# Tarea online 3

Pulse el enlace para ver la tarea a pantalla completa

UNIÓN EUROPEA
Fondo Social Europeo



# Tarea online

**Título de la tarea:** Trabajando con clases, objetos y métodos.

**Unidad:** 03

**Ciclo formativo y módulo:** DAM/DAW, Programación.

**Curso académico:** 2021/22

## ¿Qué contenidos o resultados de aprendizaje trabajaremos?

Escribe y prueba programas sencillos, reconociendo y aplicando los fundamentos de la programación orientada a objetos.

## Estado de la entrega

Esta tarea aceptará entregas desde el **viernes, 26 de noviembre de 2021, 00:00**

| | |
|---|---|
| Número del intento | Este es el intento 1. |
| Estado de la entrega | No entregado |
| Estado de la calificación | Sin calificar |
| Fecha de entrega | martes, 30 de noviembre de 2021, 23:55 |
| Fecha límite | martes, 30 de noviembre de 2021, 23:55 |
| Última modificación | - |
| Comentarios de la entrega | Comentarios (0) |

# 1.- Descripción de la tarea.

## Caso práctico

Gerd Altmann Pixabay License.

**María** está entrenándose en la programación orientada a objetos. Está empezando a realizar pruebas, utilizando métodos, creando objetos, en fin, familiarizándose poco a poco con el lenguaje Java y alguna de las bibliotecas o librerías que lleva incorporadas el paquete básico de desarrollo.

Ahora mismo está construyendo pequeñas aplicaciones para usar algunas clases que han realizado algunos de sus compañeros (clases `Dado` y `CuentaBancaria`), así como las clases relacionadas con el uso de las fechas y las horas en Java (`LocalDate`, `LocalTime`, `LocalDateTime`). También está practicando con el uso de la **documentación javadoc** que se ha generado para esas clases y está observando lo útil que puede llegar a ser disponer de este tipo de documentación. ¡Le están literalmente "salvando la vida"!

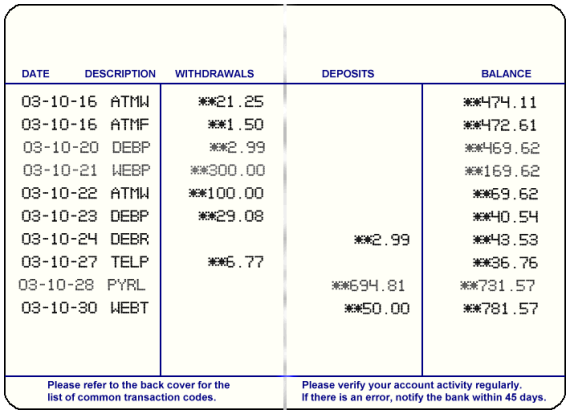
### ¿Qué te pedimos que hagas?

Las actividades a realizar se centrarán en el uso y manipulación de objetos mediante operaciones sencillas, trabajando con métodos de las clases `CuentaBancaria`, `Dado` y `LocalTime` para obtener una serie de resultados, **siendo obligatorio el uso de objetos de esas clases** según corresponda en cada ejercicio. Para ello es vital que consultes la **documentación javadoc** de cada una de esas clases, bien a través de la documentación de la API de Java (en el caso de la clase `LocalTime`) bien a través de la documentación javadoc que se te proporciona en esta tarea (para el caso de las clases `CuentaBancaria` y `Dado`).

**Debes usar obligatoriamente el proyecto NetBeans que se te proporciona** en el apartado **información de interés** de esta tarea, el cual incluye una **biblioteca específica para esta tarea** con las clases `CuentaBancaria` y `Dado`, que no forman parte de la API de Java.

# 1.1.- Ejercicio 1: uso de cuentas bancarias.

La clase **CuentaBancaria** implementa un modelo simplificado de cuenta para gestionar saldos, descubiertos, ingresos, extracciones, transferencias, embargos, etc. Es importante que le eches un vistazo a la **documentación javadoc** de esta clase antes de empezar a trabajar con ella. Así podrás ver qué tipo de información contiene, qué constructores proporciona, qué restricciones se le pueden aplicar, cuáles son los métodos disponibles, etc. Recuerda que tendrás que hacer el **import** correspondiente para poder usar esta clase en tu programa.



| DATE | DESCRIPTION | WITHDRAWALS | DEPOSITS | BALANCE |
|---|---|---|---|---|
| 03-10-16 | ATMW | **21.25 | | **474.11 |
| 03-10-16 | ATMF | **1.50 | | **472.61 |
| 03-10-20 | DEBP | **2.99 | | **469.62 |
| 03-10-21 | WEBP | **300.00 | | **169.62 |
| 03-10-22 | ATMW | **100.00 | | **69.62 |
| 03-10-23 | DEBP | **29.08 | | **40.54 |
| 03-10-24 | DEBR | | **2.99 | **43.53 |
| 03-10-27 | TELP | **6.77 | | **36.76 |
| 03-10-28 | PYRL | | **694.81 | **731.57 |
| 03-10-30 | WEBT | | **50.00 | **781.57 |

Please refer to the back cover for the list of common transaction codes.

