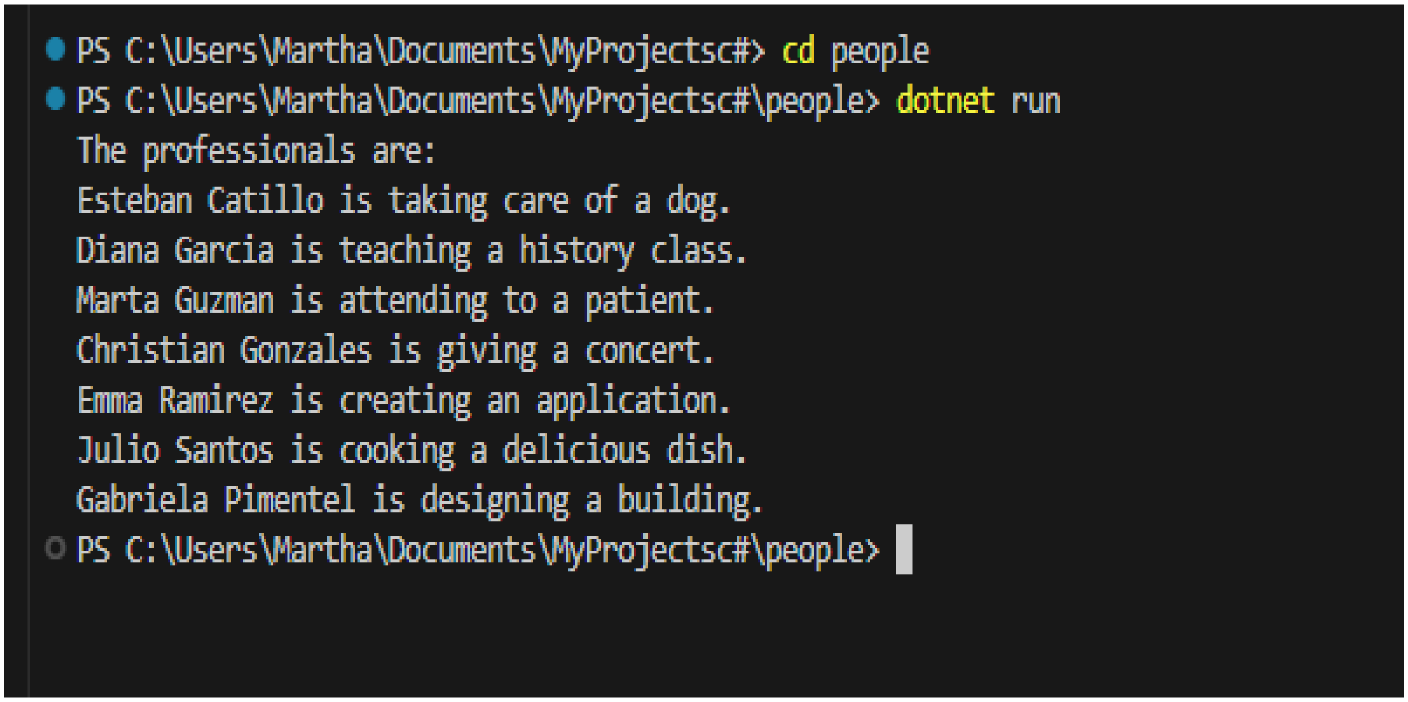
```
class Singer : Person
{
    1 reference
    public Singer(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is giving a concert.");
    }
}

2 references
class Programmer : Person
{
     1 reference
     public Programmer(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is creating an application.");
    }
}
```



```
2 references
class Chef : Person
{
    1 reference
    public Chef(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is cooking a delicious dish.");
    }
}

2 references
class Architect : Person
{
    1 reference
    public Architect(string Name, string Surnames, int Age, bool Married, string IdentityDocumentNumber)
        : base(Name, Surnames, Age, Married, IdentityDocumentNumber) { }

    2 references
    public override void PerformAction()
    {
        Console.WriteLine($"{Name} {Surnames} is designing a building.");
    }
}
```

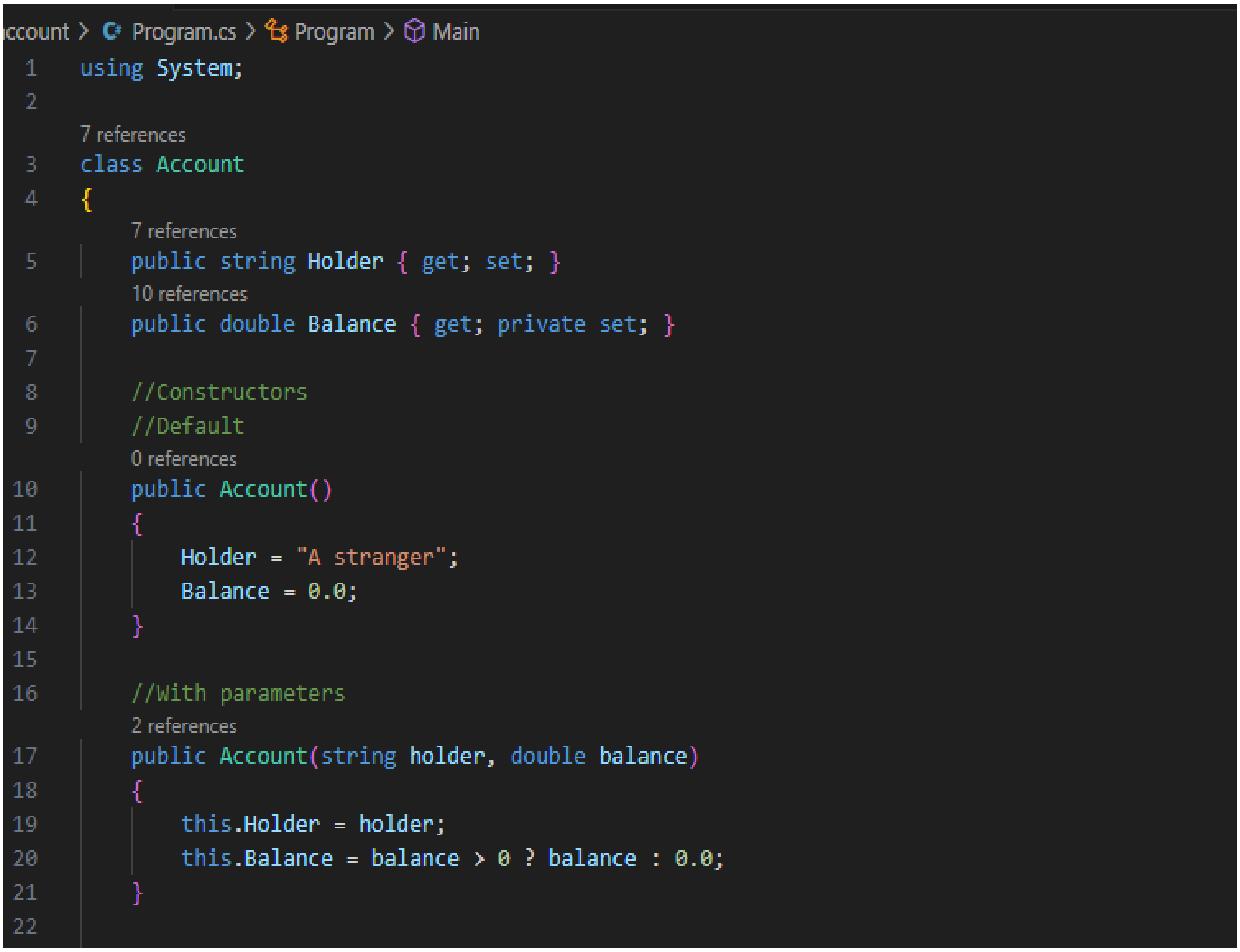


```
//Main class
0 references
class Program
{
    0 references
    static void Main()
    {
        Console.WriteLine("The professionals are:");

        Person[] people = new Person[]
        {
            new Vet("Esteban", "Catillo", 38, false, "4082509"),
            new Teacher("Diana", "Garcia", 40, true, "4023328"),
            new Nurse("Marta", "Guzman", 55, false, "4062991"),
            new Singer("Christian", "Gonzales", 31, false, "4092346"),
            new Programmer("Emma", "Ramirez", 25, false, "4032276"),
            new Chef("Julio", "Santos", 47, true, "4074328"),
            new Architect("Gabriela", "Pimentel", 35, true, "4051938")
        };

        foreach (var person in people)
        {
            person.PerformAction();
        }
    }
}
```

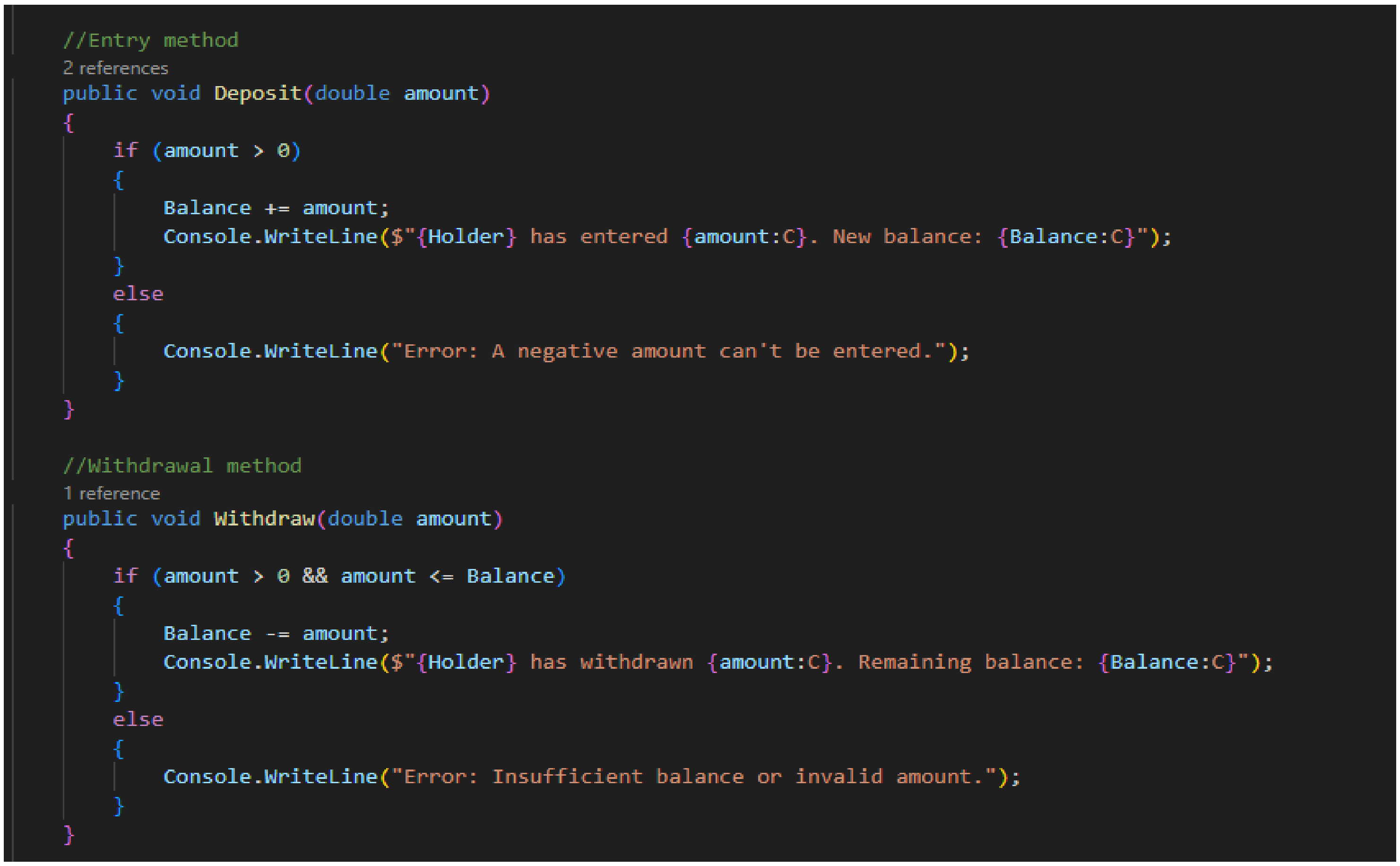


```
PS C:\Users\Martha\Documents\MyProjectsc#> cd people
PS C:\Users\Martha\Documents\MyProjectsc#\people> dotnet run
The professionals are:
Esteban Catillo is taking care of a dog.
Diana Garcia is teaching a history class.
Marta Guzman is attending to a patient.
Christian Gonzales is giving a concert.
Emma Ramirez is creating an application.
Julio Santos is cooking a delicious dish.
Gabriela Pimentel is designing a building.
PS C:\Users\Martha\Documents\MyProjectsc#\people>
```

