Índice

Ejercicios Unidad 9

Descargar estos ejercicios

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_enumeraciones. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente.

Ejercicio 1. Sistema de gestión de niveles de dificultad

Programa que gestione diferentes niveles de dificultad en un videojuego.

```
Ejercicio 1: Sistema de gestión de niveles de dificultad

=== CONFIGURACIÓN DE DIFICULTAD ===
Niveles disponibles: Facil, Medio, Dificil, Extremo

Introduce el nivel deseado: Facil
Nivel seleccionado: Facil

=== ESTADÍSTICAS DEL NIVEL ===
Nivel: Facil
Vidas: 10
Puntos por enemigo: 5

¿Quieres probar otro nivel? (S/N): s

Introduce el nivel deseado: dfa
Nivel no válido. Niveles disponibles: Facil, Medio, Dificil, Extremo

Introduce el nivel deseado: Extremo
Nivel seleccionado: Extremo

=== ESTADÍSTICAS DEL NIVEL ===
Nivel: Extremo
Vidas: 1
Puntos por enemigo: 50

¿Quieres probar otro nivel? (S/N): n
¡Gracias por jugar!
```

Requisitos:

- Definir la enumeración **NivelDificultad** con valores: Facil, Medio, Dificil, Extremo.
- Método **RecogeNivel** devuelve un NivelDificultad válido a partir de la cadena que se le pida al usuario, gestionando los posibles errores:
  - Usar `Enum.GetNames` para mostrar niveles disponibles y `Enum.TryParse` para validación.
- Método **ObtenVidas** que devuelva el número de vidas según el nivel (Facil=10, Medio=5, Dificil=3, Extremo=1).

- Método **ObtenPuntosPorEnemigo** que devuelva puntos según el nivel (Facil=5, Medio=15, Dificil=30, Extremo=50).
- Método **MuestraEstadisticas** que muestre toda la información del nivel.

Ejercicio 2. Control de abono de transporte urbano

Programa que permita controlar el coste del abono de transporte urbano de una ciudad.

```
Ejercicio 2: Control de abono de transporte urbano

=== CALCULADORA DE ABONOS ===
Tipos de abono disponibles: QuinceDias, TreintaDias, FamiliasNumerosas, TerceraEdad, Di

Introduce el tipo de abono: juvenil
Tipo de abono seleccionado: Juvenil

Los abonos QuinceDias y TreintaDias tienen duración fija.
Para otros abonos, introduce días (mínimo 7, máximo 60): 15

=== DETALLES DEL ABONO ===
Tipo de abono: Juvenil
Precio por viaje: 0,65€
Días del abono: 15
Coste total del abono: 9,75€

¿Quieres calcular otro abono? (S/N): s

Introduce el tipo de abono: quincedias
Tipo de abono seleccionado: QuinceDias

=== DETALLES DEL ABONO ===
Tipo de abono: QuinceDias
Precio por viaje: 0,70€
Días del abono: 15 (fijo)
Coste total del abono: 10,50€

¿Quieres calcular otro abono? (S/N): n
¡Gracias por usar nuestro servicio!
```

Requisitos:

- Definir la enumeración **TipoAbono** con valores: QuinceDias=70, TreintaDias=60, FamiliasNumerosas=50, TerceraEdad=30, Discapacidad=20, Juvenil=65, Infantil=35, Turistico=90.
  - Los valores representan el precio por viaje en céntimos (dividir entre 100 para obtener euros).
- Método **RecogeAbono** que pida y devuelva el abono, realizando una validación correcta usando `Enum.TryParse`.

- Método **EsAbonoFijo** que determine si un abono tiene duración fija (QuinceDias=15 días, TreintaDias=30 días).
- Método **RecogeDias** Al que le llega el tipo de abono y usando el anterior, pide el número de días (7-60) para abonos no fijos, con validación y los devuelve.
- Método **CalculaCosteTotal** que calcule el coste multiplicando precio por viaje por número de días. A este método le llegará el tipo de abono a calcular y el total de días para los que se quiere el abono. Este método devolverá una tupla con el coste total y los días.
- Método **MuestraDetalle** que muestre toda la información del abono calculado.
- Usar bucles `do-while` para validación de entrada y control del programa principal.

