

UNIVERSIDAD TECNICA DE ORURO
FACULTAD NACIONAL DE INGENIERIA
INGENIERIA DE SISTEMAS E INFORMATICA
PRACTICA N° 1

MATERIA: Actualización Tecnológica SIS2420 "A"

DOCENTE: Ing. Saul Mamani Mamani

AUXILIAR: Egr. William Mucio Achabal Villalpando

NOMBRE: Univ. Tellez Quenaya Saul

PARTE TEORICA:

1. ¿Qué es un sistema?

Un sistema es un conjunto de elementos interconectados o relacionados que trabajan juntos para cumplir un objetivo o una función específica. Los sistemas pueden encontrarse en una amplia variedad de contextos, desde la biología y la informática hasta la ingeniería, la economía y muchas otras áreas. En un sistema, los componentes individuales, conocidos como elementos o partes, interactúan entre sí para lograr un propósito común. Estas interacciones pueden ser físicas, como en una máquina que consta de engranajes y motores que se mueven de cierta manera para realizar una tarea, o pueden ser abstractas, como en un sistema de gestión empresarial que utiliza software para coordinar diferentes aspectos de una organización. Los sistemas pueden ser simples o complejos, dependiendo de la cantidad de elementos involucrados y la naturaleza de sus interacciones.

2. ¿Qué es y qué diferencias tienen una clase abstracta y una clase estática en C#?

Clase Abstracta:

Una clase abstracta es una clase que no se puede instanciar directamente. Se utiliza como una plantilla o base para otras clases que heredarán de ella. Las principales características de las clases abstractas son:

- No se pueden crear objetos directamente a partir de una clase abstracta.
- Pueden contener métodos abstractos, que son métodos que no tienen una implementación definida en la clase abstracta y deben ser implementados por las clases derivadas.

- Pueden contener métodos concretos (métodos con una implementación definida) y propiedades, que las clases derivadas pueden heredar y usar tal como están o modificar según sea necesario.

Clase Estática:

Una clase estática es una clase que no se puede instanciar en absoluto. Todos sus miembros (métodos, propiedades, campos, etc.) deben ser estáticos. Las principales características de las clases estáticas son:

- No se pueden crear instancias de la clase.
- Todos los miembros de la clase deben ser estáticos y se acceden a través del nombre de la clase, no de una instancia.
- Se utilizan comúnmente para agrupar funcionalidades relacionadas o utilitarias que no requieren estado de instancia.

3. ¿Qué es y qué diferencias tienen la herencia y polimorfismo en C#?

La herencia:

Es un concepto que permite a una clase (llamada clase derivada o subclase) heredar las características (campos, propiedades y métodos) de otra clase (llamada clase base o superclase). La clase derivada puede extender o especializar la funcionalidad de la clase base y, al mismo tiempo, hereda sus atributos y métodos. Algunas características clave de la herencia en C# son:

- **Reutilización de código:** La herencia permite reutilizar el código de la clase base en la clase derivada, evitando la duplicación de código.
- **Extensión de funcionalidad:** La clase derivada puede agregar nuevos campos, propiedades y métodos, o modificar el comportamiento heredado de la clase base.
- **Jerarquía de clases:** Se puede crear una jerarquía de clases donde una clase base puede tener múltiples clases derivadas.

El polimorfismo:

Es otro concepto fundamental en la programación orientada a objetos que se refiere a la capacidad de objetos de diferentes clases de responder a la misma llamada de método de una manera específica para cada objeto. El polimorfismo se logra mediante la herencia y la

implementación de métodos virtuales o interfaces. Algunas características clave del polimorfismo en C# son:

Métodos virtuales y anulación: En C#, puedes declarar métodos como "virtuales" en una clase base y luego anular (override) esos métodos en las clases derivadas para proporcionar implementaciones específicas de esas clases.

Uso de interfaces: También puedes lograr el polimorfismo mediante interfaces, donde varias clases pueden implementar una interfaz común y responder de manera diferente a los métodos definidos en esa interfaz.

En conclusión, la herencia se utiliza para establecer una relación de jerarquía entre clases, mientras que el polimorfismo permite que objetos de diferentes clases respondan de manera diferente a las mismas llamadas de método. Ambos conceptos son fundamentales en la programación orientada a objetos y se utilizan en C# para crear código más flexible y extensible.

4. ¿Qué es un ciclo de vida del desarrollo de software (SDLC)?

El Ciclo de Vida del Desarrollo de Software (SDLC, por sus siglas en inglés, Software Development Life Cycle) es un proceso estructurado que define las etapas y actividades necesarias para planificar, diseñar, crear, probar y mantener un sistema de software. El objetivo principal del SDLC es proporcionar un marco metodológico para el desarrollo de software que garantice la calidad del producto final, controle los costos y los plazos, y asegure que el software cumpla con los requisitos y expectativas del cliente.

A lo largo de las etapas del SDLC, los equipos de desarrollo trabajan en conjunto para llevar a cabo tareas específicas y garantizar que el software se construya de manera efectiva y eficiente.

5. Para qué sirven estos comandos de Git:

Git init: Este comando se utiliza para iniciar un nuevo repositorio de Git en un directorio. Crea una carpeta oculta llamada git que almacena toda la información sobre el control de versiones.

Git status: Muestra el estado actual del repositorio, incluyendo los archivos modificados, los archivos que están listos para ser confirmados y los archivos que no están siendo rastreados por Git.

Git add .: Agrega cambios o archivos al área de preparación (staging area) para que puedan ser incluidos en el próximo commit.

Git commit -m "Mensaje": Confirma los cambios en el repositorio con un mensaje descriptivo que indica qué cambios se realizaron en el commit.

Git log: Muestra un registro de todos los commits realizados en el repositorio, incluyendo detalles como el autor, la fecha y el mensaje de cada commit.

Git checkout: Cambia entre ramas existentes en el repositorio. Puedes utilizarlo para cambiar al historial de una rama específica.

