

**17**

**Desarrollo WEB**

v1.00 Febrero 2018

**TEST PROJECT**

Cofinanciado por:

[1. Instrucciones generales 1](#_Toc507068089)

[2. Prueba 2](#_Toc507068090)

[3. Esquema de puntuaciones 13](#_Toc507068091)

# Instrucciones generales

**Hora límite de entrega: 18:30 horas**

Cada uno de los alumnos dispone de un PC con sistema operativo Windows 10 y el entorno de desarrollo integrado NetBeans.

Esta prueba de programación tiene una puntuación, en su conjunto, de **10** puntos y está compuesta por 3 ejercicios distintos denominados Ejercicio 1, Ejercicio 2 y Ejercicio 3.

Esos 3 ejercicios deberán desarrollarse en el lenguaje de programación Java utilizando el citado entorno de desarrollo Integrado **NetBeans**, en su versión 8.2.

Las propuestas serán subidas en la 3 tareas habilitadas a tal efecto en el curso 17-18MurciaSkills de la plataforma https://aulavirtual.murciaeduca.es.

# Prueba

**0.- Fundamentos matemáticos:**

En matemáticas se definen los *números primos* como aquellos números naturales que sólo son divisibles por sí mismos y por el número 1. El propio número 1 queda excluido de la lista de números primos.

Así, la lista con los primeros *números primos* es:

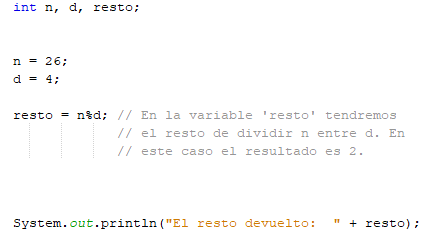
**2**, **3**, **5**, **7**, **11**, **13**, **17**, **19**...etc.

Por el contrario, son números *no primos*:

**4** (Es divisible por 2), **6** (Es divisible por 2 y 3), **8** (Es divisible por 2 y 4), **9** (Es divisible por 3), **10** (Es divisible por 2 y 5), **12** (Es divisible por 2, 3, 4 y 6)... etc.

Para averiguar si un número ***n*** es *no primo* es preciso encontrar para él un divisor. Y para saber si un número cualquiera ***d*** es divisor de otro número ***n*** sólo es necesario calcular el resto de la división de ***n*** entre ***d***. Si ese resto es 0, el número ***d*** es divisor de ***n*** y por tanto ***n*** es no primo. Si no pudiéramos encontrar un número ***d*** que divida a ***n***, entonces ***n*** es primo.

En el lenguaje de programación Java para obtener el resto de la división entre 2 variables (numéricas) se utiliza el operador **%**:



**1.- Ejercicio 1 de la prueba de programación**

Siempre hay un primo

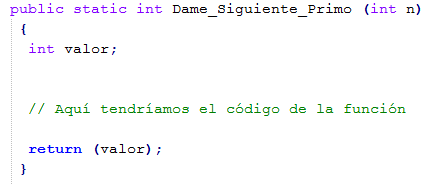
Este Ejercicio 1 está valorado con un máximo de **2.0** puntos del total de los **10.0** puntos de la prueba de programación. Se debe entregar en un proyecto NetBeans denominado **skills2018\_E1**.

Hace más de 2300 años Euclides demostró que para todo número siempre existe un número primo mayor que él. Es decir, existen infinitos números primos.

Teniendo en cuenta los conceptos expuestos en el apartado **0** de este enunciado, se pide desarrollar una función Java denominada *Dame\_Siguiente\_Primo ()* que reciba como argumento una variable de tipo entero P y devuelva el primer número primo que aparece a continuación de ese número P recibido como argumento.

Por ejemplo si el valor recibido por *Dame\_Siguiente\_Primo ()* fuese 5, el valor devuelto debería ser 7 (El 6 es no primo). Si el valor recibido fuese 24, el valor devuelto debería ser 29 (Los números 25, 26, 27 y 28 son no primos).

El prototipo de la función *Dame\_Siguiente\_Primo ()* sería:



La evaluación/calificación de esta función *Dame\_Siguiente\_Primo ()* por parte del tribunal se realizará invocando varias veces a la misma con diferentes valores y comprobando que devuelve, en cada caso, el número correcto.

**NO** está permitida la utilización de funciones matemáticas de librería ni funciones que calculen, precisamente, si un número es primo o no lo es. Sólo está permitida la utilización de la sintaxis de Java.

El argumento recibido por *Dame\_Siguiente\_Primo ()* será una variable de tipo *int* de Java, pero **siempre con valores positivos**. Por tanto, el menor número que podrá recibir esa función será 1 y mayor número será 2.147.483.647 (32 bits).

En la función *Dame\_Siguiente\_Primo ()* resulta imprescindible averiguar si un número es primo o no lo es. Así, los **2.0** puntos de este Ejercicio 1 se obtendrán **sólo** si el número de operaciones realizadas para averiguar si un número es primo es el mínimo posible. Es decir, se premia la optimización del código. Aunque se averigüe correctamente si un número es primo, si se realizaran más operaciones de las estrictamente necesarias, la calificación obtenida será **1.5** puntos.

