FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma, Desarrollo de Aplicaciones Web**
MÓDULO:	**Programación**

UNIDAD 3: DISEÑO DE CLASES Y ENCAPSULACIÓN

ACTIVIDADES

AUTORES: **Fernando Rodríguez Alonso**
Sonia Pasamar Franco

CONSTRUCTORES Y MÉTODOS DE ACCESO A ATRIBUTOS

ACTIVIDAD 1x01

Codifica una clase **Videojuego** que incluya los datos y las operaciones siguientes:

- Información de un videojuego: nombre, desarrollador, año y número máximo de jugadores.
- Un constructor que reciba como parámetros un nombre, un desarrollador, un año y un número máximo de jugadores.
- Un constructor que reciba como parámetros un nombre, un desarrollador y un año y que indique por defecto que el número máximo de jugadores es 1.
- Un constructor que reciba como parámetros un nombre y un desarrollador y que indique por defecto que el año es 2021 y el número máximo de jugadores es 1.
- Un método de objeto **obtenerEstado** que devuelva el estado del videojuego como cadena de caracteres.

Codifica otra clase **Actividad_1x01** que incluya un programa principal **main** que:

- Lea por teclado un nombre, un desarrollador, un año y un número máximo de jugadores, instancie un objeto de esta clase **Videojuego** a partir de estos datos y visualice en consola el estado de este objeto.
- Lea por teclado un nombre, un desarrollador y un año, instancie un objeto de esta clase **Videojuego** a partir de estos datos y visualice en consola el estado de este objeto.
- Lea por teclado un nombre y un desarrollador, instancie un objeto de esta clase **Videojuego** a partir de estos datos y visualice en consola el estado de este objeto.

Un ejemplo de ejecución del programa podría ser:

```
VIDEOJUEGO 1
¿Nombre? Street Fighter 5
¿Desarrollador? Capcom
¿Año? 2016
¿Número Máximo de Jugadores? 2
Videojuego [Título = Street Fighter 5, Desarrollador = Capcom,
            Año = 2016, NúmeroMáximoJugadores = 2]
VIDEOJUEGO 2
¿Nombre? Need for Speed Heat
¿Desarrollador? Electronic Arts
¿Año? 2019
Videojuego [Título = Need for Speed Heat, Desarrollador = Electronic Arts,
            Año = 2019, NúmeroMáximoJugadores = 1]
VIDEOJUEGO 3
¿Nombre? Far Cry 6
¿Desarrollador? UbiSoft
Videojuego [Título = Far Cry 6, Desarrollador = UbiSoft,
            Año = 2021, NúmeroMáximoJugadores = 1]
```

ACTIVIDAD 1x02 (repaso)

Codifica una clase **Persona** que incluya los datos y las operaciones siguientes:

- Información de una persona: nombre, apellido, edad (en años), altura (en centímetros) y si está en activo o no (trabaja o no trabaja).
- Un constructor que reciba como parámetros un nombre, un apellido, una edad, una altura y si está en activo o no.
- Un constructor que reciba como parámetros un nombre, un apellido, una edad y una altura y que indique por defecto que no está en activo (no trabaja).
- Un constructor que no tenga parámetros y que indique por defecto que el nombre es “Recién”, el apellido es “Nacido”, la edad es 0, la altura es 50 y no está en activo (no trabaja).
- Un método de objeto **obtenerEstado** que devuelva el estado de la persona como cadena de caracteres.

Codifica otra clase **Actividad_1x02** que incluya un programa principal **main** que:

- Lea por teclado un nombre, un apellido, una edad, una altura y si está en activo o no, instancie un objeto de esta clase **Persona** a partir de estos datos y visualice en consola el estado de este objeto.
- Lea por teclado un nombre, un apellido, una edad y una altura, instancie un objeto de esta clase **Persona** a partir de estos datos y visualice en consola el estado de este objeto.
- Instancie un objeto de esta clase **Persona** con todos los datos por defecto y visualice en consola el estado de este objeto.

