Examen Final Lenguajes de Programación

Ing. Vargas flores Jancarla

Temas: Programación Genérica y Manejo de Errores y Excepciones

Duración: 90 minutos

Lenguaje sugerido: Java o C# (puede adaptarse a C++ o Python según lo requerido)

Puntaje Total: 15 puntos

Instrucciones Generales:

- Resuelve todos los ejercicios propuestos.
- Comenta y captura claramente tu código.
- Ejecuta las pruebas necesarias para demostrar el funcionamiento y captura.
- Adjunta todo a un archivo y envíalo como respuesta.

Ejercicio 1: Clase Genérica de Contenedor

Implementa una clase genérica llamada `Contenedor<T>` que permita almacenar un único elemento de tipo `T` y tenga los siguientes métodos:

- `void setElemento(T elemento)`
- `T getElemento()`
- `String toString()` que muestre `"Contenedor: [elemento]"`

Requisitos:

- La clase debe ser completamente genérica.
- Debe funcionar con tipos primitivos y objetos (Integer, String, etc.).

```
using System;

namespace ExamenFinal
{
    public class Contenedor<T>
    {
        private T? elemento;
        public void SetElemento(T elemento)
        {
            this.elemento = elemento;
        }
```

```
        public T? GetElemento()
        {
```

```
            return elemento;
        }
        public override string ToString()
        {
            return $"Contenedor: [{elemento}]";
        }
    }
```

```
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("=== Ejercicio 1: Contenedor Genérico ===\n");
            // Contenedor de entero
            var contInt = new Contenedor<int>();
            Console.Write("Ingrese un número entero: ");
            if (int.TryParse(Console.ReadLine(), out int num))
            {
                contInt.SetElemento(num);
            }
            else
            {
                Console.WriteLine("⚠ Entrada inválida. Se almacenará 0 por
defecto.");
                contInt.SetElemento(0);
            }
```

```
            // Contenedor de texto
            var contString = new Contenedor<string>();
            Console.Write("Ingrese un texto: ");
            string? texto = Console.ReadLine();
            contString.SetElemento(texto ?? "");
```

```
            // Contenedor de fecha
            var contFecha = new Contenedor<DateTime>();
            Console.Write("Ingrese una fecha (ej: 2025-06-20): ");
            if (DateTime.TryParse(Console.ReadLine(), out DateTime fecha))
            {
                contFecha.SetElemento(fecha);
            }
            else
            {
                Console.WriteLine("⚠ Fecha inválida. Se usará la fecha actual.");
                contFecha.SetElemento(DateTime.Now);
            }
```

```
            // Mostrar resultados
            Console.WriteLine("\n=== Resultados ===");
            Console.WriteLine("Dato entero -> " + contInt);
            Console.WriteLine("Dato texto  -> " + contString);
            Console.WriteLine("Dato fecha  -> " + contFecha);
            Console.WriteLine("\nPresione ENTER para finalizar...");
            Console.ReadLine();
        }
    }
}
```

Ejercicio 2: División Segura

Escribe una función `dividir(int a, int b)` que devuelva el resultado de la división y maneje la división por cero usando excepciones.

Requisitos:

- Lanza una excepción personalizada si `b == 0`.
- Captura y muestra el mensaje de error sin detener el programa.

```
using System;

namespace ExamenFinal
{
    // Excepción personalizada para división por cero
    public class DivisionPorCeroException : Exception
    {
        public DivisionPorCeroException(string mensaje) : base(mensaje) { }
    }
```

```
    class Program
    {
        public static double Dividir(int a, int b)
        {
            if (b == 0)
                throw new DivisionPorCeroException("❌ Error: No se puede dividir
entre cero.");

            return (double)a / b;
        }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("=== Ejercicio 2: División Segura con Manejo de
Excepciones ===\n");
```

```
            Console.Write("Ingrese el dividendo (a): ");
            string? entradaA = Console.ReadLine();
```

```
            Console.Write("Ingrese el divisor (b): ");
            string? entradaB = Console.ReadLine();
```

```
            // Validación y ejecución
            try
            {
                int a = int.Parse(entradaA ?? "0");
                int b = int.Parse(entradaB ?? "0");
                double resultado = Dividir(a, b);
                Console.WriteLine($"\n Resultado: {a} / {b} = {resultado}");
            }
            catch (DivisionPorCeroException ex)
            {
                Console.WriteLine($"\n{ex.Message}");
            }
            catch (FormatException)
            {
```

```
                Console.WriteLine("\n⚠ Error: Debe ingresar números enteros
válidos.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"\nError inesperado: {ex.Message}");
            }
            Console.WriteLine("\nPrograma continúa normalmente...");
            Console.WriteLine("Presione ENTER para finalizar.");
            Console.ReadLine();
        }
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE                dotnet

  Presione ENTER para finalizar.

  PS C:\Users\Americo\Desktop\L programacion\ExamenFinal\Ejercicio2_DivisionSegura> dotnet run
=== Ejercicio 2: División Segura con Manejo de Excepciones ===

  Ingrese el dividendo (a): hola
  Ingrese el divisor (b): 2

  ⚠Error: Debe ingresar números enteros válidos.

  Programa continúa normalmente...
  Presione ENTER para finalizar.
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE          dotnet - Ejer

  Presione ENTER para finalizar.

  PS C:\Users\Americo\Desktop\L programacion\ExamenFinal\Ejercicio2_DivisionSegura> dotnet run
=== Ejercicio 2: División Segura con Manejo de Excepciones ===

  Ingrese el dividendo (a): 10
  Ingrese el divisor (b): 0

   Error: No se puede dividir entre cero.

  Programa continúa normalmente...
  Presione ENTER para finalizar.
```