**2.- Ejercicio 2 de la prueba de programación**

Información reservada

Este Ejercicio 2 está valorado con un máximo de **3.5** puntos del total de los **10.0** puntos de la prueba de programación. Se debe entregar en un proyecto NetBeans denominado **skills2018\_E2**.

Cierta agencia secreta vinculada a cierto gobierno ha recibido el encargo de almacenar el nombre de un espía. Lo que se necesita es que ese nombre se mantenga en secreto. Para ello en la agencia han decidido almacenar ese nombre en 2 matrices llamadas **Siguiente\_Celda** y **Letras** de tamaño 15x15que están contenidas en 2 archivos:

* El archivo denominado ***SC.bin*** contiene la matriz **Siguiente\_Celda**.
* El archivo denominado ***L.bin*** contiene la matriz **Letras**.

En cada uno de esos archivos se han depositado **225** celdas de información ordenadas por filas: Primero las **15** celdas de la fila **0**, a continuación las **15** celdas de la fila **1**, después las **15** celdas de la fila **2**… hasta llegar a las **15** celdas de la fila **14**.

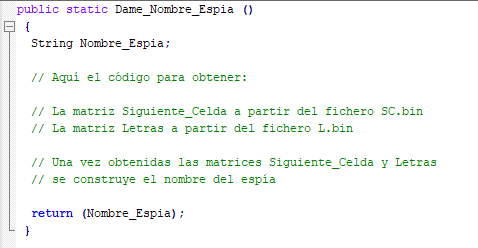
En el escritorio se dispone de esos 2 archivos ***SC.bin*** y ***L.bin***.

La matriz **Siguiente\_Celda** contendrá valores numéricos (tipo *int*) y la matriz **Letras** contendrá caracteres alfanuméricos (tipo *char*). En esta última matriz está almacenado, de forma codificada, el nombre del espía

Objetivo de este Ejercicio 2:

Desarrollar una función *Dame\_Nombre\_Espia ()* que, a partir de los archivos ***SC.bin*** y ***L.bin***, construya las 2 matrices (**Siguiente\_Celda** y **Letras**) de tamaño **15**x**15** y devuelva en una cadena de caracteres (Tipo *String* de Java) el nombre del espía que contienen.

El prototipo de la función *Dame\_Nombre\_Espia ()* sería:



Cuando se han obtenido las matrices **Siguiente\_Celda** y **Letras**, el algoritmo para reconstruir el nombre de un espía es iterativo. Se utiliza el valor contenido en cada celda de la matriz **Siguiente\_Celda** para encontrar la siguiente celda en la propia matriz **Siguiente\_Celda**. Además, ese mismo valor se utiliza para encontrar una celda en la matriz **Letras**. Las celdas de esa matriz **Letras** son, en realidad, las que contienen el nombre del espía.

1. En la primera iteración (Iteración 0) se parte de una celda inicial en la matriz **Siguiente\_Celda** (Localizada por sus coordenadas). Supongamos que esas coordenadas son F0 y C0.
2. En la iteración *i-ésima*, el contenido de la celda (Fi, Ci) en la matriz **Siguiente\_Celda** es un valor numérico **V** (tipo *int*). Con ese valor numérico **V** podemos calcular:
   1. La celda en la matriz **Letras** que contiene la letra *i-ésima* del nombre del espía (Fi, Ci).
   2. La siguiente celda de la matriz **Siguiente\_Celda** (Fi+1, Ci+1)
3. Si la celda obtenida dentro de la matriz **Letras** contiene un punto (.), el algoritmo termina. El nombre del espía está completo. Si no es así, será necesario ir al paso 2 con los nuevos valores (Fi+1, Ci+1) obtenidos en 2b.

Cuando el nombre está completo (Por haber encontrado un “.”) la función *Dame\_Nombre\_Espia ()* puede terminar y devolver la cadena con el nombre del espía construido.

Pero, ¿Cómo saber cuál es la primera celda (F0, C0) con la que iniciamos el algoritmo en la iteración 0?

El valor F0 será el *primer* número **V** primo contenido en la matriz **Siguiente\_Celda** empezando a contar desde la fila=0 y columna=0.

El valor C0 será el *último* número **V** primo contenido en la matriz **Siguiente\_Celda** empezando a contar desde la fila=0 y columna=0.