Git checkout -b NombreRama: Crea una nueva rama a partir de la rama actual y cambia a esa nueva rama.

Git branch: Muestra una lista de todas las ramas en el repositorio. La rama actual se marca con un asterisco.

Git push: Sube los cambios confirmados a un repositorio remoto, generalmente en una plataforma de alojamiento como GitHub o GitLab.

Git pull: Obtiene y fusiona (merge) los cambios desde un repositorio remoto en la rama actual.

Git merge: Combina dos ramas en una sola, generalmente se utiliza para fusionar una rama secundaria con la rama principal.

Git clone: Crea una copia completa de un repositorio remoto en tu máquina local. Es utilizado para descargar un proyecto de Git existente.

6. ¿Cuál es la diferencia entre una metodología tradicional y ágil?

La principal diferencia entre una metodología tradicional y ágil en el desarrollo de software es que las metodologías tradicionales siguen un enfoque secuencial y rígido, mientras que las metodologías ágiles adoptan un enfoque iterativo, adaptable y orientado a entregas frecuentes de valor al cliente. Las metodologías ágiles son más flexibles y enfocadas en la colaboración y la comunicación continua, mientras que las tradicionales tienden a seguir una planificación detallada y entregas al final del proyecto.

7. Dar 5 ejemplos de una metodología tradicional y 5 ejemplos de una metodología tradicional ágil

Metodologías Tradicionales (En Cascada o Predictivas):

1. **Modelo en Cascada (Waterfall):** Este modelo sigue una secuencia lineal de fases, donde cada fase (requisitos, diseño, implementación, pruebas, mantenimiento) debe completarse antes de pasar a la siguiente.
2. **Modelo en V:** Similar al modelo en cascada, pero enfatiza la relación entre las fases de desarrollo y las fases de pruebas correspondientes.
3. **Modelo de Desarrollo de Software en Fases (SDLC):** Este es un enfoque genérico que abarca varias metodologías tradicionales, como el modelo en cascada o el modelo en espiral.
4. **Modelo en Big Bang:** En este enfoque, se comienza a desarrollar el software sin una planificación detallada y se ajusta a medida que avanza.
5. **Modelo de Desarrollo Basado en Componentes (CBD):** Se centra en la reutilización de componentes de software y se planifica de manera más detallada antes de la implementación.

Metodologías Ágiles:

1. **Scrum:** Scrum es un enfoque ágil que divide el proyecto en iteraciones llamadas "sprints" y se enfoca en la colaboración, la adaptación continua y la entrega de funcionalidad valiosa en cada sprint.
2. **Kanban:** Kanban es una metodología ágil que utiliza tableros visuales para gestionar el flujo de trabajo y priorizar las tareas de manera incremental.
3. **Extreme Programming (XP):** XP es un enfoque ágil que se centra en la calidad del software, las pruebas continuas y la colaboración estrecha entre el equipo de desarrollo y el cliente.
4. **Lean Software Development:** Lean se basa en los principios de la producción lean y se enfoca en la eliminación de desperdicios, la eficiencia y la entrega de valor al cliente.
5. **Dynamic Systems Development Method (DSDM):** DSDM es una metodología ágil que se centra en la entrega temprana y continua de software funcional y prioriza la colaboración y la comunicación con el cliente.

8. ¿Qué es un Requerimiento Funcional y No Funcional?

Requisitos Funcionales:

Los requisitos funcionales se centran en las funciones, las características y el comportamiento del sistema. Estos requisitos describen qué debe hacer el software y cómo debe comportarse en situaciones específicas. Los requisitos funcionales suelen responder a las preguntas "¿Qué debe hacer el sistema?" y "¿Cómo debe hacerlo?" Algunos ejemplos de requisitos funcionales incluyen:

- La capacidad de un sistema de registro de usuarios para crear nuevas cuentas.
- La función de búsqueda que permite a los usuarios buscar información en una base de datos.
- La capacidad de un sistema de ventas en línea para calcular el total de una compra y procesar pagos.
- La función de envío de correos electrónicos de confirmación después de realizar una reserva en un sitio web de reservas de viajes.

Requisitos No Funcionales:

Los requisitos no funcionales, por otro lado, se refieren a las cualidades o atributos del sistema que no están directamente relacionados con las funciones específicas que realiza, sino más bien con cómo debe ser el sistema en términos de rendimiento, seguridad, usabilidad y otros aspectos. Estos requisitos suelen responder a preguntas como "¿Cómo debe funcionar el sistema en términos de rendimiento y seguridad?" Algunos ejemplos de requisitos no funcionales incluyen:

- El tiempo de respuesta máximo permitido para una página web.
- La capacidad de manejar un número específico de usuarios concurrentes.
- Los niveles de seguridad y cifrado de datos necesarios para proteger la información del usuario.
- Los estándares de accesibilidad que deben cumplirse para garantizar que el software sea utilizable por personas con discapacidades.

9. ¿Qué es SCRUM?

Scrum es un marco de trabajo ágil ampliamente utilizado en el desarrollo de software y en la gestión de proyectos. Fue creado inicialmente como un enfoque para gestionar proyectos de desarrollo de software, pero desde entonces se ha aplicado con éxito en una variedad de

contextos fuera del desarrollo de software. Scrum se basa en principios de agilidad y se centra en la colaboración, la adaptación continua y la entrega de valor en forma iterativa e incremental.

El objetivo principal de Scrum es permitir la entrega temprana y continua de software de alta calidad que cumple con las necesidades cambiantes del cliente. Es ampliamente utilizado en el desarrollo de software debido a su flexibilidad y capacidad para gestionar proyectos en entornos donde los requisitos pueden cambiar con frecuencia. Scrum se considera una de las metodologías ágiles más populares y efectivas en la actualidad.

10. ¿Cuáles son los roles de SCRUM?