Ejercicio 3: Lectura de Archivo con Manejo de Excepciones

Simula la lectura de un archivo (puedes usar una lista de Strings como contenido). Lanza una excepción si el archivo no existe o si una línea está vacía.

Requisitos:

- Crea una clase personalizada de excepción llamada `ArchivoInvalidoException`.
- Usa `try-catch-finally` para asegurar el cierre del archivo (simulado).
- Muestra los errores con mensajes amigables.

```
using System;
using System.Collections.Generic;

namespace ExamenFinal
{
    // Excepción personalizada para errores en archivo
    public class ArchivoInvalidoException : Exception
    {
        public ArchivoInvalidoException(string mensaje) : base(mensaje) { }
    }
```

```
    class Program
    {
        public static void LeerArchivo(List<string> lineas)
        {
            Console.WriteLine("\n=== Iniciando lectura del archivo simulado...
===");
```

```
            try
            {
                if (lineas == null || lineas.Count == 0)
                    throw new ArchivoInvalidoException(" Error: El archivo está
vacío o no existe.");
```

```
                foreach (var linea in lineas)
                {
                    if (string.IsNullOrWhiteSpace(linea))
                        throw new ArchivoInvalidoException(" Error: Se encontró
una línea vacía.");
```

```
                    Console.WriteLine(" Línea leída: " + linea);
                }
```

```
            }
            catch (ArchivoInvalidoException ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Console.WriteLine("  Archivo cerrado (simulado con finally).");
            }
        }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("=== Ejercicio 3: Lectura de archivo con ingreso
manual ===\n");
```

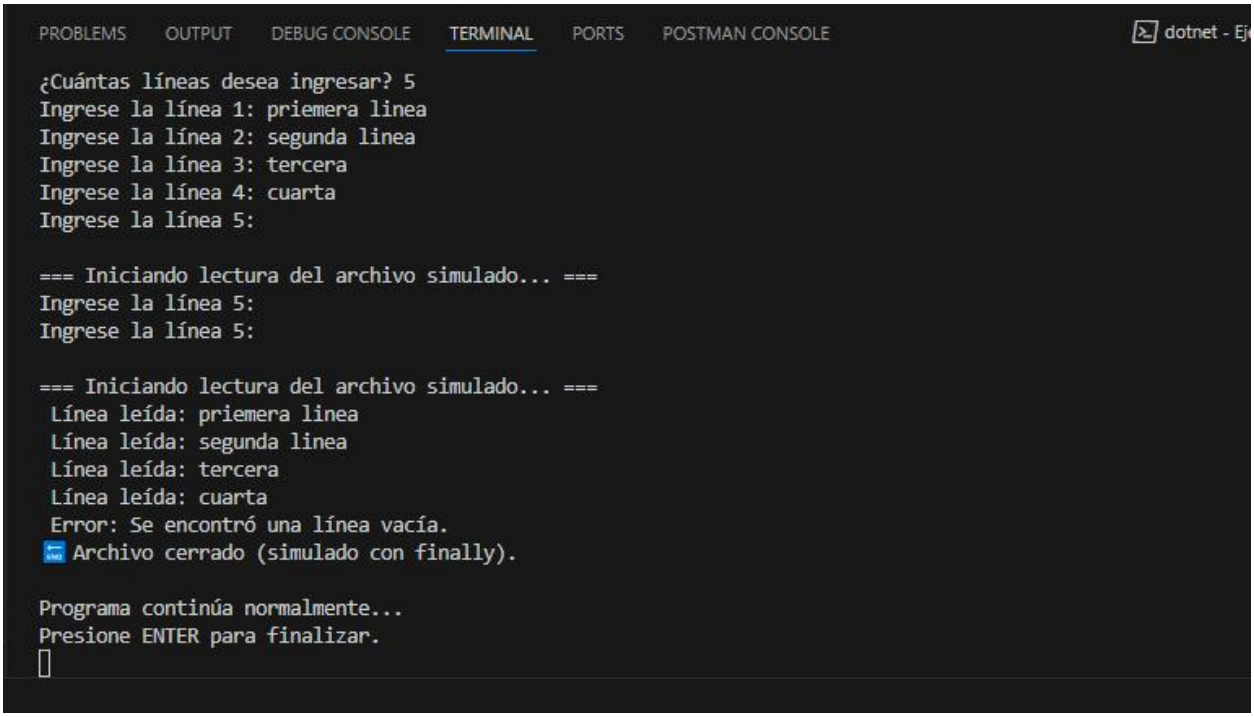
```
            List<string> contenidoArchivo = new List<string>();
```

```
            Console.Write("¿Cuántas líneas desea ingresar? ");
            if (!int.TryParse(Console.ReadLine(), out int cantidad) || cantidad
<= 0)
            {
                Console.WriteLine(" Entrada inválida. Se canceló la operación.");
                return;
            }
```

```
            for (int i = 1; i <= cantidad; i++)
            {
                Console.Write($"Ingrese la línea {i}: ");
                string? linea = Console.ReadLine();
                contenidoArchivo.Add(linea ?? "");
            }
```

```
            LeerArchivo(contenidoArchivo);
```

```
            Console.WriteLine("\nPrograma continúa normalmente...");
            Console.WriteLine("Presione ENTER para finalizar.");
            Console.ReadLine();
        }
    }
}
```