Un ejemplo de ejecución del programa podría ser:

```
PERSONA 1
¿Nombre? Juan
¿Apellido? Franco
¿Edad? 25
¿Altura? 183
¿Está en Activo? true
Persona [Nombre = Juan, Apellido = Franco,
         Edad = 25, Altura = 183, Activo = true]
PERSONA 2
¿Nombre? Sara
¿Apellido? González
¿Edad? 32
¿Altura? 171
Persona [Nombre = Sara, Apellido = González,
         Edad = 32, Altura = 171, Activo = false]
PERSONA 3
Persona [Nombre = Recién, Apellido = Nacido,
         Edad = 0, Altura = 50, Activo = false]
```

ACTIVIDAD 1x03

Codifica una clase **`Libro`** que incluya los datos y las operaciones siguientes:

- Información de un libro: título, número de ejemplares y precio.
- Un constructor que reciba como parámetros un título, un número de ejemplares y un precio.
- Un método de objeto **`obtenerEstado`** que devuelva el estado del libro como cadena de caracteres, indicando el precio con 2 dígitos decimales.
- Métodos de acceso a cada atributo del libro en modo escritura (*setters*).
- Métodos de acceso a cada atributo del libro en modo lectura (*getters*).

Codifica otra clase **`Actividad_1x03`** que incluya un programa principal **`main`** que gestione un libro con el menú de opciones siguiente:

0) Salir del programa.

1) Crear un libro a partir de datos leídos por teclado.

2) Visualizar en consola el libro.

3) Modificar el título del libro.

4) Modificar el número de ejemplares del libro.

5) Modificar el precio del libro.

6) Visualizar en consola el título del libro.

7) Visualizar en consola el número de ejemplares del libro.

8) Visualizar en consola el precio del libro.

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 8). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 8.
```

En las opciones 2, 3, 4, 5, 6, 7 y 8, se debe tener creado un libro (mediante la opción 1) antes de realizar la operación. Si esto no ocurre, visualizará en consola el mensaje:

```
No existe ningún libro creado.
```

Un ejemplo de ejecución del programa podría ser:

```
(0) Salir del programa.
(1) Crear un libro a partir de datos leídos por teclado.
(2) Visualizar en consola el libro.
(3) Modificar el título del libro.
(4) Modificar el número de ejemplares del libro.
(5) Modificar el precio del libro.
(6) Visualizar en consola el título del libro.
(7) Visualizar en consola el número de ejemplares del libro.
(8) Visualizar en consola el precio del libro.
¿Opción (0-8)? 10
La opción de menú debe estar comprendida entre 0 y 8.
```

```
...
(2) Visualizar en consola el libro.
...
¿Opción (0-8)? 2
No existe ningún libro creado.

...
(1) Crear un libro a partir de datos leídos por teclado.
...
¿Opción (0-8)? 1
¿Título? Parque Jurásico
¿Número de Ejemplares? 17
¿Precio? 29,95
Se ha creado un libro.

...
(2) Visualizar en consola el libro.
...
¿Opción (0-8)? 2
Libro [Título = Parque Jurásico, NúmeroEjemplares = 17, Precio = 29.95]

...
(4) Modificar el número de ejemplares del libro.
...
¿Opción (0-8)? 4
¿Número de Ejemplares? 13
Se ha modificado el número de ejemplares del libro.

...
(7) Visualizar en consola el número de ejemplares del libro.
...
¿Opción (0-8)? 7
Número de Ejemplares = 13

...
(5) Modificar el precio del libro.
...
¿Opción (0-8)? 5
¿Precio? 23,55
Se ha modificado el precio del libro.

...
(8) Visualizar en consola el precio del libro.
¿Opción (0-8)? 7
Precio = 23.55