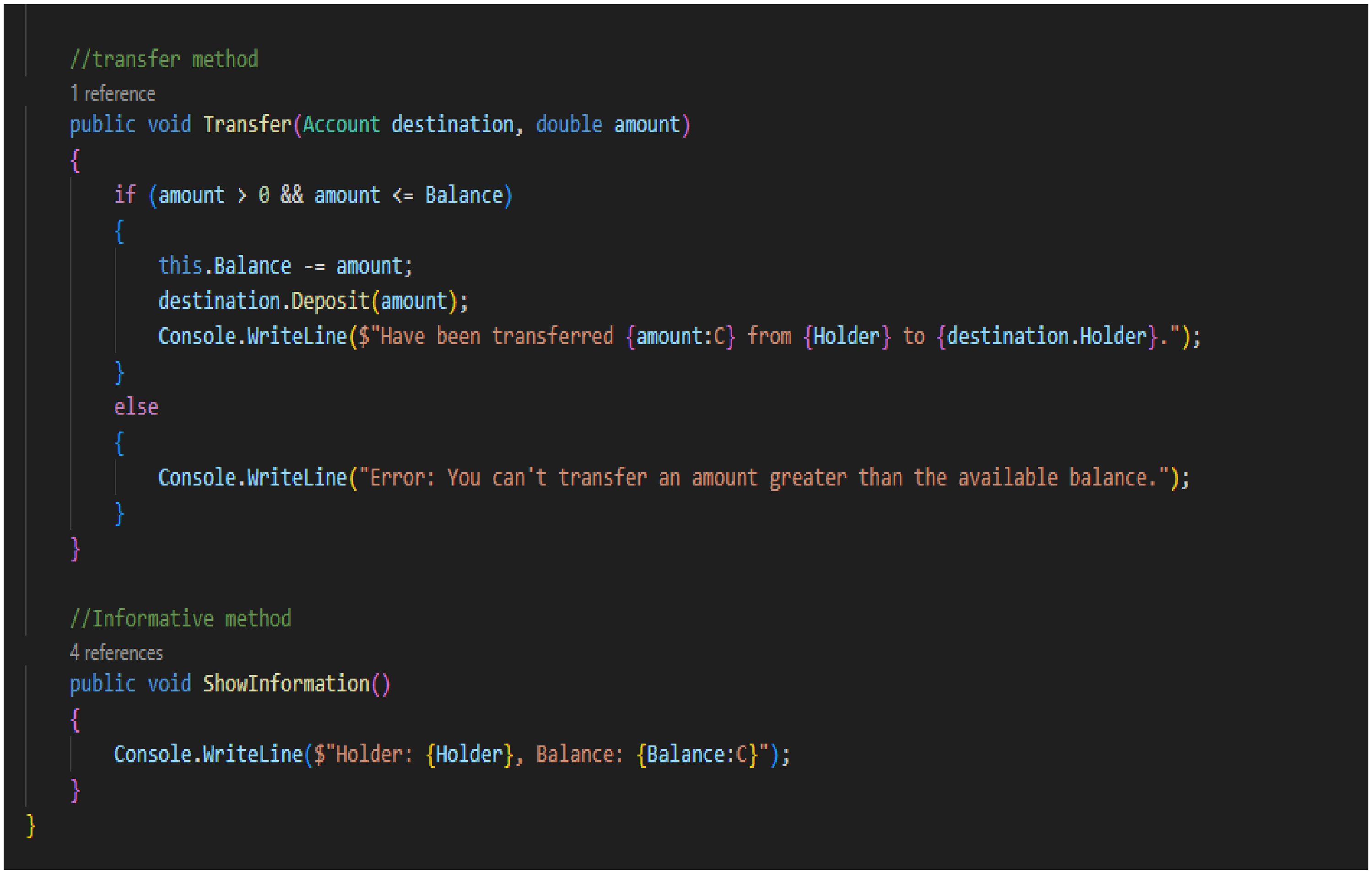
2. Crea una clase Cuenta con los métodos ingreso, reintegro y transferencia. La clase contendrá un constructor por defecto, un constructor con parámetros y los métodos getters y setters para mostrar e ingresar.



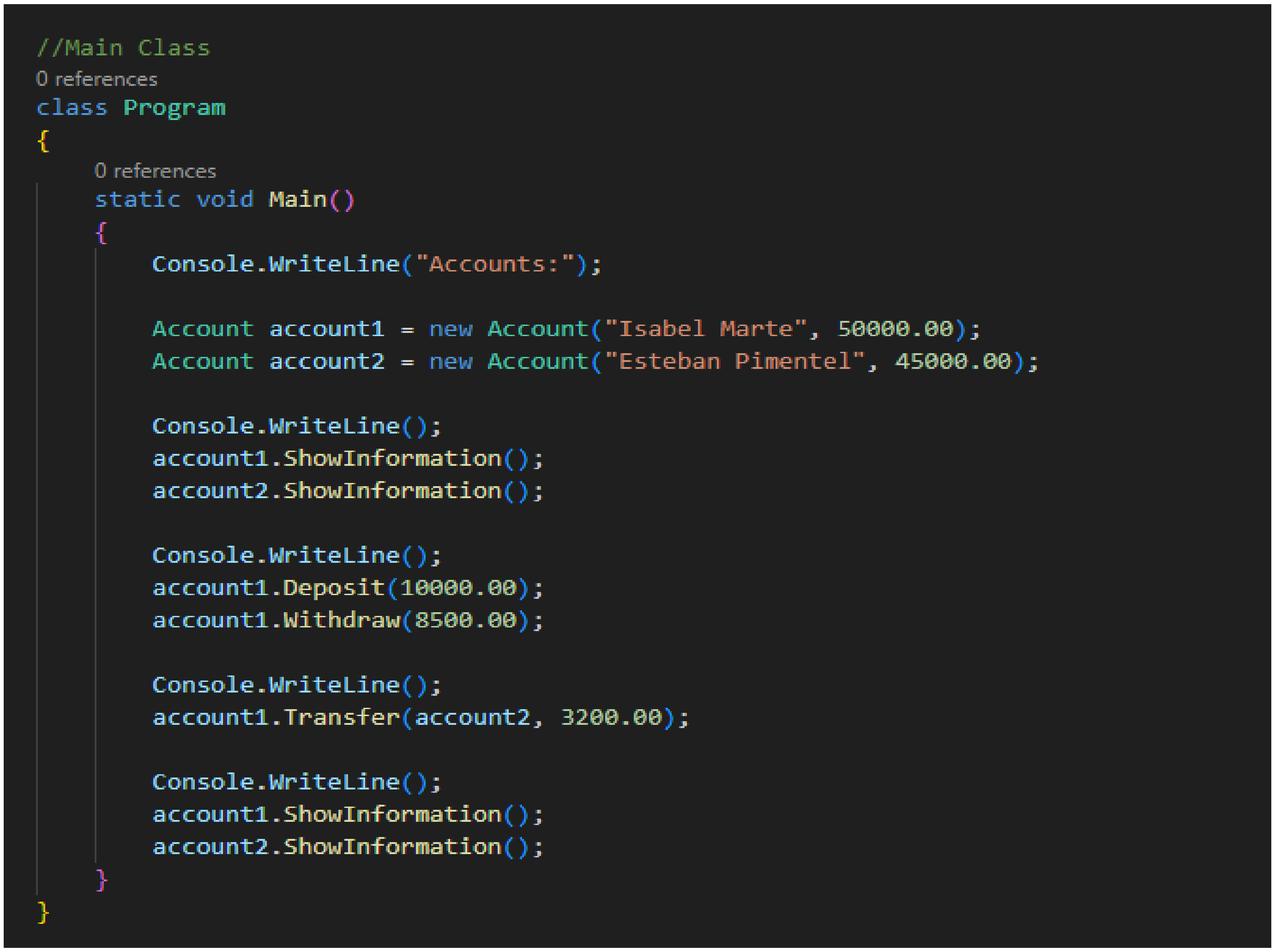
```
using System;

7 references
class Account
{
    7 references
    public string Holder { get; set; }
    10 references
    public double Balance { get; private set; }

    //Constructors
    //Default
    0 references
    public Account()
    {
        Holder = "A stranger";
        Balance = 0.0;
    }

    //With parameters
    2 references
    public Account(string holder, double balance)
    {
        this.Holder = holder;
        this.Balance = balance > 0 ? balance : 0.0;
    }