Ejercicio 4: Validación de Datos de Usuario

Crea una función que valide la edad y el nombre de un usuario.

- Si el nombre está vacío, lanza `IllegalArgumentException`.
- Si la edad es negativa, lanza `NumberFormatException`.

Requisitos:

- Captura las excepciones y muestra mensajes adecuados.
- Usa una clase `Usuario` con `nombre` y `edad` y una función `validarUsuario(Usuario u)`.

```
using System;

namespace ExamenFinal
{
    public class Usuario
    {
        public string Nombre { get; set; }
        public int Edad { get; set; }
```

```
        public Usuario(string nombre, int edad)
```

```
        {
            Nombre = nombre;
            Edad = edad;
        }
    }
```

```
    class Program
    {
        public static void ValidarUsuario(Usuario u)
        {
            if (string.IsNullOrWhiteSpace(u.Nombre))
                throw new ArgumentException(" El nombre no puede estar vacío.");
```

```
            if (u.Edad < 0)
                throw new ArgumentOutOfRangeException(" La edad no puede ser
negativa.");
```

```
            Console.WriteLine($"\n Usuario válido: {u.Nombre}, edad {u.Edad}");
        }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("=== Ejercicio 4: Validación de Datos de Usuario
===\n");
```

```
           try
{
    Console.Write("Ingrese el nombre del usuario: ");
    string? nombre = Console.ReadLine();
```

```
    Console.Write("Ingrese la edad del usuario: ");
    string? edadStr = Console.ReadLine();
```

```
    if (!int.TryParse(edadStr, out int edad))
        throw new FormatException(" Debe ingresar un número válido para la
edad.");
```

```
    Usuario u = new Usuario(nombre ?? "", edad);
    ValidarUsuario(u);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}
catch (ArgumentException ex)
{
    Console.WriteLine(ex.Message);
```

```
}
catch (FormatException ex)
{
    Console.WriteLine(ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine(" Error inesperado: " + ex.Message);
}
```

```
            Console.WriteLine("\nPrograma continúa normalmente...");
            Console.WriteLine("Presione ENTER para finalizar.");
            Console.ReadLine();
        }
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE        dotnet - Ejercici

PS C:\Users\Americo\Desktop\L programacion\ExamenFinal\Ejercicio4_ValidacionUsuario> dotnet run
=== Ejercicio 4: Validación de Datos de Usuario ===

Ingrese el nombre del usuario: meco
Ingrese la edad del usuario: 28

 Usuario válido: meco, edad 28

Programa continúa normalmente...
Presione ENTER para finalizar.
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE        dotnet - Ejercici

Presione ENTER para finalizar.
PS C:\Users\Americo\Desktop\L programacion\ExamenFinal\Ejercicio4_ValidacionUsuario> dotnet run
=== Ejercicio 4: Validación de Datos de Usuario ===

Ingrese el nombre del usuario: 123123
Ingrese la edad del usuario: qweqwe
 Debe ingresar un número válido para la edad.

Programa continúa normalmente...
Presione ENTER para finalizar.
```