...
(2) Visualizar en consola el libro.
...
¿Opción (0-8)? 2
Libro [Título = Parque Jurásico, NúmeroEjemplares = 13, Precio = 23.55]
```

DISEÑO DE CLASES

ACTIVIDAD 2x01

Codifica una clase **`Cuenta`** que incluya los datos y las operaciones siguientes:

- Información de una cuenta: número, cliente y saldo.
- Un constructor que reciba como parámetros un número y un cliente y que indique por defecto que el saldo es 0.
- Un método de objeto **`obtenerEstado`** que devuelva el estado de la cuenta como cadena de caracteres, indicando el saldo con 2 dígitos decimales.
- Un método de objeto **`ingresar`** que tenga como parámetro un importe a ingresar en la cuenta, que realice la operación cuando este importe sea positivo, y que devuelva un booleano indicando si el ingreso se ha realizado con éxito o no.
- Un método de objeto **`retirar`** que tenga como parámetro un importe a retirar de la cuenta, que realice la operación cuando este importe sea positivo y sea menor o igual que el saldo, y que devuelva un booleano indicando si la retirada se ha realizado con éxito o no.
- Un método de objeto **`transferir`** que tenga como parámetros un importe a transferir y una cuenta destino, que realice la operación cuando este importe sea positivo y sea menor o igual que el saldo de la cuenta origen, y que devuelva un booleano indicando si la transferencia se ha realizado con éxito o no.

Codifica otra clase **`Actividad_2x01`** que incluya un programa principal **`main`** que instancie dos objetos de la clase **`Cuenta`** (una cuenta con número 1 y cliente “Francisco” y otra cuenta con número 2 y cliente “Victoria”) y que gestione estas dos cuentas con el menú de opciones siguiente:

0) Salir del programa.

1) Visualizar en consola las dos cuentas.

2) Ingresar un importe en la cuenta 1.

Leerá por teclado un importe a ingresar.
Intentará ingresar este importe en la cuenta 1.
Si el ingreso se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha ingresado un importe en la cuenta 1.
```

Si no se ha podido realizar el ingreso, visualizará en consola los mensajes:

```
Error al ingresar un importe en la cuenta 1:
El importe debe ser positivo.
```

3) Retirar un importe de la cuenta 1.

Leerá por teclado un importe a retirar.
Intentará retirar este importe de la cuenta 1.
Si la retirada se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha retirado un importe de la cuenta 1.
```

Si no se ha podido realizar la retirada, visualizará en consola los mensajes:

```
Error al retirar un importe de la cuenta 1:
El importe debe ser positivo.
El importe debe ser menor o igual que el saldo de la cuenta 1.
```

4) Ingresar un importe en la cuenta 2.

Leerá por teclado un importe a ingresar.
Intentará ingresar este importe en la cuenta 2.
Si el ingreso se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha ingresado un importe en la cuenta 2.
```

Si no se ha podido realizar el ingreso, visualizará en consola los mensajes:

```
Error al ingresar un importe en la cuenta 2:
El importe debe ser positivo.
```

5) Retirar un importe de la cuenta 2.

Leerá por teclado un importe a retirar.
Intentará retirar este importe de la cuenta 2.
Si la retirada se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha retirado un importe de la cuenta 2.
```

Si no se ha podido realizar la retirada, visualizará en consola los mensajes:

```
Error al retirar un importe de la cuenta 2:
El importe debe ser positivo.
El importe debe ser menor o igual que el saldo de la cuenta 2.
```

6) Transferir un importe de la cuenta 1 a la cuenta 2.

Leerá por teclado un importe a transferir.
Intentará transferir este importe de la cuenta 1 a la cuenta 2.
Si la transferencia se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha transferido un importe de la cuenta 1 a la cuenta 2.
```

Si no se ha podido realizar la transferencia, visualizará en consola los mensajes:

```
Error al transferir un importe de la cuenta 1 a la cuenta 2:
El importe debe ser positivo.
El importe debe ser menor o igual que el saldo de la cuenta 1.
```

7) Transferir un importe de la cuenta 2 a la cuenta 1.

Leerá por teclado un importe a transferir.
Intentará transferir este importe de la cuenta 2 a la cuenta 1.
Si la transferencia se ha realizado correctamente, visualizará en consola el mensaje:

```
Se ha transferido un importe de la cuenta 2 a la cuenta 1.
```

Si no se ha podido realizar la transferencia, visualizará en consola los mensajes:

```
Error al transferir un importe de la cuenta 2 a la cuenta 1:
El importe debe ser positivo.
El importe debe ser menor o igual que el saldo de la cuenta 2.
```