```



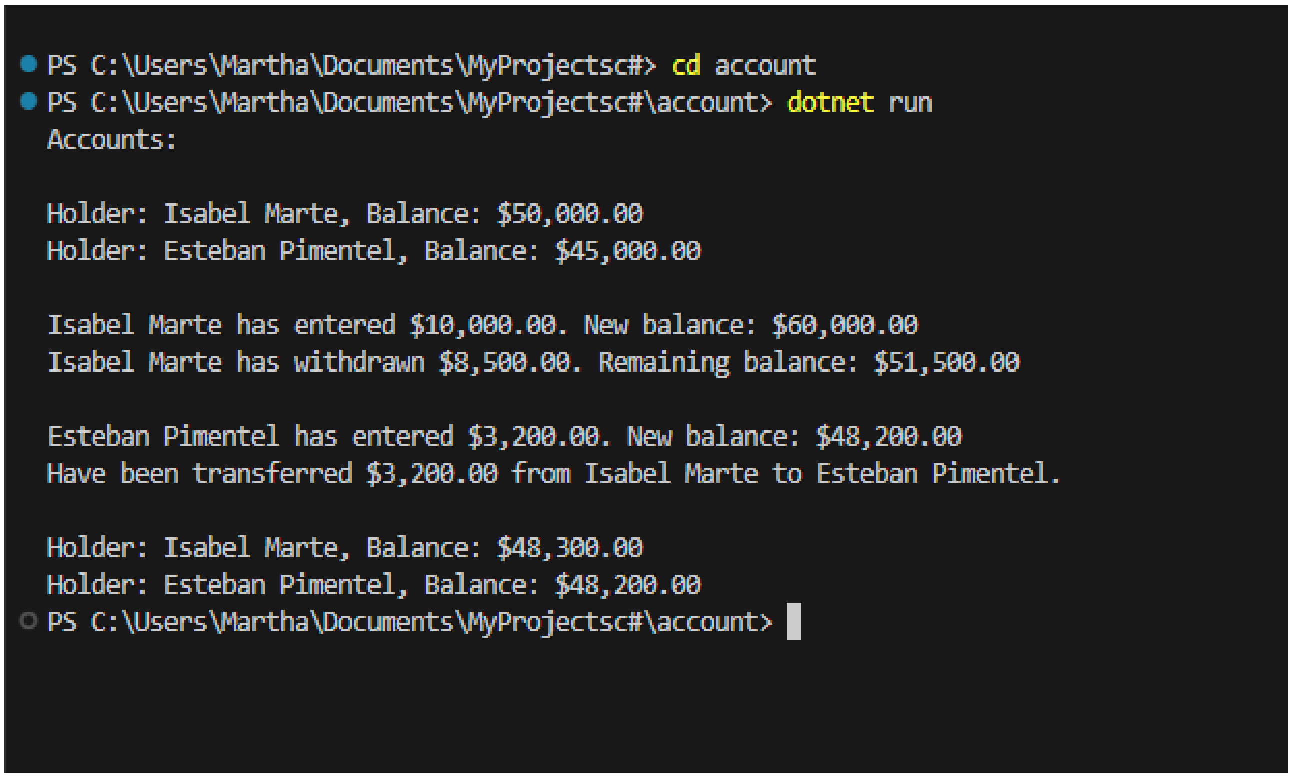
```
    //Entry method
    2 references
    public void Deposit(double amount)
    {
        if (amount > 0)
        {
            Balance += amount;
            Console.WriteLine($"{Holder} has entered {amount:C}. New balance: {Balance:C}");
        }
        else
        {
            Console.WriteLine("Error: A negative amount can't be entered.");
        }
    }

    //Withdrawal method
    1 reference
    public void Withdraw(double amount)
    {
        if (amount > 0 && amount <= Balance)
        {
            Balance -= amount;
            Console.WriteLine($"{Holder} has withdrawn {amount:C}. Remaining balance: {Balance:C}");
        }
        else
        {
            Console.WriteLine("Error: Insufficient balance or invalid amount.");
        }
    }
```



```
    //transfer method
    1 reference
    public void Transfer(Account destination, double amount)
    {
        if (amount > 0 && amount <= Balance)
        {
            this.Balance -= amount;
            destination.Deposit(amount);
            Console.WriteLine($"Have been transferred {amount:C} from {Holder} to {destination.Holder}.");
        }
        else
        {
            Console.WriteLine("Error: You can't transfer an amount greater than the available balance.");
        }
    }

    //Informative method
    4 references
    public void ShowInformation()
    {
        Console.WriteLine($"Holder: {Holder}, Balance: {Balance:C}");
    }
}
```

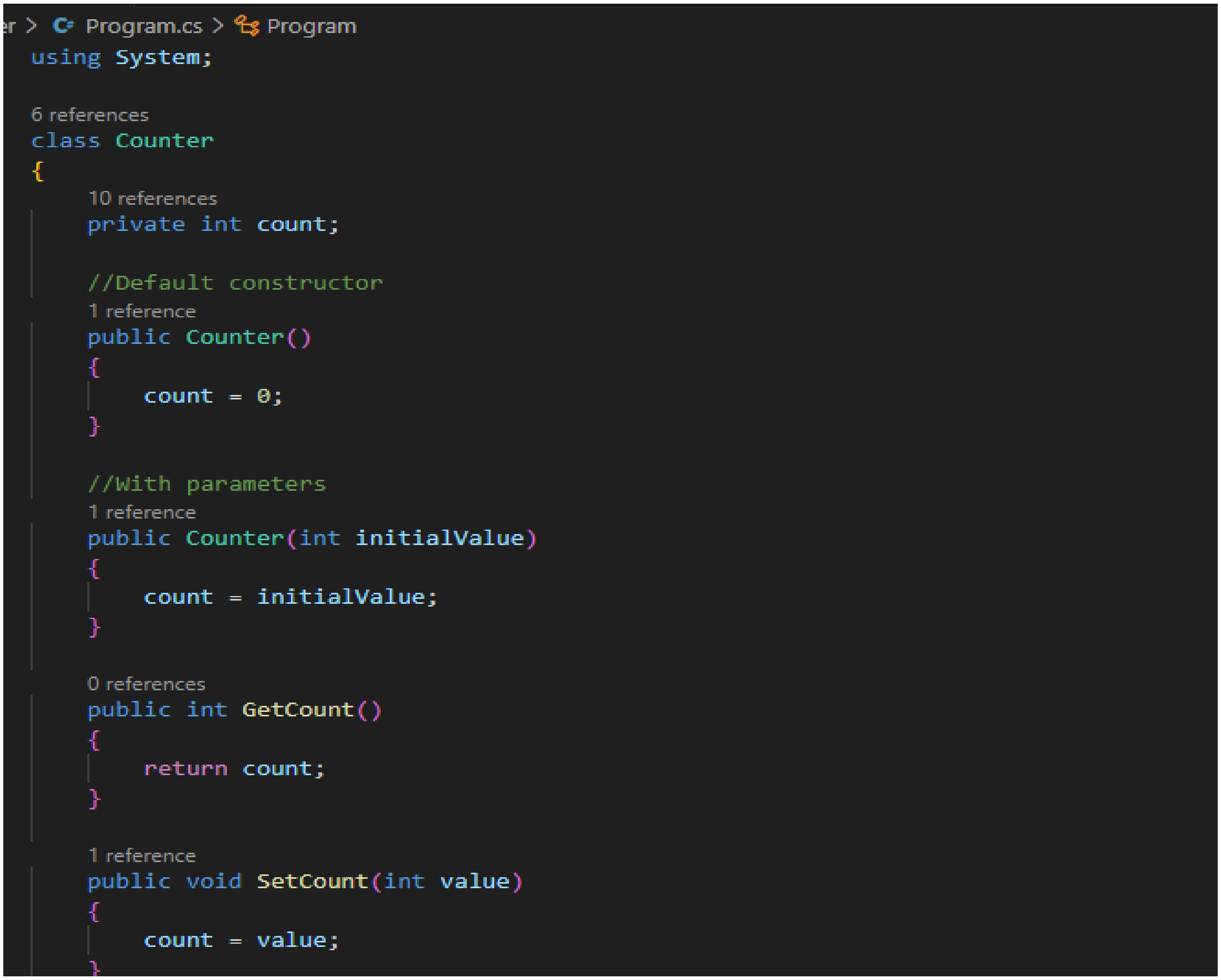


```
//Main Class
0 references
class Program
{
    0 references
    static void Main()
    {
        Console.WriteLine("Accounts:");

        Account account1 = new Account("Isabel Marte", 50000.00);
        Account account2 = new Account("Esteban Pimentel", 45000.00);

        Console.WriteLine();
        account1.ShowInformation();
        account2.ShowInformation();

        Console.WriteLine();
        account1.Deposit(10000.00);
        account1.Withdraw(8500.00);

        Console.WriteLine();
        account1.Transfer(account2, 3200.00);

        Console.WriteLine();
        account1.ShowInformation();
        account2.ShowInformation();
    }
}
```

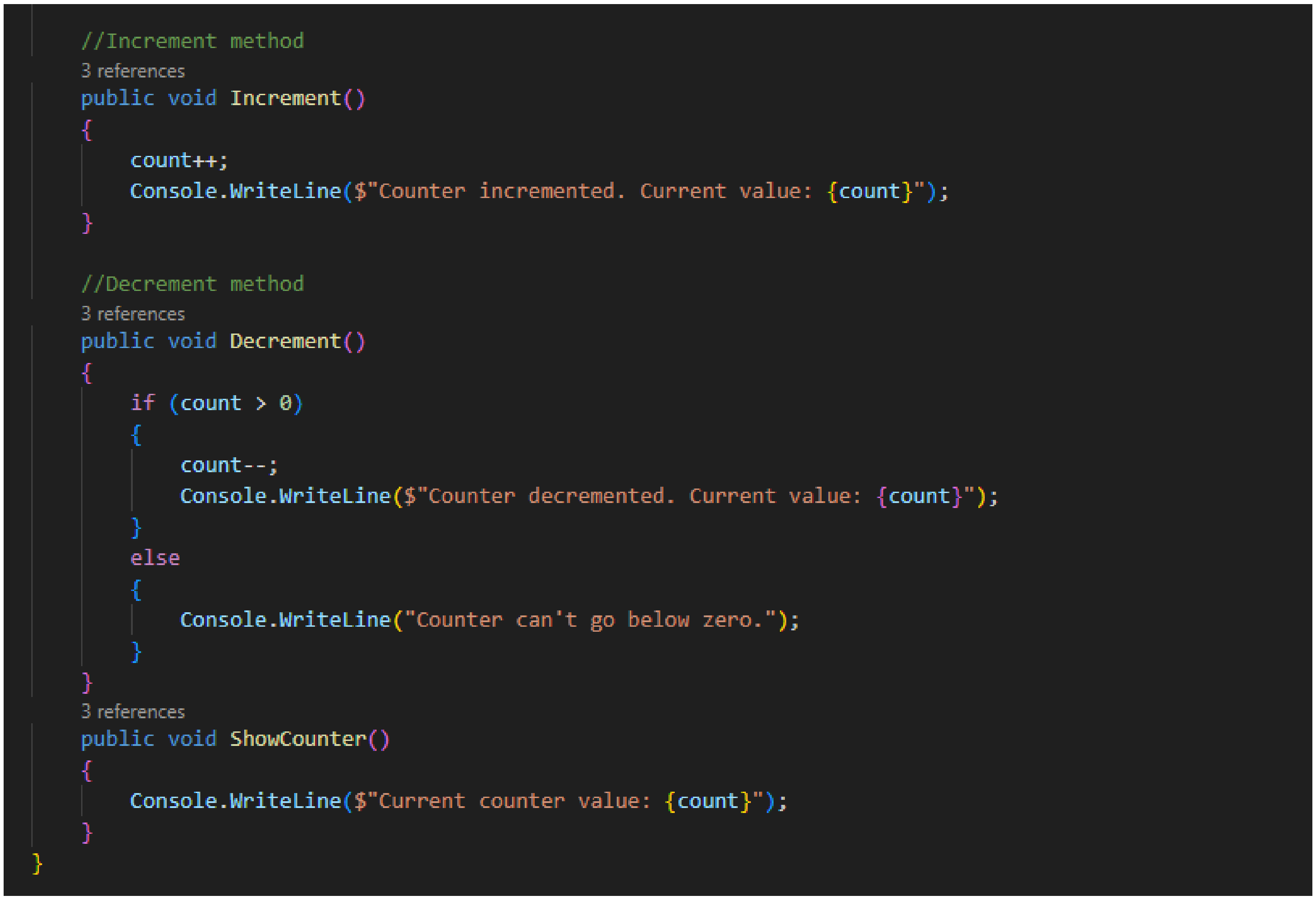


```
● PS C:\Users\Martha\Documents\MyProjectsc#> cd account
● PS C:\Users\Martha\Documents\MyProjectsc#\account> dotnet run
  Accounts:

  Holder: Isabel Marte, Balance: $50,000.00
  Holder: Esteban Pimentel, Balance: $45,000.00

  Isabel Marte has entered $10,000.00. New balance: $60,000.00
  Isabel Marte has withdrawn $8,500.00. Remaining balance: $51,500.00

  Esteban Pimentel has entered $3,200.00. New balance: $48,200.00
  Have been transferred $3,200.00 from Isabel Marte to Esteban Pimentel.

  Holder: Isabel Marte, Balance: $48,300.00
  Holder: Esteban Pimentel, Balance: $48,200.00
○ PS C:\Users\Martha\Documents\MyProjectsc#\account>
```