Please verify your account activity regularly. If there is an error, notify the bank within 45 days.

Sergio Ortega. *Passbook sample* (CC BY-SA)

Para practicar con la creación y uso de objetos de este tipo, vamos a trabajar con **tres cuentas bancarias**: una cuenta que llamaremos "privada", otra cuenta que llamaremos "conjunta" y una tercera cuenta que llamaremos "familiar". En los tres casos se tratará de objetos instancia de la clase **CuentaBancaria**.

En el proyecto de trabajo que se proporciona para esta tarea dispones de un programa **Ejercicio01** donde tendrás que llevar a cabo las siguientes acciones paso a paso:

1. **Declarar tres variables referencia** a objetos instancia de la clase **CuentaBancaria**. Podrías llamarlas, por ejemplo: **CuentaPrivada**, **CuentaConjunta** y **CuentaFamiliar**.
2. **Instanciar tres objetos de la clase** **CuentaBancaria** a los cuales apuntarán cada una de las variables anteriores. Pero antes, haremos un par de intentos con errores para comprobar que funciona el lanzamiento de excepciones por parte de sus constructores:
    1. **Intentar crear una cuenta con fecha no válida** (**01/09/2027**). Recuerda que para instanciar un objeto de una determinada clase debes invocar a un constructor de esa clase mediante el operador **new**. Dado que en este caso sabes que la llamada al constructor va a fallar seguro, no tendrás más remedio que usar un bloque **try-catch** para capturar la excepción y gestionar ese error. Si no lo hace tu código, lo hará la máquina virtual de Java y tu programa se detendrá abruptamente ante el error, cosa que nunca debe suceder. Tu código siempre debe tener previsto este tipo de posibles errores.
    2. **Intentar crear otra cuenta con un saldo no válido** (**-200.00 euros**). Nuevamente, tendrás que usar otro bloque **try-catch** y capturar la excepción que se pueda producir al invocar al constructor para evitar que tu programa "aborte".
    3. **Crear una cuenta válida con un saldo inicial de 1000.00 euros, con fecha de creación a 1 de julio de 2021 y -200.00 euros de límite de descubierto**. Utiliza para ello el **constructor de tres parámetros**. Lo que devuelva el operador **new** al invocar al constructor es lo que tendrás que asignar a la variable referencia con la cual podrás manipular el objeto. En este caso será la variable **CuentaPrivada**.
    4. **Crear una cuenta válida con un saldo inicial de 200.00 euros y con fecha de creación a 1 de julio de 2021**, sin indicar nada más. Utiliza para ello el **constructor de dos parámetros**. En este caso, utilizarás la variable **CuentaConjunta** para apuntar a ese nuevo objeto recién creado tras la llamada al constructor.
    5. **Crear una cuenta válida con los valores por omisión**. Utiliza para ello el **constructor sin parámetros**. En este caso, utilizarás la variable **CuentaFamiliar** para apuntar al nuevo objeto creado. Recuerda que cuando decimos que una variable de tipo referencia va a "apuntar" a un objeto significa simplemente que va a almacenar en

una variable (de tipo referencia, obviamente) aquello que devuelva el operador `new` tras la invocación al constructor (es decir, una referencia a la zona de memoria donde se encuentran realmente los atributos del objeto).

3. **Obtener la siguiente información de la cuenta privada y mostrarla por pantalla**. Para ello tendrás que usar algunos de sus métodos *getter*:
    1. **Identificador** de cuenta.
    2. **Fecha de creación** de la cuenta.
    3. **Límite de descubierto** de la cuenta.
    4. Si la cuenta está **embargada** o no.
    5. Si la cuenta está en **descubierto** o no.
    6. El **número de días** que lleva la cuenta abierta.
4. Llevar a cabo las siguientes **operaciones sobre las cuentas**:
    1. **Ingresar 100.00 euros** en la **cuenta familiar**.
    2. **Extraer 100.00 euros** de la **cuenta conjunta**.
    3. **Transferir 1100.00 euros** de la **cuenta privada a la familiar**.
5. Muestra **información la información sobre el estado final de cada una las cuentas** que proporciona el método `toString`):
    1. Estado final de la **cuenta privada**.
    2. Estado final de la **cuenta conjunta**.
    3. Estado final de la **cuenta familiar**.