Product Owner (Dueño del Producto):

- El Product Owner es responsable de representar las necesidades y objetivos del cliente y/o usuarios finales.
- Prioriza y gestiona el Backlog de Producto, que es una lista de todas las funcionalidades, características y tareas pendientes del proyecto.
- Define los criterios de aceptación para las historias de usuario y trabaja en estrecha colaboración con el Equipo de Desarrollo para garantizar que se entiendan y se implementen correctamente.
- Toma decisiones sobre qué funcionalidades deben desarrollarse y cuándo se entregarán al cliente.

Scrum Master (Maestro Scrum):

- El Scrum Master actúa como un facilitador y un entrenador para el equipo Scrum.
- Ayuda al equipo a comprender y adoptar los principios y prácticas de Scrum.
- Elimina obstáculos y bloqueos que puedan afectar el progreso del equipo.
- Facilita las reuniones Scrum, como la Reunión Diaria de Scrum, la Planificación del Sprint, la Revisión del Sprint y la Retrospectiva del Sprint.
- Promueve la mejora continua y la resolución de problemas dentro del equipo.

Developers (Equipo de Desarrollo):

- El Equipo de Desarrollo es un grupo de profesionales multifuncionales que trabajan juntos para desarrollar el producto.

- Son responsables de la planificación, el diseño, la implementación, las pruebas y la entrega de las funcionalidades en cada Sprint.
- Trabajan de manera autoorganizada y se esfuerzan por alcanzar los objetivos del Sprint.
- Colaboran estrechamente con el Product Owner para comprender los requisitos y con el Scrum Master para superar obstáculos.
- Deben ser capaces de tomar decisiones técnicas y de diseño de forma autónoma.

PARTE PRACTICA:

1. Realizar un programa utilizando una clase estática que permita ingresar un número por teclado y te muestre en su parte literal.

Entrada

Numero: 55

Salida

Cincuenta y cinco

PROGRAMA:

```
using System;

public static class Numero
{
    private static string[] unidades = {
        "", "uno", "dos", "tres", "cuatro", "cinco", "seis", "siete",
        "ocho", "nueve"
    };

    private static string[] decenas = {
        "", "diez", "veinte", "treinta", "cuarenta", "cincuenta",
        "sesenta", "setenta", "ochenta", "noventa"
    };

    private static string[] especiales = {
        "diez", "once", "doce", "trece", "catorce", "quince",
        "dieciséis", "diecisiete", "dieciocho", "diecinueve"
    };

    public static string Convertir(int numero)
    {
        if (numero < 10)
        {
            return unidades[numero];
        }
        else if (numero < 20)
```



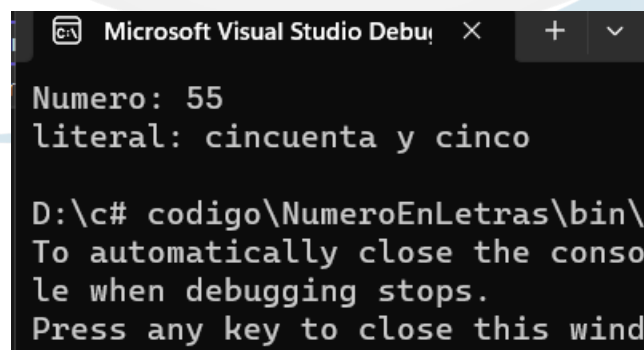
```

    {
        return especiales[numero - 10];
    }
    else
    {
        int unidad = numero % 10;
        int decena = numero / 10;
        return decenas[decena] + (unidad > 0 ? " y " +
        unidades[unidad] : "");
    }
}

class Program
{
    static void Main()
    {
        Console.Write("Numero: ");
        if (int.TryParse(Console.ReadLine(), out int numero))
        {
            string literal = Numero.Convertir(numero);
            Console.WriteLine($"literal: {literal}");
        }
        else
        {
            Console.WriteLine("Ingresar un numero valido.");
        }
    }
}

```

CORRIDO:



```

Microsoft Visual Studio Debug Console
Numero: 55
literal: cincuenta y cinco

D:\c# codigo\NumeroEnLetras\bin\
To automatically close the console
when debugging stops.
Press any key to close this window

```

2. Realizar un programa utilizando listas que te permita ingresar n números por teclado donde cada número entre en las siguientes listas:

Entrada

¿Cuántos números deseas añadir?

➤ 7

Añada 7 números:

- 2, 3, 5, 10, 5, 4, 6

Salida

- Lista 1: 2, 4, 6, 10 (Múltiplos de 2)
- Lista 2: 2, 3, 5 (Primos)
- Lista 3: 5, 10 (Múltiplos de 5)
- Lista 4: 6 (Perfectos)

PROGRAMA:

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

class Program
{
    static void Main()
    {
        Console.Write("Cuantos numeros deseas aniadir: ");
        if (int.TryParse(Console.ReadLine(), out int n))
        {
            List<int> numeros = new List<int>();

            for (int i = 0; i < n; i++)
            {
                Console.Write($"Aniada el numero [{i + 1}]: ");
                if (int.TryParse(Console.ReadLine(), out int num))
                {
                    numeros.Add(num);
                }
                else
                {
                    Console.WriteLine("Ingresa un numero valido.");
                    i--;
                }
            }

            List<int> multiplosDe2 = numeros.Where(x => x % 2 ==
0).ToList();
            List<int> primos = numeros.Where(x => Primo(x)).ToList();
            List<int> multiplosDe5 = numeros.Where(x => x % 5 ==
0).ToList();
            List<int> perfectos = numeros.Where(x =>
Perfecto(x)).ToList();