Ejercicio 3. Gestión de estados de pedidos

Programa que gestione los estados de pedidos en una tienda online usando arrays.

```
Ejercicio 3: Gestión de estados de pedidos

=== SISTEMA DE PEDIDOS ===
Estados disponibles: Pendiente, Procesando, Enviado, Entregado, Cancelado

Número de pedidos a gestionar: r
Número de pedidos a gestionar: 5

--- ASIGNACIÓN DE ESTADOS ---
Pedido 1 - Introduce estado: pendiente
Pedido 2 - Introduce estado: procesando
Pedido 3 - Introduce estado: enviado
Pedido 4 - Introduce estado: entregaos
Estado no válido. Estados disponibles: Pendiente, Procesando, Enviado, Entregado, Cance
Pedido 4 - Introduce estado: entregado
Pedido 5 - Introduce estado: cancelado

=== RESUMEN DE PEDIDOS ===
Pedido 1: Pendiente
Pedido 2: Procesando
Pedido 3: Enviado
Pedido 4: Entregado
Pedido 5: Cancelado

=== ESTADÍSTICAS ===
Pendiente: 1 pedidos
Procesando: 1 pedidos
Enviado: 1 pedidos
Entregado: 1 pedidos
Cancelado: 1 pedidos

=== PEDIDOS ACTIVOS (no entregados ni cancelados) ===
Pedido 1: Pendiente
Pedido 2: Procesando
Pedido 3: Enviado

Total de pedidos activos: 3
```

Requisitos:

- Definir la enumeración **EstadoPedido** con valores: Pendiente, Procesando, Enviado, Entregado, Cancelado.

- Usar un array `EstadoPedido[]` para almacenar los estados de los pedidos.
- Método **AsignaEstados** que pida estados para cada pedido usando `Enum.TryParse`. Si se introduce entrada no valida, se mostrarán los estados y se volverá a pedir.
- Método **MuestraResumen** que muestre todos los pedidos con sus estados.
- Método **CuentaPorEstado** que cuente cuántos pedidos hay de cada estado usando bucles. Recuerda que cada elemento de una enumeración está asociado a un entero.
- Método **MuestraEstadisticas** que muestre el recuento de cada estado.
- Método **MuestraPedidosActivos** que muestre solo pedidos que no estén Entregado ni Cancelado.
- Usar bucles `for` y `foreach` para recorrer el array de pedidos.

Ejercicio 4. Dieta semanal vegetariana

Programa que genere de manera aleatoria una dieta semanal para las cenas basada en platos vegetarianos.

```
Ejercicio 4: Dieta semanal vegetariana

=== GENERADOR DE DIETA SEMANAL ===
Generando dieta aleatoria para la semana...

=== DIETA DE LA SEMANA ===
Lunes: PastaConPesto (450 calorías)
Martes: SushiVegetariano (500 calorías)
Miércoles: SopaDeLentejas (250 calorías)
Jueves: CazuelaDeVegetales (300 calorías)
Viernes: HamburguesaVegetal (300 calorías)
Sábado: WrapDeVerdurasYHummus (300 calorías)
Domingo: BerenjenasAlHornoConQueso (250 calorías)

=== ANÁLISIS NUTRICIONAL ===
Calorías totales de la semana: 2350
Promedio de calorías por día: 335.71

Día con menos calorías: Miércoles (SopaDeLentejas - 250 calorías)
Día con más calorías: Martes (SushiVegetariano - 500 calorías)

¿Quieres generar otra dieta? (S/N): s

=== NUEVA DIETA GENERADA ===
Lunes: EnsaladaDeQuinoa (350 calorías)
Martes: FalafelConEnsalada (400 calorías)
Miércoles: CremaDeCalabaza (150 calorías)
Jueves: PizzaVegetariana (500 calorías)
Viernes: ArrozFritoConTofu (400 calorías)
Sábado: ChilesRellenosDeQueso (350 calorías)
Domingo: GnocchiConSalsaDeTomate (450 calorías)

Día con menos calorías: Miércoles (CremaDeCalabaza - 150 calorías)

¿Quieres generar otra dieta? (S/N): n
¡Que disfrutes de tu dieta vegetariana!
```

Requisitos:

- Definir enumeración **DiaSemana** con valores: Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo.
- Definir enumeración **PlatoVegetariano** con valores: EnsaladaDeQuinoa, CurryDeGarbanzos, HamburguesaVegetal, SopaDeLentejas, PastaConPesto, FalafelConEnsalada, TortillaDeEspinacas, CremaDeCalabaza, SushiVegetariano, PizzaVegetariana, BowlDeAvenaConFrutas, SmoothieVerde, WrapDeVerdurasYHummus, ArrozFritoConTofu, CazuelaDeVegetales, ChilesRellenosDeQueso, GnocchiConSalsaDeTomate, BerenjenasAlHornoConQueso.
- Array **caloriasPlatos** con las calorías de cada plato: {350, 400, 300, 250, 450, 400, 200, 150, 500, 500, 350, 200, 300, 400, 300, 350, 450, 250}.
- Método **GeneraDietaSemana** que devuelva un array con 7 platos asignados aleatoriamente sin repetir.
- Método **MuestraDietaSemana** que muestre los platos asignados a cada día con formato específico.
- Método **DiaConMenosCalorias** que devuelva el día con el plato de menor cantidad de calorías.
- Método **DiaConMasCalorias** que devuelva el día con el plato de mayor cantidad de calorías.
- Método **CaloriasDieta** que calcule y devuelva las calorías totales de la semana.
- Método **PromedioCaloriasDiarias** que calcule el promedio de calorías por día.
- Usar la clase `Random` para generar la dieta aleatoria, bucles y llamadas a los métodos que resuelvan un problema,
  para evitar repeticiones.

Ejercicio 5. Sistema de turnos de trabajo con flags

Programa que gestione turnos de trabajo usando enumeración con flags.

```
Ejercicio 5: Sistema de turnos de trabajo con flags

=== GESTIÓN DE TURNOS ===
Empleado: Juan Pérez

Turnos disponibles:
M = Mañana, T = Tarde, N = Noche, F = FinDeSemana

Introduce turnos asignados (ej: MTF): mt

=== INFORMACIÓN DEL EMPLEADO ===
Empleado: Juan Pérez
Turnos: Mañana, Tarde
Horas semanales: 16
Salario base: 1280.00€

Operaciones disponibles:
A = Añadir turno, Q = Quitar turno, M = Mostrar info, S = Salir

Operación: a
Turno a añadir (M/T/N/F): f
Turno FinDeSemana añadido.

Operaciones disponibles:
A = Añadir turno, Q = Quitar turno, M = Mostrar info, S = Salir

Operación: q
Turno a quitar (M/T/N/F): t
Turno Tarde quitado.

Operación: m
=== INFORMACIÓN DEL EMPLEADO ===
Empleado: Juan Pérez
Turnos: Mañana, FinDeSemana
Horas semanales: 18
Salario base: 1480.00€

Operación: s
¡Hasta luego!
```

Requisitos:

- Definir enumeración **TurnoTrabajo** con atributo `[Flags]`: Ninguno=0b_0000_0000, Mañana=0b_0000_0001, Tarde=0b_0000_0010, Noche=0b_0000_0100, FinDeSemana=0b_0000_1000.
- Método **ParseaTurnos** que convierta una string con todos los caracteres de los turnos (parámetro de entrada al método), a la variable de retorno de tipo TurnoTrabajo con la combinación de turnos, usando operaciones de bits.
- Método **TieneTurno** al que le llegan dos parámetros de entrada de tipo TurnoTrabajo, uno con todos los turnos del empleado y otro con el turno a verificar. Con esto se debe verificar si un empleado tiene un turno específico usando operaciones de bits.
- Método **CalculaHorasSemanales** que calcule el total de horas semanales del empleado (parámetro de entrada TurnoTrabajo). Según los turnos que tenga, se irán añadiendo las horas según (Mañana=8h, Tarde=8h, Noche=8h, FinDeSemana=10h). Usa el método TieneTurno para comprobar si el empleado ha hecho ese turno.
- Método **CalculaSalario** método parecido al anterior, pero que calcule salario base (€10/hora normal, €15/hora noche, €20/hora fin de semana). Usa el método TieneTurno para comprobar si el empleado ha hecho ese turno.
- Método **AñadeTurno** al que le llegan los turnos actuales y el turno nuevo y devuelve los turnos con el nuevo añadido. Usando operador OR (`|`).
- Método **QuitaTurno** que quite un turno usando operadores AND y NOT (`& ~`). Igual que el anterior pero quitando un turno.
- Método **MuestraInformacion** que muestre toda la información del empleado como se puede ver en la salida. Al método le llegará el nombre del empleado y sus turnos.