# Resultado del ejercicio

Dado que no hay que introducir datos de entrada y todos los valores son fijos, la ejecución de tu programa siempre deberá proporcionar el mismo resultado. Aquí tienes una muestra de cómo podría quedar:

Mostrar retroalimentación

```
USANDO CUENTAS BANCARIAS
------------------------
Fecha actual: 2021-08-24

Creación de cuentas (constructores)
-----------------------------------
Intentando crear cuenta privada con una fecha no válida.
Error: Parámetros de creación de la cuenta inválidos. Fecha de creación no válic

Intentando crear cuenta privada con un saldo no válido.
Error: Parámetros de creación de la cuenta inválidos. Saldo inicial: -200,00

Creando cuenta privada válida con un constructor con tres parámetros...
Cuenta privada creada: Id: 0 - Saldo:     1000,00 - Embargada: no

Creando cuenta conjunta válida usando un constructor con dos parámetros...
Cuenta conjunta creada: Id: 1 - Saldo:      200,00 - Embargada: no

Creando cuenta familiar válida usando un constructor sin parámetros...
```

```
Cuenta familiar creada: Id: 2 - Saldo:        0,00 - Embargada: no

Prueba de los getters de la cuenta privada:
-------------------------------------------
Id:  0
Fecha de creación:  2021-07-01
Límite de descubierto:  -200.0
Está embargada:  false
Está en descubierto:  false
Número de días que lleva la cuenta abierta:  55

Realización de operaciones sobre las cuentas
--------------------------------------------
Ingresamos 100 euros en la cuenta familiar...
Extraemos 100 euros de la cuenta conjunta...
Transferimos 1100 euros de la cuenta privada a la familiar...

Estado final de las cuentas:
---------------------------:
Estado final de la cuenta privada:  Id: 0 - Saldo:     -100,00 - Embargada: no
Estado final de la cuenta conjunta: Id: 1 - Saldo:      100,00 - Embargada: no
Estado final de la cuenta familiar: Id: 2 - Saldo:     1200,00 - Embargada: no
```

Recuerda que para resolver esta actividad debes utilizar obligatoriamente instancias de la clase **CuentaBancaria** junto con sus métodos.

Usando la clase **Dado**, que se proporciona en el proyecto de trabajo, construir el programa de prueba **Ejercicio02** que ejecute las siguientes acciones:

1. Llevar a cabo una **consulta inicial de valores globales de la clase Dado**, de manera que muestre en pantalla la siguiente información:
    1. **número de total de dados creados** hasta el momento (debería obtenerse cero);
    2. **número de total de lanzamientos** llevados a cabo hasta el momento (debería obtenerse cero);
    3. **cantidad de veces que han salido las caras con valores 1, 2, 3 y 4** en total, entre todos los dados creados hasta el momento (también debería ser cero para cada una de ellas).
2. **Creación y lanzamiento de dados**:
    1. **intentos de creación**: debes **intentar crear 10 objetos de tipo Dado** que tengan **aleatoriamente entre 0 y 8 caras** (bucle). Solo algunas llamadas al constructor funcionarán y en esos casos es cuando se podrá lanzar el dado. El resto de invocaciones al constructor harán que salte una excepción de tipo **IllegalArgumentException**. En tales casos habrá que recogerla y mostrar el mensaje de error por pantalla;
    2. en los casos en los que se pueda lanzar el dado, se harán tantos lanzamientos del objeto **Dado** creado como caras tenga ese dado. Es decir, que tendrás que implementar otro bucle dentro del bucle de intento de creación de dados. Cada vez que se realice un lanzamiento no se debe escribir nada en pantalla. Se mostrará el resultado completo una vez realizados todos;
    3. Una vez realizados todos los lanzamientos observamos los resultados obtenidos con ese dado mostrando por pantalla:
        1. la serie histórica de lanzamientos del dado;
        2. la suma total de todos los lanzamientos del dado.
3. **Consulta final de valores globales de la clase Dado**. Una vez finalizado el paso anterior, pasaremos a mostrar por pantalla los resultados de la misma consulta que tuviste que realizar al principio del programa, aunque ahora los resultados deberán ser muy diferentes (ya no serán cero):
    1. **número de total de dados creados** hasta el momento;
    2. **número de total de lanzamientos** llevados a cabo hasta el momento;
    3. **cantidad de veces que han salido las caras con valores 1, 2, 3 y 4** en total, entre todos los dados creados hasta el momento.

Recuerda que tendrás que hacer el **import** correspondiente para poder usar esta clase en tu programa.

Para resolver este ejercicio **no debes usar más variables que las de referencia a objetos de la clase Dado junto con los contadores de los bucles y el número aleatorio que vayas generando para el número de caras de cada dado**. El resto de variables son innecesarias y su uso será penalizado. Toda la información que requieras la podrás obtener consultando a los métodos apropiados de los objetos **Dado**, que para eso los tienes.

# Ejemplos de ejecución

Dado que no hay que introducir datos de entrada y que el número de caras con las que se va a intentar crear cada dado en cada uno de los diez intentos será aleatorio (entre 0 y 10), la ejecución de tu programa será imprevisible y cada vez que lo

ejecutes proporcionará un resultado diferente. Para que se repita un mismo resultado tendrá que ser mucha coincidencia.

Para que puedas comparar entre los resultados que aparezcan tras la ejecución de tu programa y algunas de la infinidad de posibles salidas que podrían obtenerse, te proporcionamos tres posibles ejemplos de las decenas de posibilidades que podrían producirse y que dependerán del azar. Recuerda que se trata de algunas de las muchas posibilidades que pueden darse, así que no te empeñes en que tus resultados coincidan con alguno de estos. Podrías necesitar decenas de ejecuciones para obtener el mismo resultado. Se trata de que los resultados que tú obtengas tengan sentido, del mismo modo que lo tienen los que te vamos a ofrecer.