            Console.WriteLine("Múltiplos de 2: " + string.Join(", ",
multiplosDe2));
```

```

        Console.WriteLine("Primos: " + string.Join(", ", primos));
        Console.WriteLine("Múltiplos de 5: " + string.Join(", ",
multiplosDe5));
        Console.WriteLine("Perfectos: " + string.Join(", ",
perfectos));
    }
    else
    {
        Console.WriteLine("Ingrese un numero valido.");
    }
}

static bool Primo(int numero)
{
    if (numero <= 1)
        return false;

    for (int i = 2; i * i <= numero; i++)
    {
        if (numero % i == 0)
            return false;
    }
    return true;
}

static bool Perfecto(int numero)
{
    int sumaDivisores = 1;
    for (int i = 2; i * i <= numero; i++)
    {
        if (numero % i == 0)
        {
            sumaDivisores += i;
            if (i != numero / i)
                sumaDivisores += numero / i;
        }
    }
    return sumaDivisores == numero;
}
}

```

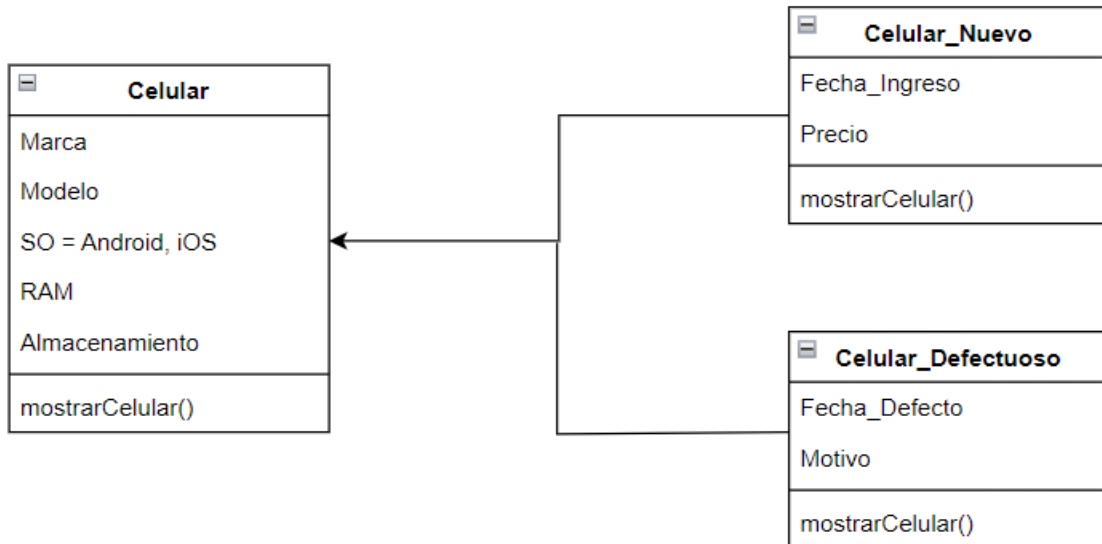
CORRIDO:

```

Cuantos numeros deseas aniadir: 7
Aniada el numero [1]: 2
Aniada el numero [2]: 3
Aniada el numero [3]: 5
Aniada el numero [4]: 7
Aniada el numero [5]: 6
Aniada el numero [6]: 10
Aniada el numero [7]: 496
Múltiplos de 2: 2, 6, 10, 496
Primos: 2, 3, 5, 7
Múltiplos de 5: 5, 10
Perfectos: 6, 496

```

3. Realizar un programa que tenga las siguientes clases utilizando polimorfismo y herencia. La clase Celular debe ser una clase abstracta.



PROGRAMA:

```

using System;
abstract class Celular
{
    public string Marca { get; set; }
    public string Modelo { get; set; }
    public string SistemaOperativo { get; set; }
    public int RAM { get; set; }
    public int Almacenamiento { get; set; }
    public Celular(string marca, string modelo, string
sistemaOperativo, int ram, int almacenamiento)
    {
        Marca = marca;
        Modelo = modelo;
        SistemaOperativo = sistemaOperativo;
    }
}

```

```
        RAM = ram;
        Almacenamiento = almacenamiento;
    }
    public abstract void MostrarCelular();
}
class CelularNuevo : Celular
{
    public DateTime FechaIngreso { get; set; }
    public double Precio { get; set; }

    public CelularNuevo(string marca, string modelo, string
sistemaOperativo, int ram, int almacenamiento, DateTime fechaIngreso,
double precio)
        : base(marca, modelo, sistemaOperativo, ram, almacenamiento)
    {
        FechaIngreso = fechaIngreso;
        Precio = precio;
    }
    public override void MostrarCelular()
    {
        Console.WriteLine($"Marca: {Marca}");
        Console.WriteLine($"Modelo: {Modelo}");
        Console.WriteLine($"Sistema Operativo: {SistemaOperativo}");
        Console.WriteLine($"RAM: {RAM} GB");
        Console.WriteLine($"Almacenamiento: {Almacenamiento} GB");
        Console.WriteLine($"Fecha de Ingreso:
{FechaIngreso.ToShortDateString()}");
        Console.WriteLine($"Precio: ${Precio}");
    }
}
class CelularDefectuoso : Celular
{
    public DateTime FechaDefecto { get; set; }
    public string Motivo { get; set; }
    public CelularDefectuoso(string marca, string modelo, string
sistemaOperativo, int ram, int almacenamiento, DateTime fechaDefecto,
string motivo)
        : base(marca, modelo, sistemaOperativo, ram, almacenamiento)
    {
        FechaDefecto = fechaDefecto;
        Motivo = motivo;
    }
    public override void MostrarCelular()
    {
        Console.WriteLine($"Marca: {Marca}");
        Console.WriteLine($"Modelo: {Modelo}");
        Console.WriteLine($"Sistema Operativo: {SistemaOperativo}");
        Console.WriteLine($"RAM: {RAM} GB");
```

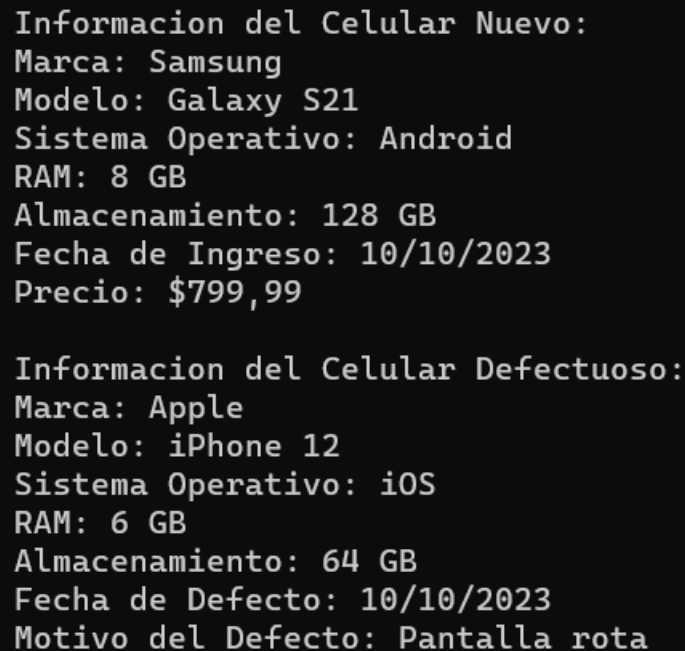
```