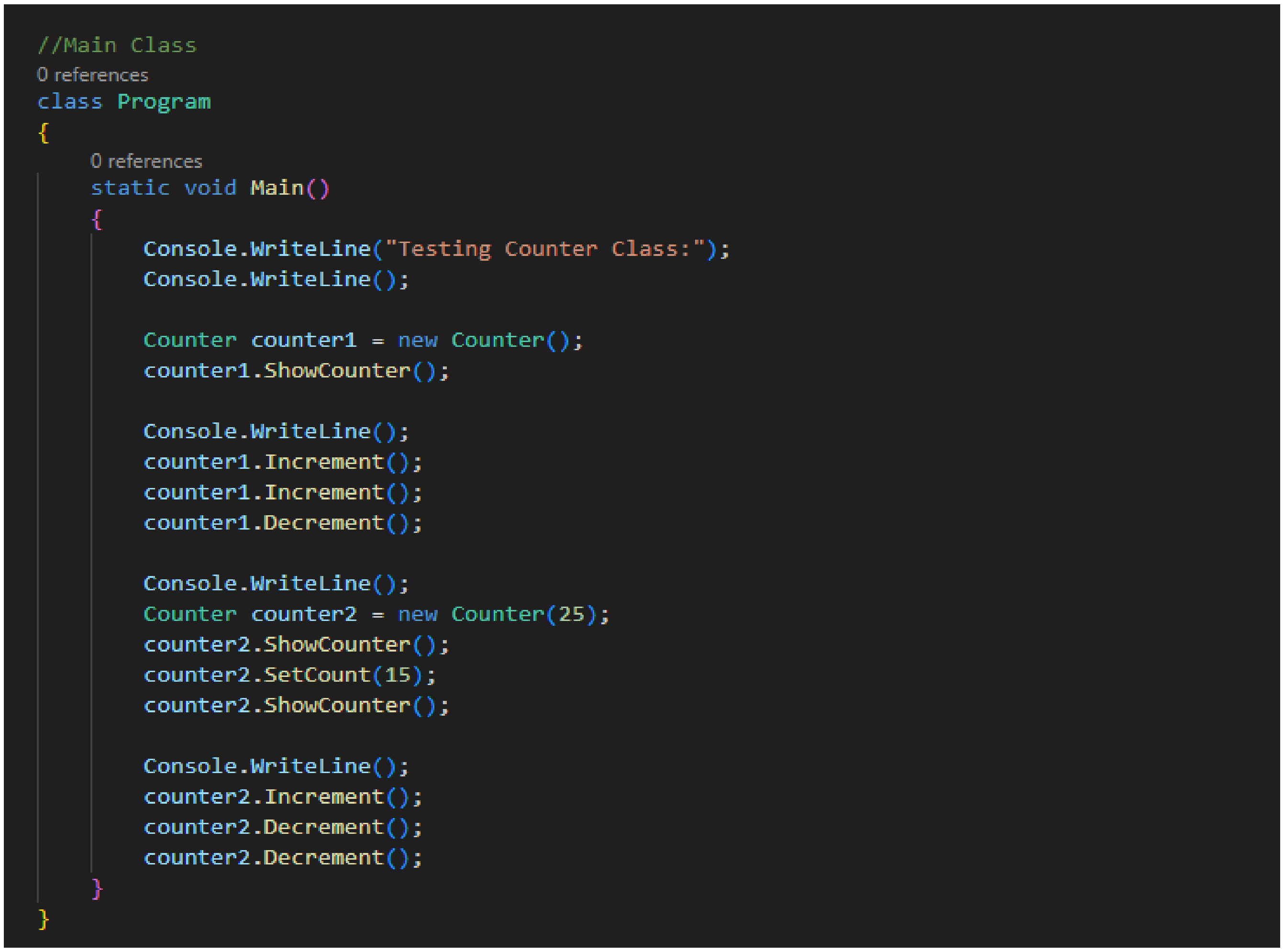
3. Crea una clase Contador con los métodos para incrementar y decrementar el contador. La clase contendrá un constructor por defecto, un constructor con parámetros, y los métodos getters y setters.



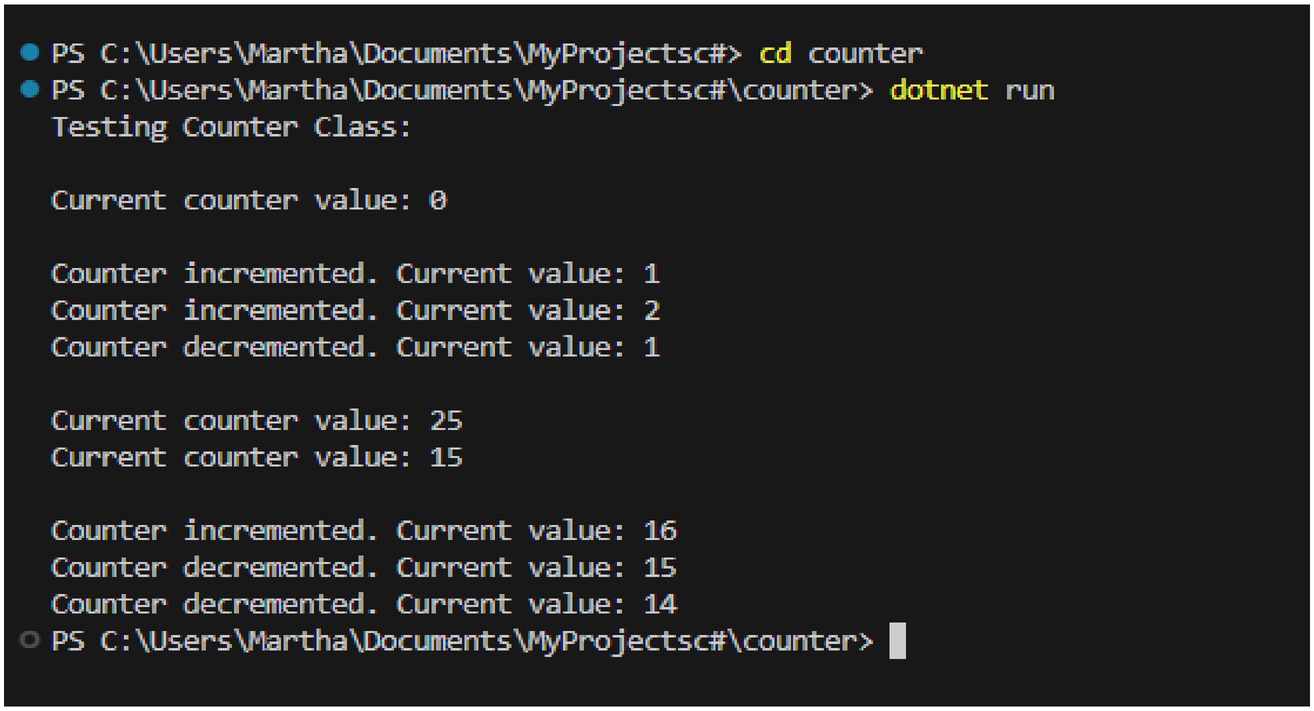
```
using System;

6 references
class Counter
{
    10 references
    private int count;

    //Default constructor
    1 reference
    public Counter()
    {
        count = 0;
    }

    //With parameters
    1 reference
    public Counter(int initialValue)
    {
        count = initialValue;
    }

    0 references
    public int GetCount()
    {
        return count;
    }

    1 reference
    public void SetCount(int value)
    {
        count = value;
    }
```



```
    //Increment method
    3 references
    public void Increment()
    {
        count++;
        Console.WriteLine($"Counter incremented. Current value: {count}");
    }

    //Decrement method
    3 references
    public void Decrement()
    {
        if (count > 0)
        {
            count--;
            Console.WriteLine($"Counter decremented. Current value: {count}");
        }
        else
        {
            Console.WriteLine("Counter can't go below zero.");
        }
    }
    3 references
    public void ShowCounter()
    {
        Console.WriteLine($"Current counter value: {count}");
    }
}
```

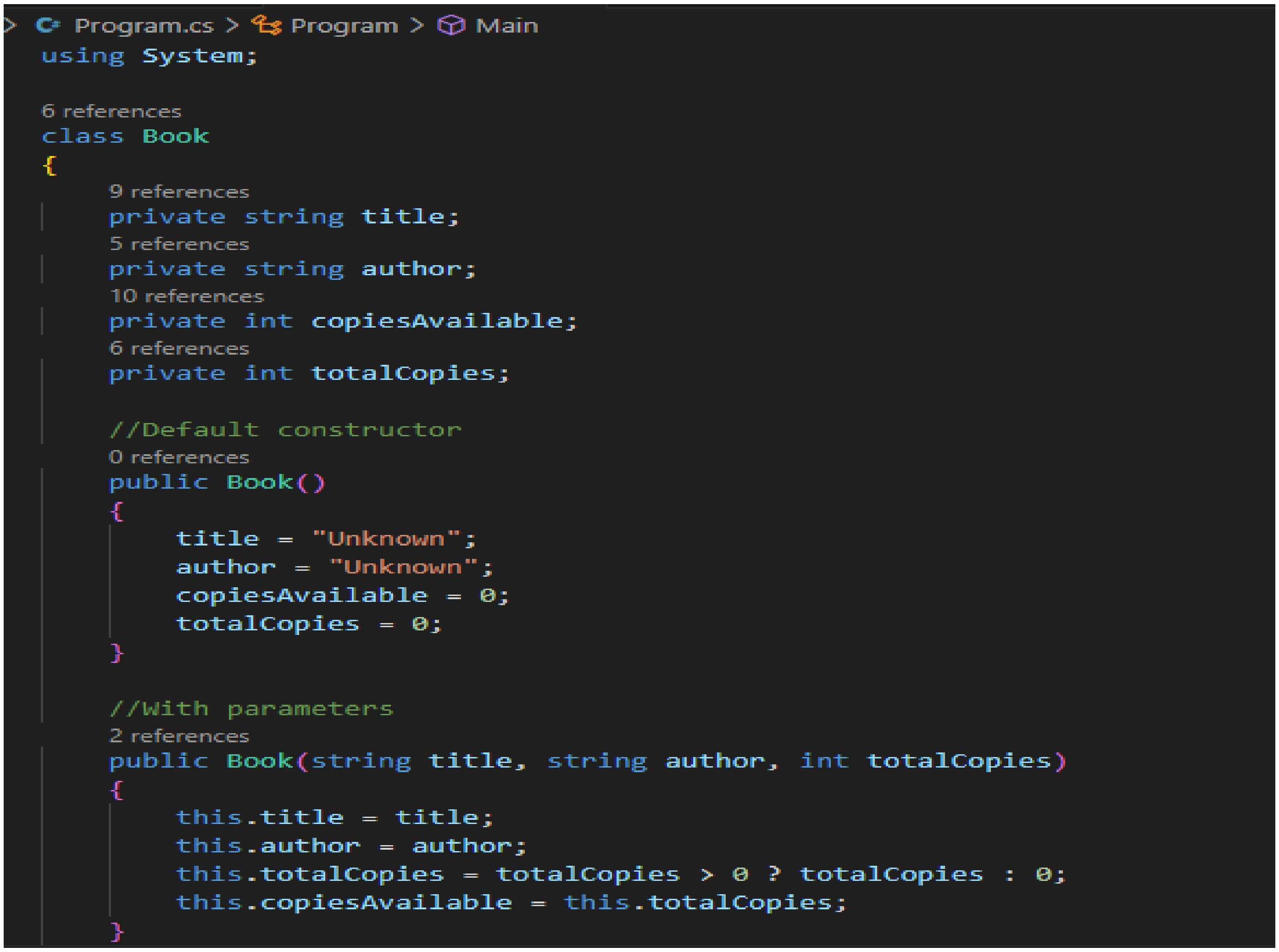


```
//Main Class
0 references
class Program
{
    0 references
    static void Main()
    {
        Console.WriteLine("Testing Counter Class:");
        Console.WriteLine();

        Counter counter1 = new Counter();
        counter1.ShowCounter();

        Console.WriteLine();
        counter1.Increment();
        counter1.Increment();
        counter1.Decrement();

        Console.WriteLine();
        Counter counter2 = new Counter(25);
        counter2.ShowCounter();
        counter2.SetCount(15);
        counter2.ShowCounter();

        Console.WriteLine();
        counter2.Increment();
        counter2.Decrement();
        counter2.Decrement();
    }
}
```