### Primera prueba de ejecución

Mostrar retroalimentación

```
LANZANDO LOS DADOS
------------------

1.-CONSULTA INICIAL DE VALORES GLOBALES
---------------------------------------

Número de total de dados creados hasta el momento: 0.
Número de total de lanzamientos llevados a cabo hasta el momento: 0.
Número de veces que se ha obtenido 1: 0
Número de veces que se ha obtenido 2: 0
Número de veces que se ha obtenido 3: 0
Número de veces que se ha obtenido 4: 0

2.- CREACIÓN Y LANZAMIENTO DE DADOS
-----------------------------------

Intento 1: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 2: Intentando crear dado aleatorio de  8 caras. Correcto. Creado dado de
Lo lanzamos 8 veces: SEIS UNO DOS DOS TRES OCHO CINCO UNO
Suma total de los lanzamientos del dado: 28
Intento 3: Intentando crear dado aleatorio de  3 caras. Error. Numero de caras r
Intento 4: Intentando crear dado aleatorio de  2 caras. Error. Numero de caras r
Intento 5: Intentando crear dado aleatorio de  5 caras. Error. Numero de caras r
Intento 6: Intentando crear dado aleatorio de  2 caras. Error. Numero de caras r
Intento 7: Intentando crear dado aleatorio de  5 caras. Error. Numero de caras r
Intento 8: Intentando crear dado aleatorio de  4 caras. Correcto. Creado dado de
Lo lanzamos 4 veces: DOS DOS TRES CUATRO
Suma total de los lanzamientos del dado: 11
Intento 9: Intentando crear dado aleatorio de  6 caras. Correcto. Creado dado de
Lo lanzamos 6 veces: CUATRO CINCO CINCO UNO CUATRO SEIS
Suma total de los lanzamientos del dado: 25
Intento 10: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras

3.-ANÁLISIS DE RESULTADOS FINALES
---------------------------------
Número de total de dados creados hasta el momento: 3.
Número de total de lanzamientos llevados a cabo hasta el momento: 18.
```

```
Número de veces que se ha obtenido 1 entre todos los lanzamientos de todos los c
Número de veces que se ha obtenido 2 entre todos los lanzamientos de todos los c
Número de veces que se ha obtenido 3 entre todos los lanzamientos de todos los c
Número de veces que se ha obtenido 4 entre todos los lanzamientos de todos los c
```

Como puedes observar, en este caso se ha logrado instanciar **tres objetos de tipo dado**. Uno de **ocho caras**, otro de **cuatro** y el último de **seis**, realizándose ocho, cuatro y seis lanzamientos respectivamente.

## Segunda prueba de ejecución

Mostrar retroalimentación

```
LANZANDO LOS DADOS
------------------

1.-CONSULTA INICIAL DE VALORES GLOBALES
---------------------------------------

Número de total de dados creados hasta el momento: 0.
Número de total de lanzamientos llevados a cabo hasta el momento: 0.
Número de veces que se ha obtenido 1: 0
Número de veces que se ha obtenido 2: 0
Número de veces que se ha obtenido 3: 0
Número de veces que se ha obtenido 4: 0

2.- CREACIÓN Y LANZAMIENTO DE DADOS
-----------------------------------

Intento 1: Intentando crear dado aleatorio de  0 caras. Error. Numero de caras r
Intento 2: Intentando crear dado aleatorio de  4 caras. Correcto. Creado dado de
Lo lanzamos 4 veces: TRES UNO CUATRO TRES
Suma total de los lanzamientos del dado: 11
Intento 3: Intentando crear dado aleatorio de  5 caras. Error. Numero de caras r
Intento 4: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 5: Intentando crear dado aleatorio de  6 caras. Correcto. Creado dado de
Lo lanzamos 6 veces: DOS DOS SEIS TRES CINCO DOS
Suma total de los lanzamientos del dado: 20
Intento 6: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 7: Intentando crear dado aleatorio de  2 caras. Error. Numero de caras r
Intento 8: Intentando crear dado aleatorio de  0 caras. Error. Numero de caras r
Intento 9: Intentando crear dado aleatorio de  2 caras. Error. Numero de caras r
Intento 10: Intentando crear dado aleatorio de  7 caras. Error. Numero de caras

3.-ANÁLISIS DE RESULTADOS FINALES
---------------------------------
Número de total de dados creados hasta el momento: 2.
Número de total de lanzamientos llevados a cabo hasta el momento: 10.
Número de veces que se ha obtenido 1 entre todos los lanzamientos de todos los c
Número de veces que se ha obtenido 2 entre todos los lanzamientos de todos los c
```

```
Número de veces que se ha obtenido 3 entre todos los lanzamientos de todos los c
Número de veces que se ha obtenido 4 entre todos los lanzamientos de todos los c
```

En este caso el azar ha querido que solamente se lleguen a crear **dos dados**, uno de **cuatro** caras y otro de **seis**.