        Console.WriteLine($"Almacenamiento: {Almacenamiento} GB");
        Console.WriteLine($"Fecha de Defecto:
{FechaDefecto.ToShortDateString()}");
        Console.WriteLine($"Motivo del Defecto: {Motivo}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        CelularNuevo celularNuevo = new CelularNuevo("Samsung",
"Galaxy S21", "Android", 8, 128, DateTime.Now, 799.99);
        CelularDefectuoso celularDefectuoso = new
CelularDefectuoso("Apple", "iPhone 12", "iOS", 6, 64, DateTime.Now,
"Pantalla rota");
        Console.WriteLine("Informacion del Celular Nuevo:");
        celularNuevo.MostrarCelular();
        Console.WriteLine();
        Console.WriteLine("Informacion del Celular Defectuoso:");
        celularDefectuoso.MostrarCelular();
    }
}

```

CORRIDO:



```

Informacion del Celular Nuevo:
Marca: Samsung
Modelo: Galaxy S21
Sistema Operativo: Android
RAM: 8 GB
Almacenamiento: 128 GB
Fecha de Ingreso: 10/10/2023
Precio: $799,99

Informacion del Celular Defectuoso:
Marca: Apple
Modelo: iPhone 12
Sistema Operativo: iOS
RAM: 6 GB
Almacenamiento: 64 GB
Fecha de Defecto: 10/10/2023
Motivo del Defecto: Pantalla rota

```

4. Del ejercicio 3 crear una lista utilizando la clase Celular_Nuevo

- Añadir 10 celulares_nuevos.

- Crear una función Prom_Celular, utilizando expresiones lambda sacar el promedio del precio de los celulares.
- Crear una función Cel_MarcaS para buscar los celulares de Marca = Samsung, utilizando expresiones lambda.
- Crear una función Celular_RSA, utilizando consultas LinQ mostrar los celulares que son de RAM = 8GB, SO = Android y Almacenamiento de 128 GB.
- Crear una función Celular_Ingreso, utilizando consultas LinQ mostrar los celulares que ingresaron el año 2005.
- Crear dos funciones, la primera función usando: Expresiones lambda y la segunda función: consultas LinQ, donde se debe mostrar el modelo y el precio de los celulares Apple.

PROGRAMA:

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

class Program
{
    static void Main(string[] args)
    {
        Funciones funcion = new Funciones();
        funcion.AgregarCelulares();
        funcion.Prom_Celular();
        funcion.Cel_MarcaS();
        funcion.Celular_RSA();
        funcion.Celular_Ingreso();
        funcion.Exp_Lambda();
        funcion.Cons_Linq();
    }
}

class Celular
{
    public string Marca { get; set; }
    public string Modelo { get; set; }
    public string SistemaOperativo { get; set; }
    public int RAM { get; set; }
    public int Almacenamiento { get; set; }
    public Celular(string marca, string modelo, string
sistemaOperativo, int ram, int almacenamiento)
    {
        Marca = marca;
```



```
Modelo = modelo;
SistemaOperativo = sistemaOperativo;
RAM = ram;
Almacenamiento = almacenamiento;
}
}

class CelularNuevo : Celular
{
    public DateTime FechaIngreso { get; set; }
    public double Precio { get; set; }

    public CelularNuevo(string marca, string modelo, string
sistemaOperativo, int ram, int almacenamiento, DateTime fechaIngreso,
double precio)
        : base(marca, modelo, sistemaOperativo, ram, almacenamiento)
    {
        FechaIngreso = fechaIngreso;
        Precio = precio;
    }
}

class Funciones
{
    private List<CelularNuevo> celularesNuevos = new
List<CelularNuevo>();

    public void AgregarCelulares()
    {
        celularesNuevos.Add(new CelularNuevo("Samsung", "Galaxy S21",
"Android", 8, 128, new DateTime(2022, 1, 15), 799.99));
        celularesNuevos.Add(new CelularNuevo("Apple", "iPhone 12",
"iOS", 6, 64, new DateTime(2022, 3, 10), 899.99));
        celularesNuevos.Add(new CelularNuevo("Samsung", "Galaxy S22",
"Android", 12, 256, new DateTime(2022, 2, 5), 999.99));
        celularesNuevos.Add(new CelularNuevo("Apple", "iPhone 13",
"iOS", 6, 128, new DateTime(2022, 5, 20), 1099.99));
        celularesNuevos.Add(new CelularNuevo("Google", "Pixel 6",
"Android", 8, 128, new DateTime(2022, 4, 30), 799.99));
        celularesNuevos.Add(new CelularNuevo("OnePlus", "9 Pro",
"Android", 12, 256, new DateTime(2022, 6, 10), 899.99));
        celularesNuevos.Add(new CelularNuevo("Samsung", "Galaxy S21
FE", "Android", 6, 128, new DateTime(2022, 7, 1), 599.99));
        celularesNuevos.Add(new CelularNuevo("Apple", "iPhone SE",
"iOS", 4, 64, new DateTime(2022, 8, 15), 499.99));
        celularesNuevos.Add(new CelularNuevo("Xiaomi", "Mi 11",
"Android", 8, 256, new DateTime(2022, 9, 5), 699.99));
    }
}
```

```
        celularesNuevos.Add(new CelularNuevo("Samsung", "Galaxy A52",