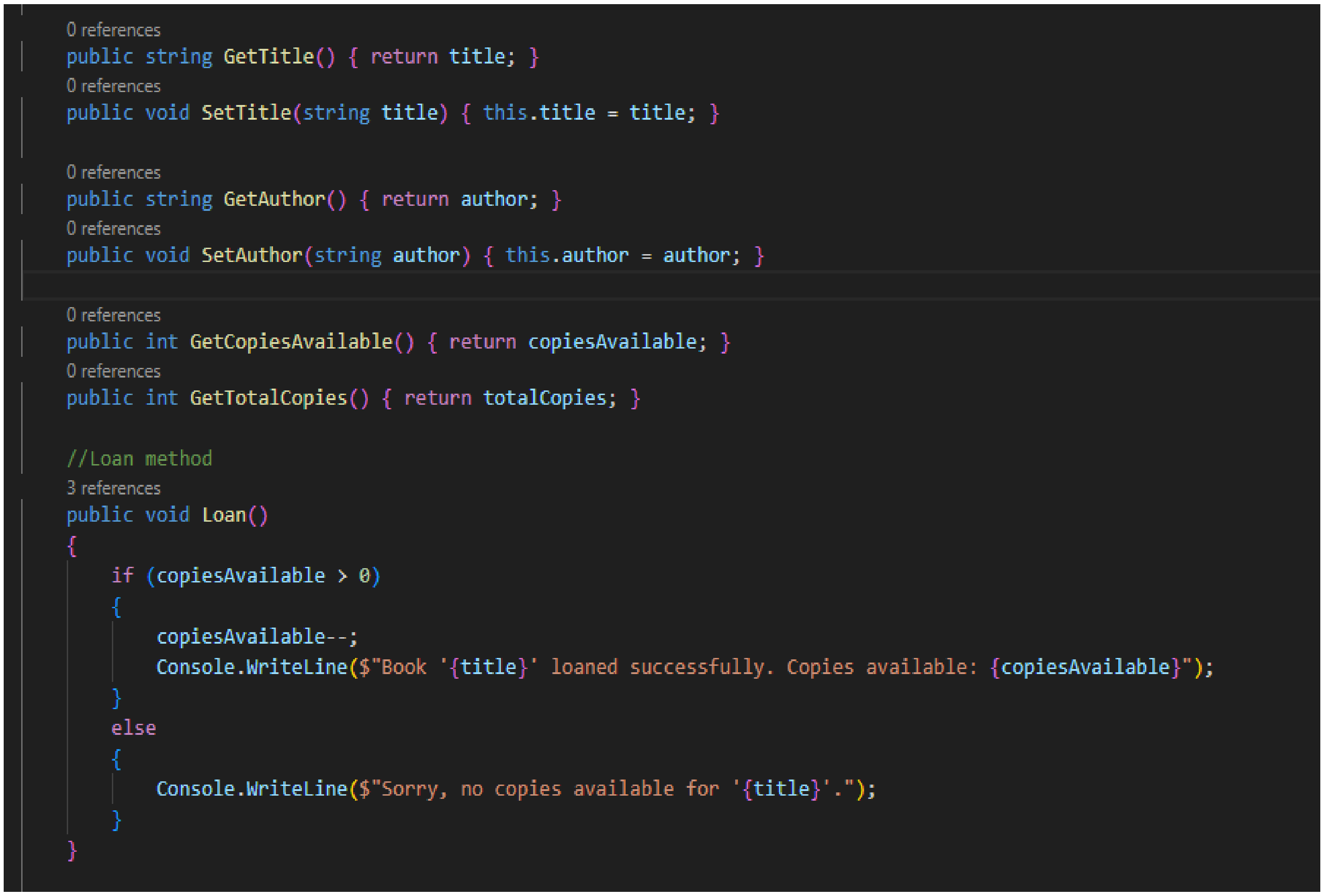


```
PS C:\Users\Martha\Documents\MyProjectsc#> cd counter
PS C:\Users\Martha\Documents\MyProjectsc#\counter> dotnet run
Testing Counter Class:

Current counter value: 0

Counter incremented. Current value: 1
Counter incremented. Current value: 2
Counter decremented. Current value: 1

Current counter value: 25
Current counter value: 15

Counter incremented. Current value: 16
Counter decremented. Current value: 15
Counter decremented. Current value: 14
PS C:\Users\Martha\Documents\MyProjectsc#\counter>
```

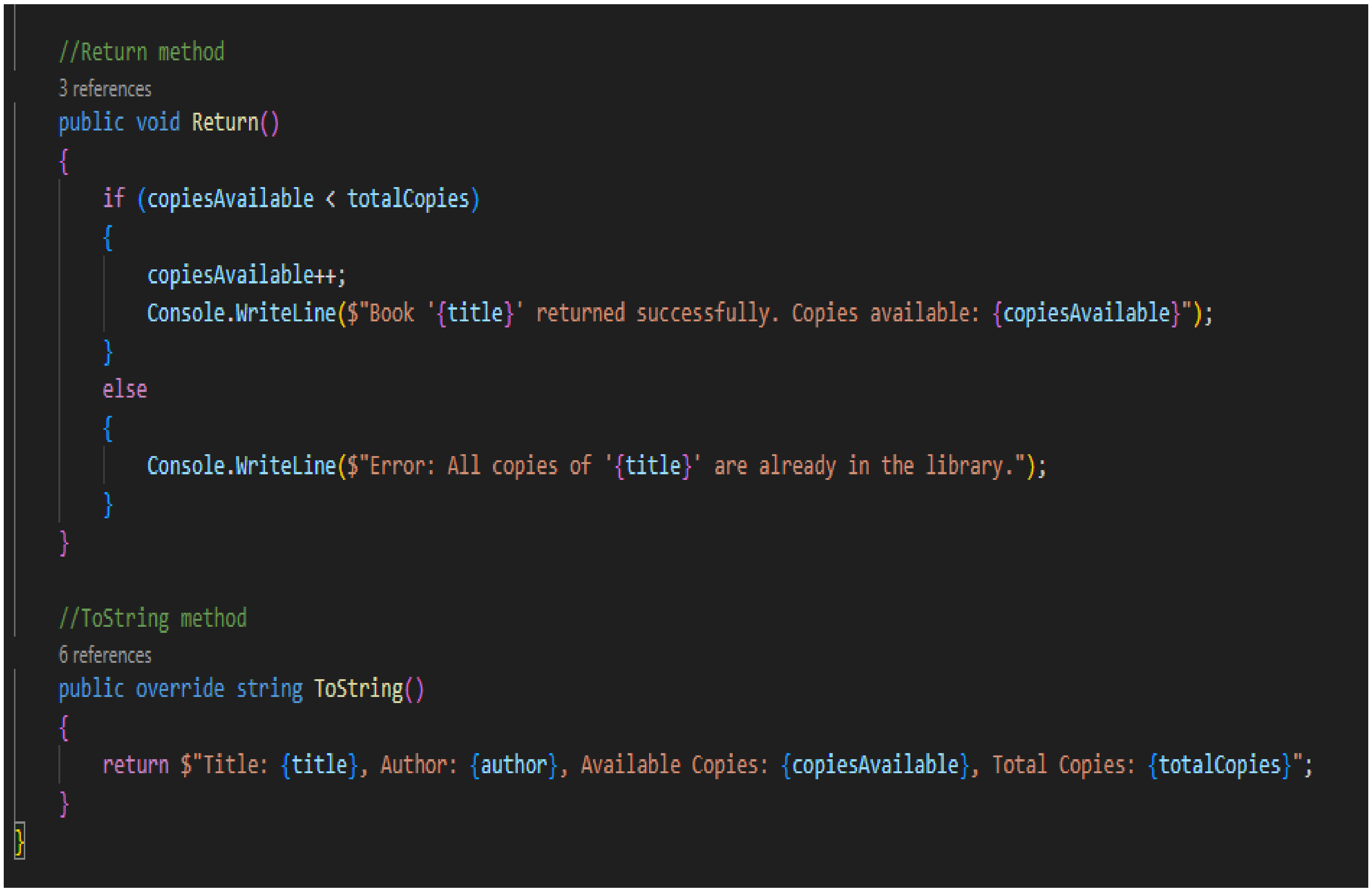
4. Crea una clase Libro con los métodos préstamo, devolución y ToString. La clase contendrá un constructor por defecto, un constructor con parámetros y los métodos getters y setters.