## Tercera prueba de ejecución

Mostrar retroalimentación

```
LANZANDO LOS DADOS
------------------

1.-CONSULTA INICIAL DE VALORES GLOBALES
---------------------------------------

Número de total de dados creados hasta el momento: 0.
Número de total de lanzamientos llevados a cabo hasta el momento: 0.
Número de veces que se ha obtenido 1: 0
Número de veces que se ha obtenido 2: 0
Número de veces que se ha obtenido 3: 0
Número de veces que se ha obtenido 4: 0

2.- CREACIÓN Y LANZAMIENTO DE DADOS
-----------------------------------

Intento 1: Intentando crear dado aleatorio de  7 caras. Error. Numero de caras r
Intento 2: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 3: Intentando crear dado aleatorio de  2 caras. Error. Numero de caras r
Intento 4: Intentando crear dado aleatorio de  7 caras. Error. Numero de caras r
Intento 5: Intentando crear dado aleatorio de  6 caras. Correcto. Creado dado de
Lo lanzamos 6 veces: UNO TRES CUATRO TRES UNO DOS
Suma total de los lanzamientos del dado: 14
Intento 6: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 7: Intentando crear dado aleatorio de  6 caras. Correcto. Creado dado de
Lo lanzamos 6 veces: TRES UNO CUATRO CUATRO DOS TRES
Suma total de los lanzamientos del dado: 17
Intento 8: Intentando crear dado aleatorio de  6 caras. Correcto. Creado dado de
Lo lanzamos 6 veces: TRES DOS CINCO CUATRO SEIS UNO
Suma total de los lanzamientos del dado: 21
Intento 9: Intentando crear dado aleatorio de  1 caras. Error. Numero de caras r
Intento 10: Intentando crear dado aleatorio de  4 caras. Correcto. Creado dado c
Lo lanzamos 4 veces: TRES DOS UNO UNO
Suma total de los lanzamientos del dado: 7

3.-ANÁLISIS DE RESULTADOS FINALES
---------------------------------
Número de total de dados creados hasta el momento: 4.
Número de total de lanzamientos llevados a cabo hasta el momento: 22.
Número de veces que se ha obtenido 1 entre todos los lanzamientos de todos los c
```

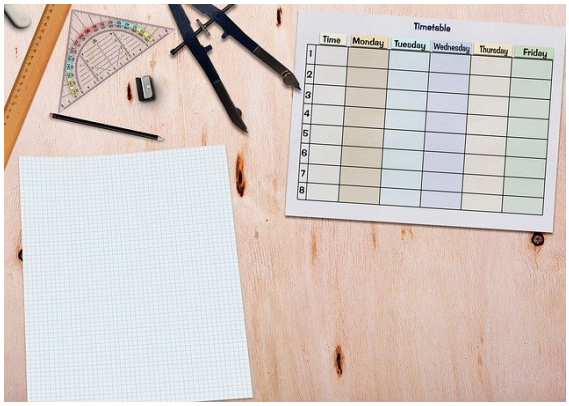
```
Número de veces que se ha obtenido 2 entre todos los lanzamientos de todos los d
Número de veces que se ha obtenido 3 entre todos los lanzamientos de todos los d
Número de veces que se ha obtenido 4 entre todos los lanzamientos de todos los d
```

En este último ejemplo de prueba se han llegado a crear **cuatro dados**: **tres de seis caras** y **uno de cuatro**.

# 1.3.- Ejercicio 3: horario de clases.

Teniendo en cuenta el siguiente horario de clases para un día específico de seis horas de docencia:

1. las clases comienzan a las 8:00 con dos horas de *Programación*;
2. a continuación hay dos horas de *Sistemas Informáticos*;
3. finalmente se termina con dos horas de *Entornos de Desarrollo*.




Darkmoon (Licencia Pixabay)

Escribe un programa en Java (`Ejercicio03`) que, dada una hora determinada introducida por teclado, indique la clase en la que te encuentras. Para ello habrá que llevar a cabo las siguientes acciones:

1. **Creación un primer objeto** de la clase `LocalTime` con la **hora de inicio de las clases**, es decir, las **8:00**.
2. **Lectura de teclado de una hora** determinada (hora y minutos):
    1. primero se solicitará la hora (entre 0 y 23) y mientras no se introduzca una hora válida (porque no esté en el rango o porque ni siquiera sea un número entero) se volverá a solicitar hasta que se disponga de una hora válida;
    2. a continuación se solicitará el minuto (entre 0 y 59) y se realizará la misma validación que en el caso de la hora.
3. **Creación de un segundo objeto** de la clase `LocalTime` con la **hora y minuto válidos que se han introducido por teclado**.
4. A partir de esos dos objetos de tipo `LocalTime` habrá que **obtener el rango en el que se encuentra la hora introducida**, que tendrá que ser uno de los siguientes casos:
    1. **aún no han comenzado las clases**. En tal caso habrá que calcular **cuántos minutos faltan** para que comiencen las clases;
    2. estamos en clase de ***Programación***;
    3. estamos en clase de ***Sistemas Informáticos***;
    4. estamos en clase de ***Entornos de Desarrollo***;
    5. **ya han finalizado las clases**. En tal caso habrá que calcular **cuántos minutos han transcurrido** desde la finalización de las clases.
5. **Mostrar por pantalla**:
6. la hora introducida por teclado (objeto `LocalTime` en formato `hh:ss`);
    1. **el caso en que nos encontremos** para esa hora y, en caso de que hayamos tenido que calcularlo, cuántos minutos faltaban para empezar o cuántos minutos habían pasado desde la finalización de las clases;