"Android", 6, 128, new DateTime(2022, 10, 20), 499.99));
    }

    public void Prom_Celular()
    {
        Console.WriteLine("-----");
    };
    double promedioPrecio = celularesNuevos.Average(c =>
c.Precio);
    Console.WriteLine($"PPOMEDIO PRECIOS DE CELULARES:
${promedioPrecio:F2}\n");
}

    public void Cel_MarcaS()
    {
        Console.WriteLine("-----");
    };
    var celularesSamsung = celularesNuevos.Where(c =>
c.Marca.Equals("Samsung")).ToList();
    Console.WriteLine("CELULARES MARCA SAMSUNG:");
    foreach (var celular in celularesSamsung)
    {
        Console.WriteLine($"Modelo: {celular.Modelo}");
        Console.WriteLine($"Precio: ${celular.Precio}");
        Console.WriteLine();
    }
}

    public void Celular_RSA()
    {
        var resultado = from celular in celularesNuevos
                        where celular.RAM == 8 &&
celular.SistemaOperativo == "Android" && celular.Almacenamiento == 128
                        select celular;
        Console.WriteLine("-----");
    };
    Console.WriteLine("CELULARES CON RAM = 8GB, SO = ANDROID Y
ALAMCENAMIENTO DE 128 GB:");
    foreach (var celular in resultado)
    {
        Console.WriteLine($"Modelo: {celular.Modelo}");
        Console.WriteLine($"Precio: ${celular.Precio}");
        Console.WriteLine();
    }
}

    public void Celular_Ingreso()
```

```

{
    var resultado = from celular in celularesNuevos
                    where celular.FechaIngreso.Year == 2005
                    select celular;
    Console.WriteLine("-----");
");
    Console.WriteLine("CELULARES QUE INGRESARON EL ANIO 2005:");
    foreach (var celular in resultado)
    {
        Console.WriteLine($"Modelo: {celular.Modelo}");
        Console.WriteLine($"Precio: ${celular.Precio}");
        Console.WriteLine();
    }
}
public void Exp_Lambda()
{
    var celularesApple = celularesNuevos.Where(c =>
c.Marca.Equals("Apple"));
    Console.WriteLine("-----");
");
    Console.WriteLine("Modelo y Precio de los Celulares Apple
(Expresiones Lambda):");
    foreach (var celular in celularesApple)
    {
        Console.WriteLine($"Modelo: {celular.Modelo}");
        Console.WriteLine($"Precio: ${celular.Precio}");
        Console.WriteLine();
    }
}
public void Cons_Linq()
{
    var celularesApple = from celular in celularesNuevos
                        where celular.Marca.Equals("Apple")
                        select celular;
    Console.WriteLine("-----");
");
    Console.WriteLine("Modelo y Precio de los Celulares Apple
(Consultas LINQ):");
    foreach (var celular in celularesApple)
    {
        Console.WriteLine($"Modelo: {celular.Modelo}");
        Console.WriteLine($"Precio: ${celular.Precio}");
        Console.WriteLine();
    }
}
}

```

CORRIDO:

```
Microsoft Visual Studio Debug Console

-----
PPOMEDIO PRECIOS DE CELULARES: $779,99
-----

CELULARES MARCA SAMSUNG:
Modelo: Galaxy S21
Precio: $799,99

Modelo: Galaxy S22
Precio: $999,99

Modelo: Galaxy S21 FE
Precio: $599,99

Modelo: Galaxy A52
Precio: $499,99
-----

CELULARES CON RAM = 8GB, SO = ANDROID Y ALAMCENAMIENTO DE 128 GB:
Modelo: Galaxy S21
Precio: $799,99

Modelo: Pixel 6
Precio: $799,99
-----

CELULARES QUE INGRESARON EL ANIO 2005:
-----

Modelo y Precio de los Celulares Apple (Expresiones Lambda):
Modelo: iPhone 12
Precio: $899,99

Modelo: iPhone 13
Precio: $1099,99

Modelo: iPhone SE
Precio: $499,99
-----

Modelo y Precio de los Celulares Apple (Consultas LINQ):
Modelo: iPhone 12
Precio: $899,99

Modelo: iPhone 13
Precio: $1099,99

Modelo: iPhone SE
Precio: $499,99
```

5. Realizar las Historias de Usuario y el Product Backlog para la empresa ChocoMax

La empresa ChocoMax está ubicada en la ciudad de Tarija, donde la empresa se dedica a la elaboración y venta de chocolates. Actualmente la empresa gestiona la venta de los chocolates de forma manuscrita, también se detectó que no cuenta con un buen control de los

vendedores en que turno están o cuanto fue su venta en el día, lo cual genero perdidas económicas,

En la empresa ChocoMax existen dos turnos (turno mañana y turno tarde), cada vendedor trabaja solamente un turno. El Gerente general sólo le interesa la parte de reportes de las ventas por día, por mes y por vendedor, el vendedor requiere registrar las ventas, buscar o añadir los datos del cliente para emitir un recibo de la venta.

Historia de Usuario 1:

- **Como** gerente general.
- **Quiero** acceder a un sistema de gestión de ventas.
- **Para** mejorar el control y seguimiento de las operaciones de venta de la empresa.