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 7). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 7.
```

ACTIVIDAD 2x02

Codifica una clase **`Racional`** que incluya los datos y las operaciones siguientes:

- Información de un racional: numerador y denominador.
- Un constructor que reciba como parámetros un numerador y un denominador.
- Otro constructor que no reciba parámetros y que cree un racional de forma aleatoria:
  - El númerador será un número entero generado de forma aleatoria entre -10 y 10.
  - El denominador será un número entero generado de forma aleatoria entre 1 y 10.
- Un método de clase **`validar`** que tenga como parámetros un numerador y un denominador y que devuelva un booleano indicando si estos datos son válidos según las condiciones:
  - El numerador debe ser un número entero.
  - El denominador debe ser un número natural positivo (mayor que cero).
- Un método de objeto **`obtenerEstado`** que devuelva el estado del racional como cadena de caracteres, expresado con el formato `N/D`.
- Un método de objeto **`sumar`** que tenga como parámetro otro racional y que sume el racional más el otro racional de la siguiente manera:

$$\frac{a}{b}+\frac{c}{d}=\frac{a\times d}{b\times d}+\frac{b\times c}{b\times d}=\frac{a\times d+b\times c}{b\times d}$$

- Un método de objeto **`restar`** que tenga como parámetro otro racional y que reste el racional menos el otro racional de la siguiente manera:

$$\frac{a}{b}-\frac{c}{d}=\frac{a\times d}{b\times d}-\frac{b\times c}{b\times d}=\frac{a\times d-b\times c}{b\times d}$$

- Un método de objeto **`multiplicar`** que tenga como parámetro otro racional y que multiplique el racional por el otro racional de la siguiente manera:

$$\frac{a}{b}\times\frac{c}{d}=\frac{a\times c}{b\times d}$$

- Un método de objeto **`dividir`** que tenga como parámetro otro racional y que divida el racional entre el otro racional de la siguiente manera:

$$\frac{a}{b}\div\frac{c}{d}=\frac{a\times d}{b\times c}$$

Añade a esta clase **`Racional`** la siguiente operación **opcional** o de **ampliación**:

- Un método de objeto **`simplificar`** que no tenga ningún parámetro y que simplifique o reduzca el racional utilizando el máximo común divisor:
  - Si el numerador del racional es 0, el denominador del racional será 1.
  - Si el numerador del racional es mayor que 0, el racional simplificado o reducido se obtendrá de la siguiente forma:

$$numerador=\frac{numerador}{mcd(numerador,denominador)}$$

$$denominador=\frac{denominador}{mcd(numerador,denominador)}$$

- Si el numerador del racional es menor que 0, el racional simplificado o reducido se obtendrá de la siguiente forma:

$$numerador = \frac{numerador}{mcd(-numerador, denominador)}$$

$$denominador = \frac{denominador}{mcd(-numerador, denominador)}$$

Codifica otra clase **`Actividad_2x02`** que incluya un programa principal **`main`** que gestione un racional con el menú de opciones siguiente:

0) Salir del programa.

1) Leer por teclado un racional válido.

Realizará el siguiente proceso iterativo hasta obtener unos datos válidos para un racional:
- Leerá por teclado un numerador y un denominador.
- Validará que estos datos cumplan las condiciones de un racional.
- Si estos datos no son válidos, visualizará en consola los mensajes:

```
Los datos del racional no son válidos:
El denominador debe ser positivo.
```

Instanciará un objeto de la clase **`Racional`** con estos datos válidos y visualizará en consola el mensaje:

```
Se ha creado un racional válido.
```

2) Visualizar en consola el racional.

Si no existe ninguna instancia de la clase **`Racional`**, visualizará en consola el mensaje:

```
No existe ningún racional válido creado.
```

Si existe una instancia de la clase **`Racional`**, visualizará en consola este racional expresado con el formato `N/D`.

3) Sumar el racional más otro racional.

Si no existe ninguna instancia de la clase **`Racional`**, visualizará en consola el mensaje:

```
No existe ningún racional válido creado.
```

Si existe una instancia de la clase **`Racional`**:
- Instanciará otro objeto de la clase **`Racional`** con el numerador y el denominador generados de forma aleatoria.
- Visualizará en consola los dos racionales con el formato `N/D`.
- Calculará la suma del racional más el otro racional y visualizará en consola el racional resultante con el formato `N/D`.

4) Restar el racional menos otro racional.

Si no existe ninguna instancia de la clase **`Racional`**, visualizará en consola el mensaje:

```
No existe ningún racional válido creado.
```

Si existe una instancia de la clase **`Racional`**:
- Instanciará otro objeto de la clase **`Racional`** con el numerador y el denominador generados de forma aleatoria.
- Visualizará en consola los dos racionales con el formato `N/D`.
- Calculará la resta del racional menos el otro racional y visualizará en consola el racional resultante con el formato `N/D`.

5) Multiplicar el racional por otro racional.

Si no existe ninguna instancia de la clase **`Racional`**, visualizará en consola el mensaje:

`No existe ningún racional válido creado.`

Si existe una instancia de la clase **`Racional`**:

- Instanciará otro objeto de la clase **`Racional`** con el numerador y el denominador generados de forma aleatoria.
- Visualizará en consola los dos racionales con el formato `N/D`.
- Calculará la multiplicación del racional por el otro racional y visualizará en consola el racional resultante con el formato `N/D`.

6) Dividir el racional entre otro racional.

Si no existe ninguna instancia de la clase **`Racional`**, visualizará en consola el mensaje:

`No existe ningún racional válido creado.`

Si existe una instancia de la clase **`Racional`**:

- Instanciará otro objeto de la clase **`Racional`** con el numerador y el denominador generados de forma aleatoria.
- Visualizará en consola los dos racionales con el formato `N/D`.
- Calculará la división del racional entre el otro racional y visualizará en consola el racional resultante con el formato `N/D`.

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 6). Si no lo es, visualizará en consola el mensaje:

`La opción de menú debe estar comprendida entre 0 y 6.`

ACTIVIDAD 2x03

Codifica una clase **`Tiempo`** que incluya los datos y las operaciones siguientes:

- Información de un tiempo: horas, minutos y segundos.
- Un constructor que reciba como parámetros unas horas, unos minutos y unos segundos.
- Un método de clase **`validar`** que tenga como parámetros unas horas, unos minutos y unos segundos y que devuelva un booleano indicando si estos datos son válidos según las condiciones:
  - Las horas deben estar comprendidas entre 0 y 23.
  - Los minutos deben estar comprendidos entre 0 y 59.
  - Los segundos deben estar comprendidos entre 0 y 59.
- Un método de objeto **`obtenerEstado`** que devuelva el estado del tiempo como cadena de caracteres, expresado con el formato `hh:mm:ss`.
- Un método de objeto **`sumar`** que tenga como parámetro otro tiempo y que calcule la suma del tiempo más el otro tiempo:
  - Si la suma produce un tiempo igual o inferior a `23h:59m:59s`, modificará el tiempo con la suma y devolvera verdadero para indicar que la suma se ha realizado con éxito.
  - Si la suma produce un tiempo superior a `23h:59m:59s`, no modificará el tiempo y devolverá falso para indicar que ha habido un desbordamiento superior de tiempo.
- Un método de objeto **`restar`** que tenga como parámetro otro tiempo y que calcule la resta del tiempo menos el otro tiempo:
  - Si la resta produce un tiempo igual o superior a `00h:00m:00s`, modificará el tiempo con la resta y devolverá verdadero para indicar que la resta se ha realizado con éxito.
  - Si la resta produce un tiempo inferior a `00h:00m:00s`, no modificará el tiempo y devolverá falso para indicar que ha habido un desbordamiento inferior de tiempo.

Codifica otra clase **`Actividad_2x03`** que incluya un programa principal **`main`** que gestione un tiempo con el menú de opciones siguiente:

0) Salir del programa.

1) Leer por teclado un tiempo válido.

Realizará el siguiente proceso iterativo hasta obtener unos datos válidos para un tiempo:
- Leerá por teclado unas horas, unos minutos y unos segundos.
- Validará que estos datos cumplan las condiciones de un tiempo.
- Si estos datos no son válidos, visualizará en consola los mensajes:

```
Los datos del tiempo no son válidos:
Las horas deben estar comprendidas entre 0 y 23.
Los minutos deben estar comprendidos entre 0 y 59.
Los segundos deben estar comprendidos entre 0 y 59.
```

Instanciará un objeto de la clase **`Tiempo`** con estos datos válidos y visualizará en consola el mensaje:

```
Se ha creado un tiempo válido.
```

2) Visualizar en consola el tiempo.

Si no existe ninguna instancia de la clase **`Tiempo`**, visualizará en consola el mensaje:

```
No existe ningún tiempo válido creado.
```

Si existe una instancia de la clase **`Tiempo`**, visualizará en consola este tiempo expresado con el formato `hh:mm:ss`.

3) Sumar al tiempo otro tiempo.

Si no existe ninguna instancia de la clase **`Tiempo`**, visualizará en consola el mensaje:

```
No existe ningún tiempo válido creado.
```

Si existe una instancia de la clase **`Tiempo`**:

- Instanciará otro objeto de la clase **`Tiempo`** con otros datos válidos, siguiendo los pasos descritos en la opción 1.
- Visualizará en consola los dos tiempos con el formato `hh:mm:ss` y calculará la suma de ellos.
- Si la suma se ha realizado con éxito, visualizará en consola el tiempo resultante.
- Si la suma ha producido un desbordamiento superior de tiempo, visualizará en consola los mensajes:

```
Error al sumar el tiempo más otro tiempo:
La suma debe ser igual o inferior a 23h:59m:59s.
```

4) Restar al tiempo otro tiempo.

Si no existe ninguna instancia de la clase **`Tiempo`**, visualizará en consola el mensaje:

```
No existe ningún tiempo válido creado.
```

Si existe una instancia de la clase **`Tiempo`**:

- Instanciará otro objeto de la clase **`Tiempo`** con otros datos válidos, siguiendo los pasos descritos en la opción 1.
- Visualizará en consola los dos tiempos con el formato `hh:mm:ss` y calculará la resta de ellos.
- Si la resta se ha realizado con éxito, visualizará en consola el tiempo resultante.
- Si la resta ha producido un desbordamiento inferior de tiempo, visualizará en consola los mensajes:

```
Error al restar el tiempo menos otro tiempo:
La resta debe ser igual o superior a 00h:00m:00s.
```

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 4). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 4.
```

ACTIVIDAD 2x04 (ampliación)

Codifica una clase **`Fecha`** que incluya los datos y las operaciones siguientes:

- Información de una fecha: día, mes y año.
- Un constructor que reciba como parámetros un día, un mes y un año.
- Un método de clase **`esBisiesto`** que tenga como parámetro un año y que devuelva verdadero si el año es bisiesto o falso en caso contrario. Un año es bisiesto si cumple dos condiciones:
  - El año es divisible entre 4.
  - El año no es divisible entre 100 o el año es divisible entre 400.
- Un método de clase **`obtenerNumeroDiasDeMes`** que tenga como parámetros un mes y un año y que devuelva el número de días que tiene el mes del año:
  - Febrero tiene 28 días si el año no es bisiesto y tiene 29 días si el año es bisiesto.
  - Abril, junio, septiembre y noviembre tienen 30 días.
  - Enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días.
- Un método de clase **`validar`** que tenga como parámetros un día, un mes y un año y que devuelva un booleano indicando si estos datos son válidos según las condiciones:
  - Se sigue el calendario gregoriano, que se basa en años bisiestos y años no bisiestos.
  - El año debe estar comprendido entre 1583 y 2999.
  - El mes debe estar comprendido entre 1 y 12.
  - El día debe estar comprendido entre 1 y el número de días que tiene el mes del año.
- Un método de objeto **`obtenerEstadoCorto`** que devuelva el estado de la fecha como cadena de caracteres, expresado con el formato `"DD/MM/AAAA"`.
- Un método de objeto **`obtenerEstadoLargo`** que devuelva el estado de la fecha como cadena de caracteres, expresado con el formato `"DD de <nombre_mes> de AAAA"`, siendo `<nombre_mes>` el nombre del mes (enero, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre o diciembre) correspondiente al mes `MM` (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 o 12).
- Un método de objeto **`sumar`** que tenga como parámetro un número de días y que calcule la suma de la fecha más ese número de días:
  - Si la suma produce una fecha igual o inferior al `31/12/2999`, modificará la fecha con la suma y devolverá verdadero para indicar que la suma se ha realizado con éxito.
  - Si la suma produce una fecha superior al `31/12/2999`, no modificará la fecha y devolverá falso para indicar que ha habido un desbordamiento superior de fecha.
- Un método de objeto **`restar`** que tenga como parámetro un número de días y que calcule la resta de la fecha menos ese número de días:
  - Si la resta produce una fecha igual o superior al `01/01/1583`, modificará la fecha con la resta y devolverá verdadero para indicar que la resta se ha realizado con éxito.
  - Si la resta produce una fecha inferior al `01/01/1583`, no modificará la fecha y devolverá falso para indicar que ha habido un desbordamiento inferior de fecha.
- Un método de clase **`obtenerNumeroDiasDesdeInicio`** que tenga como parámetro una fecha y que devuelva el número de días que hay entre el inicio del calendario (`01/01/1583`) y esa fecha.
- Un método de objeto **`obtenerNumeroDiasEntreFechas`** que tenga como parámetro otra fecha y que devuelva el número de días que hay entre la fecha y la otra fecha.