**Para llevar a cabo los cálculos y comprobaciones del paso 4 solo debes usar esos dos objetos de tipo** `LocalTime` (hora de inicio y hora introducida). A partir de ellos y aprovechando los métodos proporcionados de esta clase (`isBefore`, `isAfter`, `until`, `plusHours`, etc.). También te será de utilidad el enumerado `ChronoUnit` para indicar que la unidad deseada en los cálculos es "minutos" (`ChronoUnit.MINUTES`). La idea de este ejercicio es que practiques con el uso de esos métodos, no que hagas cálculos "a mano" utilizando los valores de horas y minutos independientemente.

Lo que obtengas como resultado de las comprobaciones y cálculos anteriores será lo que tendrás que a mostrar como mensaje de salida. Puedes almacenarlo en una cadena de caracteres para mostrarlo en el bloque final de salida de resultados.

Recuerda que la clase `LocalTime` **no dispone de constructores públicos**, de manera que tendrás que usar los **métodos "fábrica" o pseudoconstructores** que proporcione para crear objetos instancia de esta clase.

# Ejemplos de ejecución

A continuación te proporcionamos algunos ejemplos de cómo podría quedar la ejecución del programa dependiendo de las posibles entradas que se puedan proporcionar.

### Ejemplo 1: 6:45

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): x
Error de lectura: no es un número entero válido.
Introduzca hora (00-23): -1
Introduzca hora (00-23): 24
Introduzca hora (00-23): 6
Introduzca minuto (00-59): x
Error de lectura: no es un número entero válido.
Introduzca minuto (00-59): -1
Introduzca minuto (00-59): 60
Introduzca minuto (00-59): 45
Hora introducida: 06:45
Clase correspondiente: Aún no ha comenzado la jornada. Faltan 75 minutos para cc
```

Como puedes observar, si se introducen números fuera del rango permitido se vuelve a solicitar la información. Por otro lado, si se introduce algo diferente a un número entero, el programa no debería "romperse" debido a una `InputMismatchException`, sino que la excepción debería ser capturada y convenientemente gestionada para a continuación volver a pedir el dato.

### Ejemplo 2: 8:00

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 8
Introduzca minuto (00-59): 0
Hora introducida: 08:00
Clase correspondiente: Programación.
```

### Ejemplo 3: 9:59

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 9
Introduzca minuto (00-59): 59
Hora introducida: 09:59
Clase correspondiente: Programación.
```

### Ejemplo 4: 10:21

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 10
Introduzca minuto (00-59): 21
Hora introducida: 10:21
Clase correspondiente: Sistemas informáticos.
```

### Ejemplo 5: 13:59

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 13
Introduzca minuto (00-59): 59
Hora introducida: 13:59
Clase correspondiente: Entornos de Desarrollo.
```

### Ejemplo 6: 14:00

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 14
Introduzca minuto (00-59): 0
Hora introducida: 14:00
Clase correspondiente: La jornada ya ha finalizado. Han pasado 0 minutos desde e
```

### Ejemplo 7: 15:30

Mostrar retroalimentación

```
HORARIO DE CLASES
-----------------
Introducción del horario que desea comprobar:
Introduzca hora (00-23): 15
```

# 2.- Información de interés.

## Recursos necesarios y recomendaciones

**Debes usar obligatoriamente el proyecto base NetBeans que te proporcionamos** aquí. Ese proyecto incluye una **biblioteca específica para esta tarea** (`libtarea3`) que contiene las clases `CuentaBancaria` y `Dado`. Esas clases no forman parte de la API de Java.

mohamed Hassan (Licencia Pixabay)

Es vital que utilices este proyecto porque si no lo haces no dispondrás de esas clases y no podrás implementar correctamente tus programas al no disponer de esas clases ni de la configuración necesaria de tus bibliotecas.

Además, este proyecto contiene tres programas (clases `Ejercicio01`, `Ejercicio02` y `Ejercicio03`) correspondientes a cada uno a los ejercicios que debes implementar. Cada uno tendrá su propio `main` con el esqueleto del programa y algunas indicaciones de las pautas que debes seguir y así orientarte un poco el trabajo que debes realizar. ¡Aprovéchalos!

**Descarga y utiliza este proyecto. ¡No crees tu propio proyecto en Netbeans! Si lo haces, tu tarea será considerada como NO APTA** pues no tendrá las características que necesitamos que tenga.