Historia de Usuario 2:

- **Como** gerente general.
- **Quiero** poder ver informes de ventas diarios, mensuales y por vendedor.
- **Para** tomar decisiones basadas en datos sobre la gestión de la empresa.

Historia de Usuario 3:

- **Como** gerente general.
- **Quiero** recibir notificaciones de ventas excepcionales o problemas en tiempo real.
- **Para** abordar de manera proactiva cualquier problema que surja.

Historia de Usuario 4:

- **Como** vendedor.
- **Quiero** registrar las ventas de manera rápida y eficiente.
- **Para** ahorrar tiempo y mejorar la precisión en el registro de ventas.

Historia de Usuario 5:

- **Como** vendedor.
- **Quiero** buscar y agregar datos de clientes al registrar una venta.
- **Para** generar recibos de venta con la información del cliente.

Historia de Usuario 6:

- **Como** vendedor.
- **Quiero** tener un registro de las ventas realizadas durante mi turno.
- **Para** facilitar la reconciliación de ventas y el seguimiento de mis propias operaciones.

Historia de Usuario 7:

- **Como** vendedor.
- **Quiero** poder generar recibos de venta automáticamente.
- **Para** agilizar el proceso de ventas y brindar un comprobante al cliente de manera inmediata.

Historia de Usuario 8:

- **Como** vendedor.
- **Quiero** poder cambiar mi contraseña de acceso al sistema.
- **Para** garantizar la seguridad de mis datos de inicio de sesión.

Historia de Usuario 9:

- **Como** gerente general.
- **Quiero** acceder a un panel de control que muestre métricas clave sobre las operaciones de ventas.
- **Para** tener una visión rápida del desempeño de la empresa.

Historia de Usuario 10:

- **Como** vendedor.
- **Quiero** poder imprimir recibos de venta en papel para los clientes que lo soliciten.
- **Para** ofrecer opciones de comprobantes de venta a los clientes.

Product Backlog:

1. Crear una base de datos centralizada para almacenar registros de ventas y datos de clientes.
2. Desarrollar una interfaz de usuario para el sistema de gestión de ventas.
3. Implementar un sistema de notificaciones para el gerente general sobre ventas excepcionales o problemas.
4. Desarrollar una función de registro de ventas para los vendedores, que permita un registro rápido y eficiente.
5. Implementar una función de búsqueda y añadir datos de clientes al registrar una venta.
6. Desarrollar una función de generación automática de recibos de venta.
7. Implementar la funcionalidad de registro de ventas por turno para los vendedores.
8. Crear una función que permita a los vendedores cambiar su contraseña de acceso al sistema.
9. Desarrollar un panel de control con métricas clave sobre las operaciones de ventas para el gerente general.

- 10. Implementar una función de impresión de recibos de venta en papel para los clientes que lo soliciten.

6. Realizar las Historias de Usuario y el Product Backlog para mejorar el Sistema Dragón FNI

Historia de Usuario 1:

- **Como** Universitario.
- **Quiero** recibir notificaciones en tiempo real sobre cambios en mi horario o información académica importante.
- **Para** estar al tanto de cualquier actualización relevante sin tener que revisar constantemente el sistema.

Historia de Usuario 2:

- **Como** Universitario.
- **Quiero** poder solicitar y obtener un comprobante de inscripción en línea.
- **Para** tener acceso rápido a documentos necesarios para trámites administrativos.

Historia de Usuario 3:

- **Como** estudiante universitario.
- **Quiero** ver estadísticas de rendimiento académico, como promedios y tendencias a lo largo del tiempo.
- **Para** evaluar mi progreso y tomar decisiones informadas sobre mi educación.

Historia de Usuario 4:

- **Como** administrador del sistema.
- **Quiero** una función de importación de datos para cargar información de cursos y estudiantes de manera eficiente.
- **Para** facilitar la actualización de datos masivos.

Historia de Usuario 5:

- **Como** administrador del sistema.

- **Quiero** una función de generación de informes académicos automatizada para simplificar la generación de informes académicos periódicos.
- **Para** ahorrar tiempo en la preparación de informes.

Historia de Usuario 6:

- **Como** universitario.
- **Quiero** recibir recordatorios automáticos de fechas importantes, como fechas límite de inscripción o exámenes finales.
- **Para** asegurarme de no perder plazos importantes y estar bien preparado para los exámenes.

Historia de Usuario 7:

- **Como** administrador del sistema.
- **Quiero** una función de generación de informes personalizados para obtener datos específicos según sea necesario.
- **Para** facilitar la toma de decisiones basadas en datos y proporcionar información detallada cuando sea necesario.

Historia de Usuario 8:

- **Como** universitario.
- **Quiero** tener acceso a recursos educativos en línea, como material de lectura, presentaciones y ejercicios prácticos.
- **Para** mejorar mi aprendizaje y comprensión de los cursos.

Historia de Usuario 9:

- **Como** administrador del sistema.
- **Quiero** una función de gestión de eventos para programar y anunciar eventos académicos y extracurriculares en la facultad.
- **Para** mantener a los estudiantes informados sobre actividades importantes.

Historia de Usuario 10:

- **Como** administrador del sistema.
- **Quiero** una función de encuestas y retroalimentación en línea para recopilar opiniones y comentarios de los estudiantes.
- **Para** mejorar continuamente la calidad de los servicios académicos.

Historia de Usuario 11:

- **Como** universitario.
- **Quiero** poder registrar mis preferencias de horario para futuros semestres.
- **Para** ayudar en la planificación de horarios y cursos de manera anticipada.

Historia de Usuario 12:

- **Como** administrador del sistema.
- **Quiero** una función de seguimiento de solicitud de documentos académicos, como certificados y diplomas.
- **Para** asegurarme de que los estudiantes reciban sus documentos de manera oportuna.