Codifica otra clase **Actividad_2x04** que incluya un programa principal **main** que gestione una fecha con el menú de opciones siguiente:

0) Salir del programa.

1) Leer por teclado una fecha válida.

Realizará el siguiente proceso iterativo hasta obtener unos datos válidos para una fecha:
- Leerá por teclado un día, un mes y un año.
- Validará que estos datos cumplan las condiciones de una fecha.
- Si estos datos no son válidos, visualizará en consola los mensajes:

```
Los datos de la fecha no son válidos:
El año debe estar comprendido entre 1583 y 2999.
El mes debe estar comprendido entre 1 y 12.
El día debe estar comprendido entre 1 y
el número de días que tiene el mes del año.
```

Instanciará un objeto de la clase **Fecha** con estos datos válidos y visualizará en consola el mensaje:

```
Se ha creado una fecha válida.
```

2) Visualizar en consola la fecha con dos formatos.

Si no existe ninguna instancia de la clase **Fecha**, visualizará en consola el mensaje:

```
No existe ninguna fecha válida creada.
```

Si existe una instancia de la clase **Fecha**, visualizará en consola esta fecha expresada con el formato `"DD/MM/AAAA"` y con el formato `"DD de <nombre_mes> de AAAA"`.

3) Sumar a la fecha un número de días.

Si no existe ninguna instancia de la clase **Fecha**, visualizará en consola el mensaje:

```
No existe ninguna fecha válida creada.
```

Si existe una instancia de la clase **Fecha**:
- Leerá por teclado un número de días, que deberá ser positivo.
- Visualizará en consola la fecha y el número de días y calculará la suma de ellos.
- Si la suma se ha realizado con éxito, visualizará en consola la suma resultante.
- Si la suma ha producido un desbordamiento superior de fecha, visualizará en consola los mensajes:

```
Error al sumar la fecha más el número de días:
La suma debe ser igual o inferior al 31/12/2999.
```

4) Restar a la fecha un número de días.

Si no existe ninguna instancia de la clase **Fecha**, visualizará en consola el mensaje:

```
No existe ninguna fecha válida creada.
```

Si existe una instancia de la clase **Fecha**:
- Leerá por teclado un número de días, que deberá ser positivo.
- Visualizará en consola la fecha y el número de días y calculará la resta de ellos.
- Si la resta se ha realizado con éxito, visualizará en consola la resta resultante.
- Si la resta ha producido un desbordamiento inferior de fecha, visualizará en consola los mensajes:

```
Error al restar la fecha menos el número de días:
La resta debe ser igual o superior al 01/01/1583.
```

5) Obtener el número de días entre la fecha y otra fecha.

Si no existe ninguna instancia de la clase **`Fecha`**, visualizará en consola el mensaje:

```
No existe ninguna fecha válida creada.
```

Si existe una instancia de la clase **`Fecha`**:

- Instanciará otro objeto de la clase **`Fecha`** con otros datos válidos, siguiendo los pasos descritos en la opción 1.
- Visualizará en consola las dos fechas.
- Calculará el número de días que hay entre las dos fechas y lo visualizará en consola.

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 5). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 5.
```