Descargar proyecto Netbeans para la tarea 3 (zip - 453.74 KB)

Una vez descargado el proyecto, renómbralo con el estilo de nombre que solemos darle al paquete de entrega de la tarea:

**Apellido1_Apellido2_Nombre_PROG_Tarea03**

Y ya podrás ponerte a trabajar sobre él. ¡Ánimo!

Aunque en el proyecto base que se te proporciona ya contiene la **documentación javadoc** de las clases `CuentaBancaria` y `Dado`. Te las proporcionamos también aparte por si quieres disponer de ellas para consultarlas independientemente del IDE Netbeans:

Descargar javadoc de las clases `CuentaBancaria` y `Dado` para la tarea 3 (zip - 419.05 KB)

Una vez descargado y descomprimido el paquete debes abrir con un navegador web el archivo `index.html` para empezar a navegar por el javadoc de esta API y consultar la documentación de cada clase.

# Indicaciones de entrega

Una vez realizada la tarea, el envío se realizará a través de la plataforma. Comprime la carpeta del proyecto NetBeans en un fichero `.zip` y nómbralo siguiendo las siguientes pautas:

**Apellido1_Apellido2_Nombre_PROG_Tarea03**

# 3.- Evaluación de la tarea.

## Criterios de evaluación implicados

En esta unidad se va a evaluar el resultao de aprendizaje RA2. "Escribe y prueba programas sencillos, reconociendo y aplicando los fundamentos de la programación orientada a objetos." y algunos criterios de evaluación correspondientes al resultado de aprendizaje RA5 "Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases". Ello se hará mediante los siguientes Criterios de Evaluación correspondientes a cada resultado de aprendizaje citado:

Criterios de Evaluación correspondientes al **RA2**:

- RA2.a) Se han identificado los fundamentos de la programación orientada a objetos.
- RA2.b) Se han escrito programas simples.
- RA2.c) Se han instanciado objetos a partir de clases predefinidas.
- RA2.d) Se han utilizado métodos y propiedades de los objetos.
- RA2.e) Se han escrito llamadas a métodos estáticos.
- RA2.f) Se han utilizado parámetros en la llamada a métodos.
- RA2.g) Se han incorporado y utilizado librerías de objetos.
- RA2.h) Se han utilizado constructores.
- RA2.i) Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples


Peggy_Marco (Pixabay Licens

Criterios de Evaluación correspondientes al **RA5**:

- RA5.a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.
- RA5.b) Se han aplicado formatos en la visualización de la información.
- RA5.c) Se han reconocido las posibilidades de entrada/salida del lenguaje y las librerías asociadas.

De manera general en esta tarea podremos comprobar que

- **Declaración** de **variables referencia**.
- Creación de objetos LocalDate.
- Creación de objetos CuentaBancaria válidos.
- **Obtención de información** de objetos CuentaBancaria mediante métodos ***getter***.

- Aplicación de métodos: **ingresar**, **extraer**, **transferir**.
- Obtención de información mediante el uso del método toString.
- Se muestra la **información correcta por pantalla**, utilizando el **formato apropiado**.
- Obtención de información de clase. Uso correcto de **métodos estáticos**.
- Generación de **números aleatorios**.
- Creación de objetos **Dado.** Gestión apropiada de **excepciones** en la invocación a **constructores**.
- Uso de los **métodos de lanzamiento**.
- Uso de los **métodos de obtención de información** (***getters***) para averiguar todos los lanzamientos obtenidos y su suma.
- **Creación de objetos** LocalTime. Uso de **métodos "*fábrica*"**. Solo es necesario crear dos objetos. Ninguno más.
- **Comprobación de ubicación de una hora** dentro de varios rangos horarios: uso de isBefore *y plusHours* usando únicamente los dos objetos LocalTime
- Obtención del **tiempo transcurrido entre dos horas**. Uso de until. Uso del enum ChronoUnit

## ¿Cómo valoramos y puntuamos tu tarea?

A continuación mostraremos los elementos que podrán penalizar en la evaluación de la tarea. Tenlos muy en cuenta al resolver los ejercicios:

1. **Los programas deben compilar**. Cualquier funcionalidad pedida en el enunciado debe poderse probar y ha de funcionar correctamente de acuerdo a las especificaciones del enunciado.**Si el programa ni siquiera compila, el ejercicio podría considerarse nulo y su puntuación podría llegar a ser 0.**
2. Se utilizan **estructuras de salto incondicional** para el **control del flujo**, o bien herramientas como return, exit(0) o cualquier otro mecanismo que no sea el **fin de la ejecución natural del programa**.
3. No se respeta la **estructura de bloques** propuesta en la plantilla: **entrada de datos**, **procesamiento** y **salida de resultados**.
4. **Algunos identificadores no cumplen con el convenio sobre "asignación de nombres a identificadores"** establecido para el lenguaje Java para constantes, variables, métodos, clases, etc.
5. **Se declaran identificadores que no tienen nombres significativos o descriptivos** que representen de alguna manera la información que están almacenando para que el código quede lo más claro, legible y autodocumentado posible.
6. No se observa una **corrección ortográfica y gramatical**, así como la **coherencia en las expresiones lingüísticas**, tanto en los **comentarios en el código** como en los **textos de los mensajes** que aparezcan en pantalla para pedir información de entrada al usuario o para mostrar resultados de salida. Deben evitarse **mensajes de entrada de datos y/o salida de resultados** inapropiados, descontextualizados, insuficientes o incorrectos.
7. **El código no está correctamente indentado. Es fundamental para poder observar e intuir rápidamente, y de forma visual, la estructura de los programas. Recuerda que NetBeans puede hacer esto por ti (selecciona el código a "*formatear*" o "*embellecer*" y pulsa Alt+Mayús.+F).**