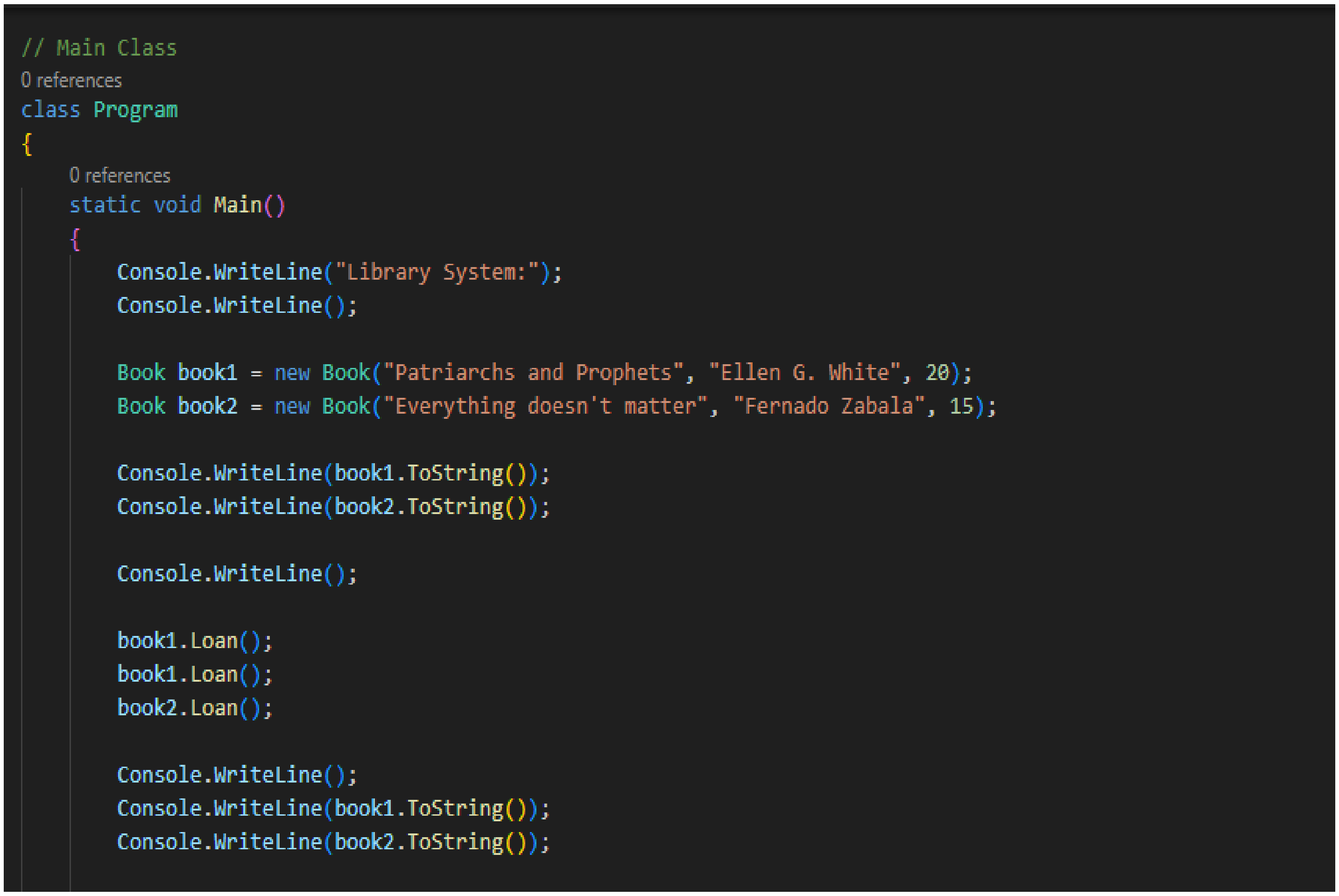
```
using System;

6 references
class Book
{
    9 references
    private string title;
    5 references
    private string author;
    10 references
    private int copiesAvailable;
    6 references
    private int totalCopies;

    //Default constructor
    0 references
    public Book()
    {
        title = "Unknown";
        author = "Unknown";
        copiesAvailable = 0;
        totalCopies = 0;
    }

    //With parameters
    2 references
    public Book(string title, string author, int totalCopies)
    {
        this.title = title;
        this.author = author;
        this.totalCopies = totalCopies > 0 ? totalCopies : 0;
        this.copiesAvailable = this.totalCopies;
    }
```



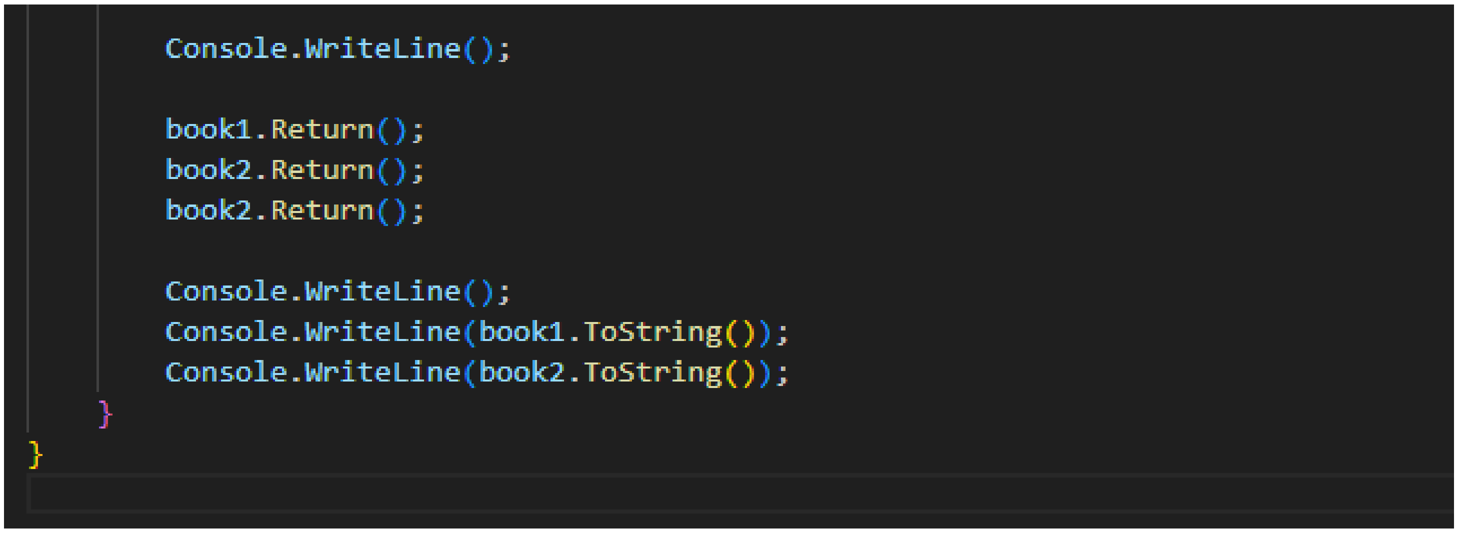
```
0 references
public string GetTitle() { return title; }
0 references
public void SetTitle(string title) { this.title = title; }

0 references
public string GetAuthor() { return author; }
0 references
public void SetAuthor(string author) { this.author = author; }

0 references
public int GetCopiesAvailable() { return copiesAvailable; }
0 references
public int GetTotalCopies() { return totalCopies; }

//Loan method
3 references
public void Loan()
{
    if (copiesAvailable > 0)
    {
        copiesAvailable--;
        Console.WriteLine($"Book '{title}' loaned successfully. Copies available: {copiesAvailable}");
    }
    else
    {
        Console.WriteLine($"Sorry, no copies available for '{title}'.");
    }
}
```



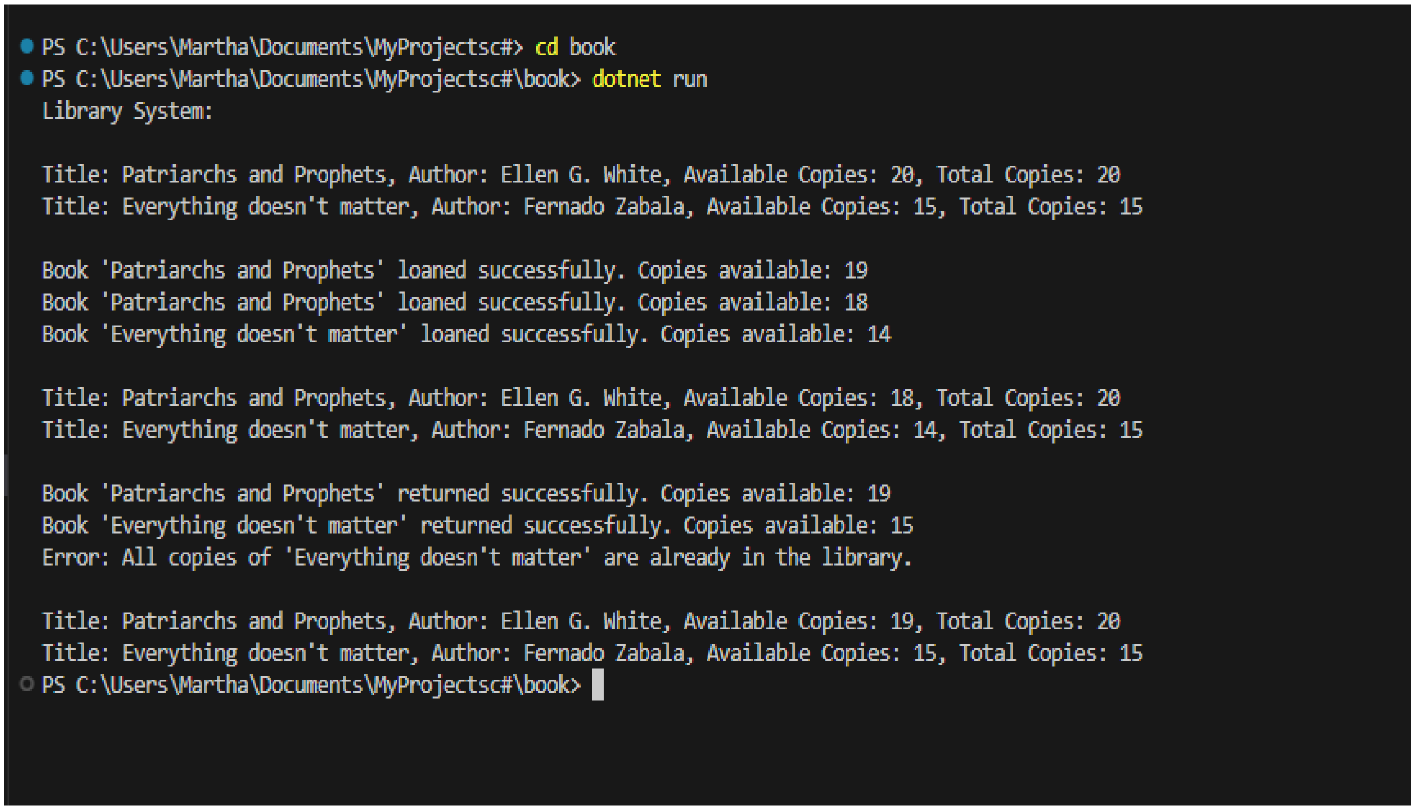
```
//Return method
3 references
public void Return()
{
    if (copiesAvailable < totalCopies)
    {
        copiesAvailable++;
        Console.WriteLine($"Book '{title}' returned successfully. Copies available: {copiesAvailable}");
    }
    else
    {
        Console.WriteLine($"Error: All copies of '{title}' are already in the library.");
    }
}

//ToString method
6 references
public override string ToString()
{
    return $"Title: {title}, Author: {author}, Available Copies: {copiesAvailable}, Total Copies: {totalCopies}";
}
}
```



```
// Main Class
0 references
class Program
{
    0 references
    static void Main()
    {
        Console.WriteLine("Library System:");
        Console.WriteLine();

        Book book1 = new Book("Patriarchs and Prophets", "Ellen G. White", 20);
        Book book2 = new Book("Everything doesn't matter", "Fernado Zabala", 15);

        Console.WriteLine(book1.ToString());
        Console.WriteLine(book2.ToString());

        Console.WriteLine();

        book1.Loan();
        book1.Loan();
        book2.Loan();

        Console.WriteLine();
        Console.WriteLine(book1.ToString());
        Console.WriteLine(book2.ToString());
```