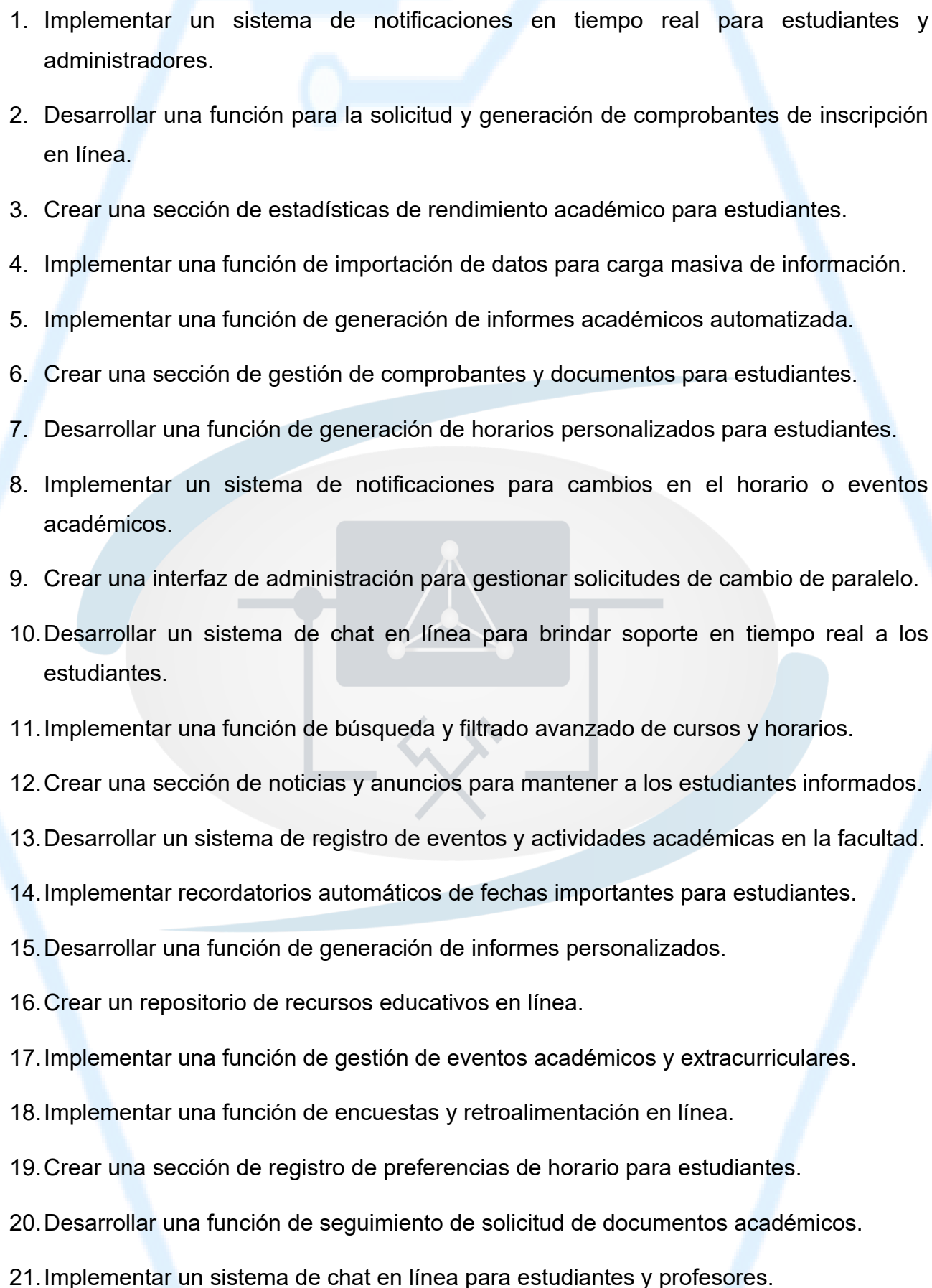
Historia de Usuario 13:

- **Como** estudiante universitario.
- **Quiero** una función de chat en línea para comunicarme fácilmente con compañeros de clase y profesores.
- **Para** facilitar la colaboración y el intercambio de información.

Historia de Usuario 14:

- **Como** administrador del sistema.
- **Quiero** una función de control de acceso y seguridad mejorada para proteger los datos de los estudiantes.
- **Para** garantizar la privacidad y seguridad de la información académica.

Product Backlog:

- 
1. Implementar un sistema de notificaciones en tiempo real para estudiantes y administradores.
 2. Desarrollar una función para la solicitud y generación de comprobantes de inscripción en línea.
 3. Crear una sección de estadísticas de rendimiento académico para estudiantes.
 4. Implementar una función de importación de datos para carga masiva de información.
 5. Implementar una función de generación de informes académicos automatizada.
 6. Crear una sección de gestión de comprobantes y documentos para estudiantes.
 7. Desarrollar una función de generación de horarios personalizados para estudiantes.
 8. Implementar un sistema de notificaciones para cambios en el horario o eventos académicos.
 9. Crear una interfaz de administración para gestionar solicitudes de cambio de paralelo.
 10. Desarrollar un sistema de chat en línea para brindar soporte en tiempo real a los estudiantes.
 11. Implementar una función de búsqueda y filtrado avanzado de cursos y horarios.
 12. Crear una sección de noticias y anuncios para mantener a los estudiantes informados.
 13. Desarrollar un sistema de registro de eventos y actividades académicas en la facultad.
 14. Implementar recordatorios automáticos de fechas importantes para estudiantes.
 15. Desarrollar una función de generación de informes personalizados.
 16. Crear un repositorio de recursos educativos en línea.
 17. Implementar una función de gestión de eventos académicos y extracurriculares.
 18. Implementar una función de encuestas y retroalimentación en línea.
 19. Crear una sección de registro de preferencias de horario para estudiantes.
 20. Desarrollar una función de seguimiento de solicitud de documentos académicos.
 21. Implementar un sistema de chat en línea para estudiantes y profesores.

22. Mejorar las medidas de seguridad y control de acceso en el sistema.