En esta otra tabla puedes observar la puntuación máxima asignada a cada ejercicio de la tarea así como los principales elementos que se tendrán en cuenta a la hora de corregir cada uno de ellos:

## Rúbrica de la tarea (sobre 10 puntos)

| | |
|---|---|
| **Punto Control 1:** Se han identificado los fundamentos de la programación orientada a objetos, y utilizado en la creación de programas simples.<br>• RA2.a) Se han identificado los fundamentos de la programación orientada a objetos.<br>• RA2.b) Se han escrito programas simples. | 6,23% |
| **Punto Control 2:** Se han instanciado objetos a partir de clases predefinidas, la creación de estos objetos se hace con sentido para la resolución del problema planteado en cada caso, sin la creación excesiva de objetos y controlando las necesidades que solicita cada problema particular<br>• RA2.c) Se han instanciado objetos a partir de clases predefinidas. | 4,15% |
| **Punto de Control 3:** Se han utilizado métodos y propiedades de los objetos en el desarrollo de la soluciones de los ejercicios propuestos, no solo su definición si no también su utilización coherente.<br>• RA2.d) Se han utilizado métodos y propiedades de los objetos. | 6,23% |
| **Punto de Control 4:** Se han escrito llamadas a métodos estáticos, entendiendo su uso como métodos de clase y no de objeto, y facilitando de esta forma la resolución de cada problema concreto.<br>• RA2.e) Se han escrito llamadas a métodos estáticos. | 6,23% |
| **Punto de Control 5:** Se han utilizado parámetros en la llamada a métodos. La llamada a los método tanto de librería de clases de Java, como implementadas por el alumno/a incorporarán los parámetros correctamente definidos y verificados antes de su uso, eligiendo en cada caso la llamada con los parámetros que más se ajusten a las necesidades de cada problema.<br>• RA2.f) Se han utilizado parámetros en la llamada a métodos. | 4,15% |
| **Punto de Control 6:** Se han importado librerías de la API de Java descritas en la unidad que facilitan resolución de los problemas mediante la utilización de objetos y métodos conociendo su alcance y uso | 4,15% |

| | |
|---|---|
| correcto, eligiendo el método o métodos de las librerías más apropiados.<br>• RA2.g) Se han incorporado y utilizado librerías de objetos. | |
| **Punto de Control 7:** En la resolución de los problemas propuestos se han utilizado constructores más apropiados en cada caso particular, bien creados por el alumno/a, o bien haciendo uso de los ya definidos en las librerías de API de Java que se ha visto en la unidad.<br>• RA2.h) Se han utilizado constructores. | 2,08% |
| **Punto de Control 8:** Para la resolución de la tarea se utiliza un entorno integrado de desarrollo (IDE) en la creación y compilación de programas simples, en este caso NetBeans, presentando un proyecto agrupando la resolución de la tarea en archivos independientes dentro del proyecto.<br>• RA2.i) Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples | 2,08% |
| **Punto de Control 9:** La entrada salida de datos se realiza mediante la consola del IDE utilizando las clases necesarias y salvando los posibles errores que pudiesen generar de su uso en los diferentes problemas planteados.<br>• RA5.a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.<br>• RA5.c) Se han reconocido las posibilidades de entrada/salida del lenguaje y las librerías asociadas. | 11,76% |
| **Punto de Control 10:** Los resultados y la salida de datos se presentan acorde con los formatos de visualización de la información requeridos en cada problema.<br>• RA5.b) Se han aplicado formatos en la visualización de la información. | 2,94% |
| **Punto de Control 10:** Los ejercicios desarrollados funcionan correctamente de acuerdo a las especificaciones y limitaciones requeridas en la tarea<br>• RA2.a) Se han identificado los fundamentos de la programación orientada a objetos.<br>• RA2.b) Se han escrito programas simples. | 50,00% |

- RA2.c) Se han instanciado objetos a partir de clases predefinidas.
- RA2.d) Se han utilizado métodos y propiedades de los objetos.
- RA2.e) Se han escrito llamadas a métodos estáticos.
- RA2.f) Se han utilizado parámetros en la llamada a métodos.
- RA2.g) Se han incorporado y utilizado librerías de objetos.
- RA2.h) Se han utilizado constructores.
- RA2.i) Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples
- RA5.a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.
- RA5.b) Se han aplicado formatos en la visualización de la información.
- RA5.c) Se han reconocido las posibilidades de entrada/salida del lenguaje y las librerías asociadas.