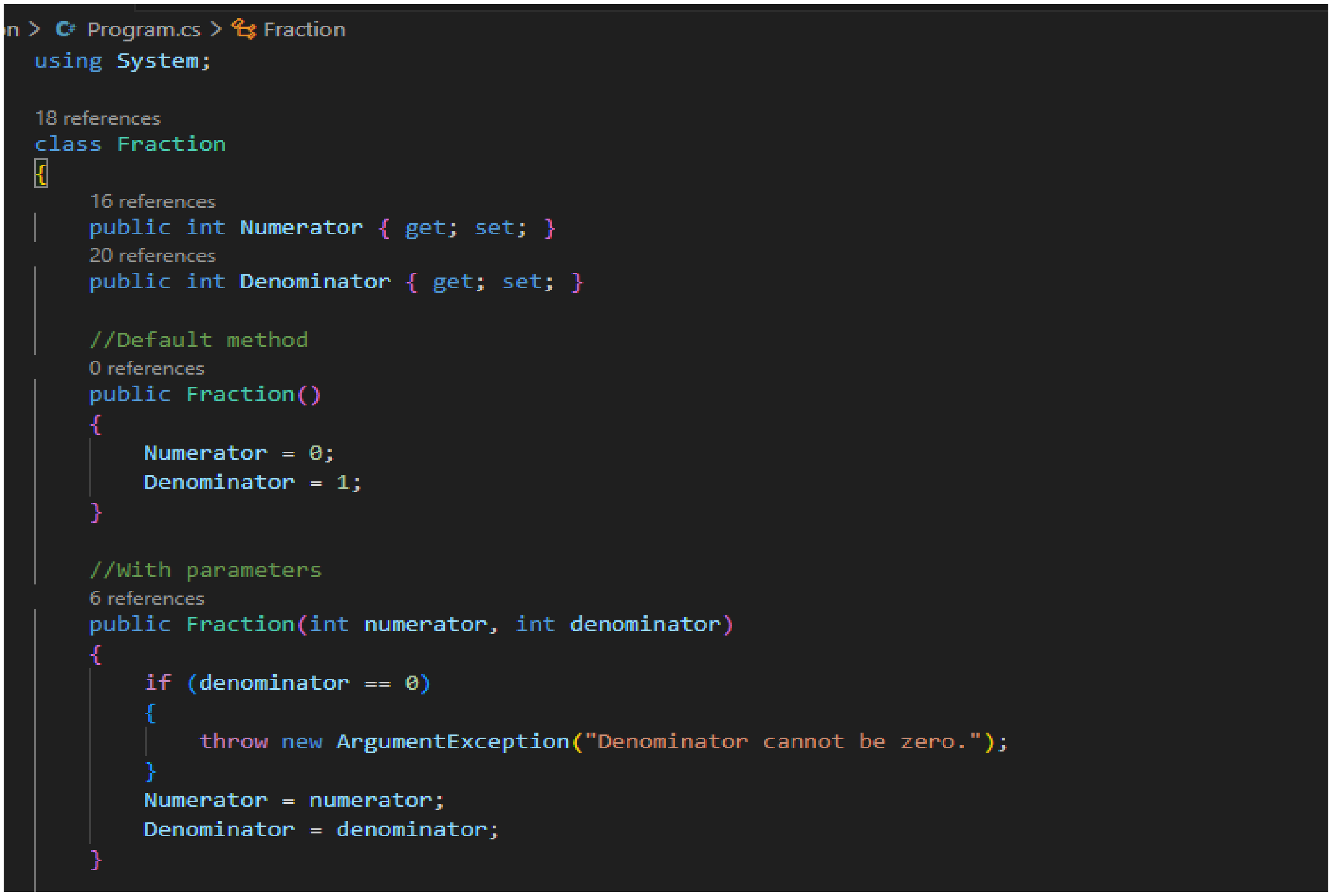


```
        Console.WriteLine();

        book1.Return();
        book2.Return();
        book2.Return();

        Console.WriteLine();
        Console.WriteLine(book1.ToString());
        Console.WriteLine(book2.ToString());
    }
}
```

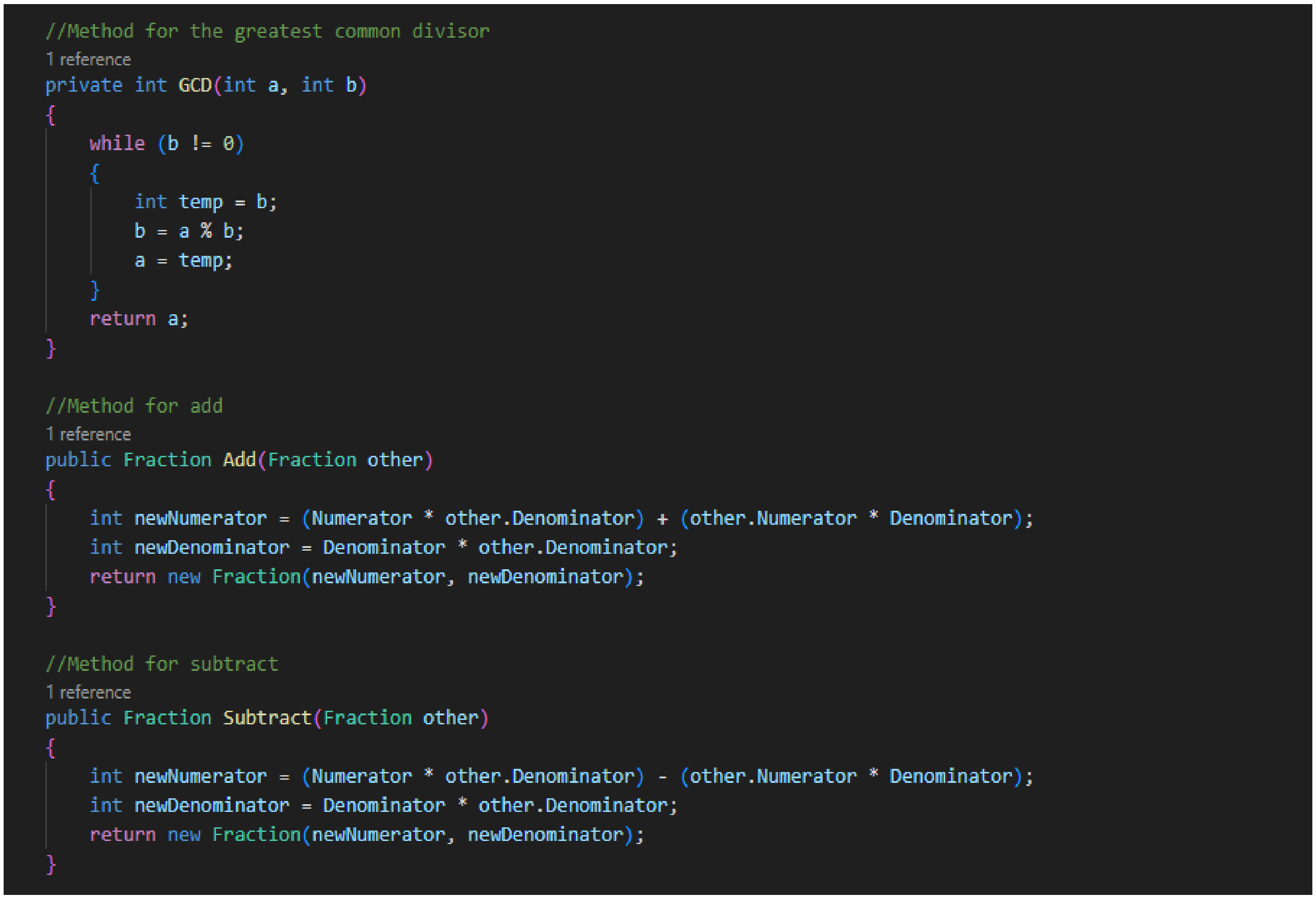


```
PS C:\Users\Martha\Documents\MyProjectsc#> cd book
PS C:\Users\Martha\Documents\MyProjectsc#\book> dotnet run
Library System:

Title: Patriarchs and Prophets, Author: Ellen G. White, Available Copies: 20, Total Copies: 20
Title: Everything doesn't matter, Author: Fernado Zabala, Available Copies: 15, Total Copies: 15

Book 'Patriarchs and Prophets' loaned successfully. Copies available: 19
Book 'Patriarchs and Prophets' loaned successfully. Copies available: 18
Book 'Everything doesn't matter' loaned successfully. Copies available: 14

Title: Patriarchs and Prophets, Author: Ellen G. White, Available Copies: 18, Total Copies: 20
Title: Everything doesn't matter, Author: Fernado Zabala, Available Copies: 14, Total Copies: 15

Book 'Patriarchs and Prophets' returned successfully. Copies available: 19
Book 'Everything doesn't matter' returned successfully. Copies available: 15
Error: All copies of 'Everything doesn't matter' are already in the library.

Title: Patriarchs and Prophets, Author: Ellen G. White, Available Copies: 19, Total Copies: 20
Title: Everything doesn't matter, Author: Fernado Zabala, Available Copies: 15, Total Copies: 15
PS C:\Users\Martha\Documents\MyProjectsc#\book>
```

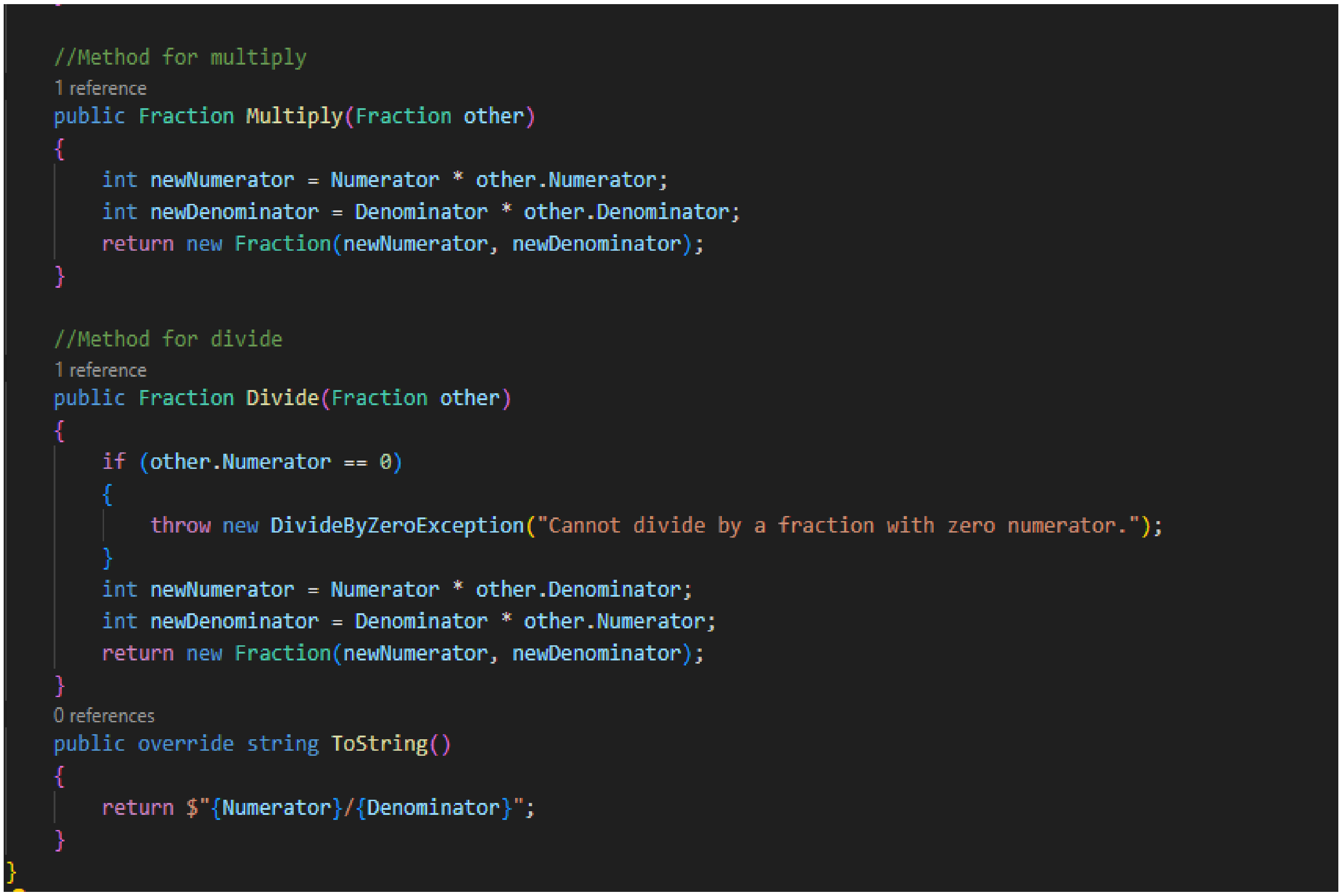
5. Crea una clase Fracción con métodos para sumar, restar, multiplicar y dividir fracciones.



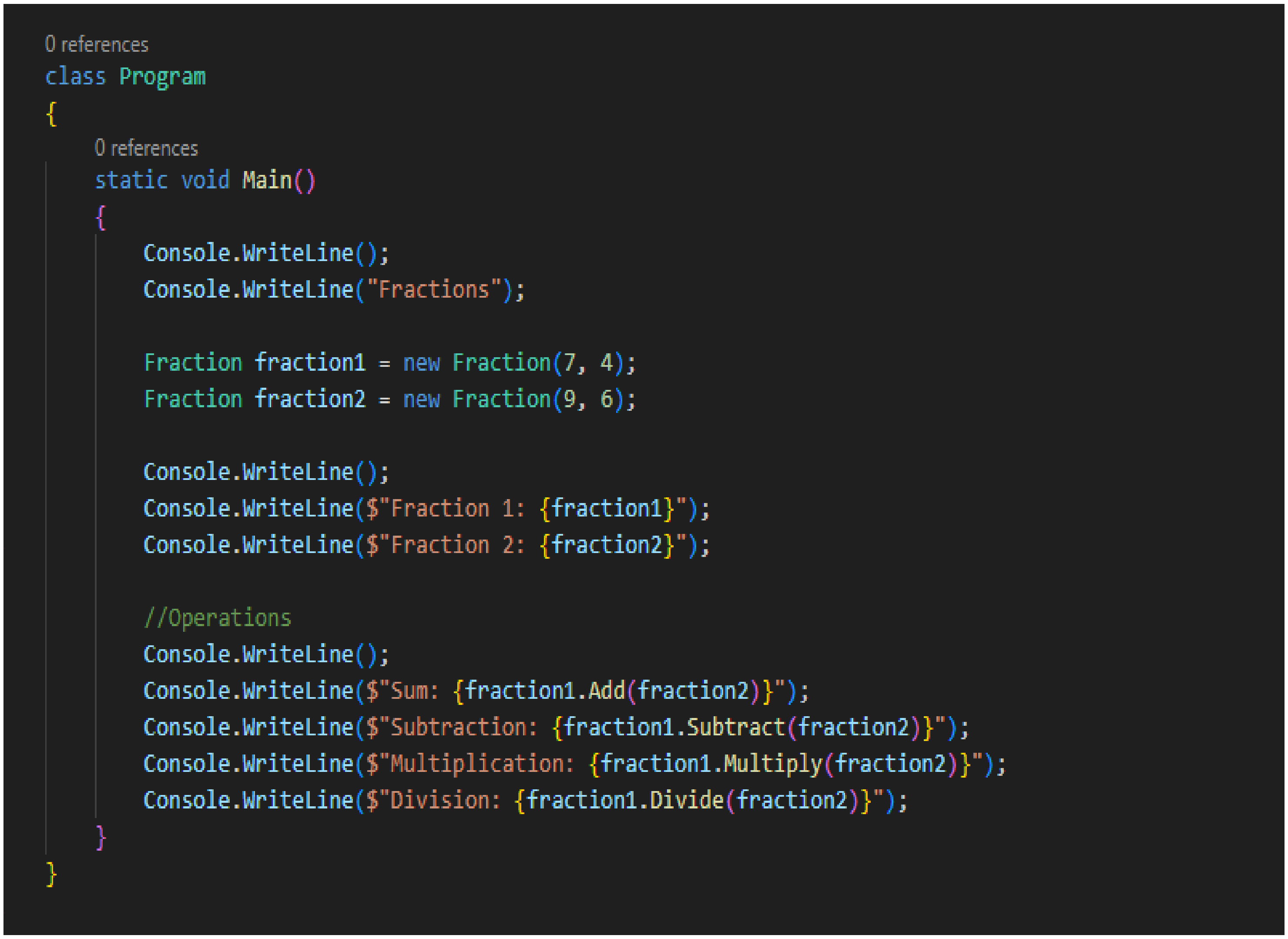
```
using System;

18 references
class Fraction
{
    16 references
    public int Numerator { get; set; }
    20 references
    public int Denominator { get; set; }

    //Default method
    0 references
    public Fraction()
    {
        Numerator = 0;
        Denominator = 1;
    }

    //With parameters
    6 references
    public Fraction(int numerator, int denominator)
    {
        if (denominator == 0)
        {
            throw new ArgumentException("Denominator cannot be zero.");
        }
        Numerator = numerator;
        Denominator = denominator;
    }
```



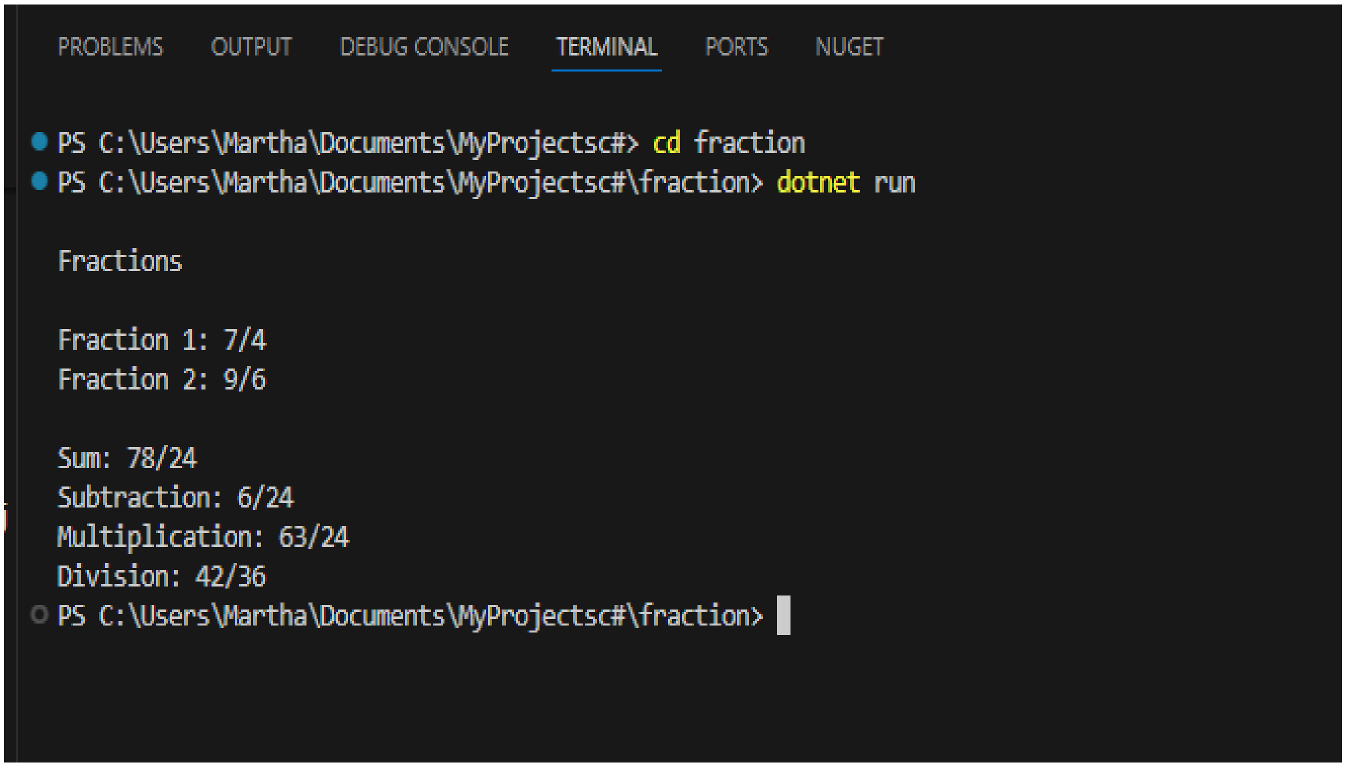
```
//Method for the greatest common divisor
1 reference
private int GCD(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

//Method for add
1 reference
public Fraction Add(Fraction other)
{
    int newNumerator = (Numerator * other.Denominator) + (other.Numerator * Denominator);
    int newDenominator = Denominator * other.Denominator;
    return new Fraction(newNumerator, newDenominator);
}

//Method for subtract
1 reference
public Fraction Subtract(Fraction other)
{
    int newNumerator = (Numerator * other.Denominator) - (other.Numerator * Denominator);
    int newDenominator = Denominator * other.Denominator;
    return new Fraction(newNumerator, newDenominator);
}
```



```
//Method for multiply
1 reference
public Fraction Multiply(Fraction other)
{
    int newNumerator = Numerator * other.Numerator;
    int newDenominator = Denominator * other.Denominator;
    return new Fraction(newNumerator, newDenominator);
}

//Method for divide
1 reference
public Fraction Divide(Fraction other)
{
    if (other.Numerator == 0)
    {
        throw new DivideByZeroException("Cannot divide by a fraction with zero numerator.");
    }
    int newNumerator = Numerator * other.Denominator;
    int newDenominator = Denominator * other.Numerator;
    return new Fraction(newNumerator, newDenominator);
}
0 references
public override string ToString()
{
    return $"{Numerator}/{Denominator}";
}
}
```



```
0 references
class Program
{
    0 references
    static void Main()
    {
        Console.WriteLine();
        Console.WriteLine("Fractions");

        Fraction fraction1 = new Fraction(7, 4);
        Fraction fraction2 = new Fraction(9, 6);

        Console.WriteLine();
        Console.WriteLine($"Fraction 1: {fraction1}");
        Console.WriteLine($"Fraction 2: {fraction2}");

        //Operations
        Console.WriteLine();
        Console.WriteLine($"Sum: {fraction1.Add(fraction2)}");
        Console.WriteLine($"Subtraction: {fraction1.Subtract(fraction2)}");
        Console.WriteLine($"Multiplication: {fraction1.Multiply(fraction2)}");
        Console.WriteLine($"Division: {fraction1.Divide(fraction2)}");
    }
}
```



```
● PS C:\Users\Martha\Documents\MyProjectsc#> cd fraction
● PS C:\Users\Martha\Documents\MyProjectsc#\fraction> dotnet run

  Fractions

  Fraction 1: 7/4
  Fraction 2: 9/6

  Sum: 78/24
  Subtraction: 6/24
  Multiplication: 63/24
  Division: 42/36
○ PS C:\Users\Martha\Documents\MyProjectsc#\fraction>
```