**Ayuda 1**: El fichero ***SC.bin*** tiene un tamaño de 900 bytes (225 int´s x 4 bytes). El fichero ***L.bin*** tiene un tamaño de 450 bytes (225 char´s x 2 bytes). Para leer de un fichero en Java se debe hacer:

1.1 Importar la librería de manejo de ficheros (java.io.RandomAccessFile):



1.2 Declarar los nombres de los descriptores de los ficheros:





1.3 Por cada valor a extraer del fichero ***SC.bin*** realizar la lectura:



Si se trata de un carácter en el fichero ***L.bin***:



1.4 Una vez leídos todos los valores, cerrar los ficheros:



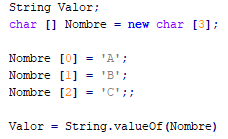


1.5 Todas las funciones implicadas en el manejo de ficheros (incluida *main ()*) deben añadir la cláusula de disparo de excepciones (throws Exception):





**Ayuda 2**: Para convertir un array de elementos de tipo *char* en una variable de tipo *String* se puede utilizar el método *valueOf()* de la clase *String*. Por ejemplo:



La calificación de este Ejercicio 2 se especifica a continuación:

* **1.0** puntos si se construye correctamente **Siguiente\_Celda** y **Letras** a partir de los ficheros ***SC.bin*** y ***L.bin*** respectivamente.
* **2.0** puntos si se obtiene correctamente el nombre del espía a partir de **Siguiente\_Celda** y **Letras**.
* **0.5** puntos si no se utilizan bucles anidados para recorrer **Siguiente\_Celda** y **Letras**.

**ATENCIÓN**: El participante podría intentar encontrar, de forma manual, el nombre del espía a partir de los ficheros ***SC.bin*** y ***L.bin***. Una vez hecho eso podría hacer que su función *Dame\_Nombre\_Espia ()* devolviese ese nombre encontrado. En tal caso, la calificación de este segundo ejercicio sería **0.0**, pues para la evaluación y calificación de las propuestas se utilizarán otros ficheros ***SC.bin*** y ***L.bin*** que contengan otros nombres de espías.

**3.- Ejercicio 3 de la prueba de programación**

Juego de cronos

Este Ejercicio 3 está valorado con un máximo de **4.5** puntos del total de los **10.0** puntos de la prueba de programación. Se debe entregar en un proyecto NetBeans denominado **skills2018\_E3**. Antes de comenzar la resolución de este Ejercicio 3 debe leerse el último párrafo la página 12 de este enunciado

Un fabricante de automóviles ha decidido falsear los datos de emisión de gases contaminantes en sus nuevos motores de gasóleo. El propósito es hacer pensar a los clientes (Y a las autoridades) que la emisión de esos gases por la combustión de los motores es inferior a la que realmente se producirá.

Para conseguir sus abyectos propósitos se necesita que en esos motores se controlen, mediante cronómetros, 2 *electroválvulas* que gestionan la emisión al exterior de esos gases.

Una electroválvula es un dispositivo que debe abrirse (Permitiendo la salida de gases) cada cierto tiempo ***C*** y permanecer abierta durante otro cierto tiempo ***A***. Transcurrido ese tiempo ***A***, la electroválvula debe cerrarse.

Por ejemplo, podría suceder que una de las electroválvulas debiera abrirse cada 750 milisegundos y debiera permanecer abierta durante 175 milisegundos.

Así, partiendo del instante t=0 milisegundos, las 3 primeras etapas serían:

|  |  |
| --- | --- |
| **Tiempo (Milisegundos)** | **Evento** |
| 0 |  |
| 750 (0+750) | Se abre la electroválvula (Por primera vez) |
| 925 (750+175) | Se cierra la electroválvula |
|  |  |
| 1500 (750+750) | Se abre la electroválvula (Por segunda vez) |
| 1675 (1500+175) | Se cierra la electroválvula |
|  |  |
| 2250 (1500+750) | Se abre la electroválvula (Por tercera vez) |
| 2420 (2250+175) | Se cierra la electroválvula |
|  |  |

La única condición indispensable es que el tiempo de apertura sea siempre inferior al intervalo entre aperturas consecutivas. Así, no debería suceder que se tenga que abrir una electroválvula cada 500 milisegundos y el tiempo de apertura fuese 600 milisegundos.

Los intervalos entre sucesivas aperturas y el tiempo de esas aperturas para las 2 electroválvulas están almacenados en el fichero ***Tiempos.bin***. En ese fichero se tienen 2 valores de tipo *int* y 4 valores de tipo *long* especificando:

* Número de electroválvula (0 ó 1)🡺Tipo *int*
* Intervalo entre sucesivas aperturas🡺Tipo *long*
* Tiempo de apertura 🡺Tipo *long*

Como ejemplo, ese fichero podría contener los valores:

0, 2000, 250🡺 Electroválvula 0, intervalo entre aperturas=2000, Tiempo de apertura=500

1, 800, 400🡺 Electroválvula 1, intervalo entre aperturas=800, Tiempo de apertura=400

Las 2 electroválvulas son independientes entre sí: Los intervalos entre aperturas y los tiempos de apertura de las mismas son completamente individuales.

Objetivos de este Ejercicio 3:

Objetivo 1: Leer el fichero ***Tiempos.bin*** con los tiempos de las 2 electroválvulas.

Objetivo 2: Una vez obtenidos esos tiempos, y como no se dispone del hardware necesario, se debe realizar una simulación del funcionamiento de las secuencias de apertura y cierre de las 2 electroválvulas. Esa simulación consiste en mostrar, **en estricto orden cronológico**, en el *OutPut* de NetBeans un mensaje indicando:

* Número de evento (1, 2, 3… etc)
* Número de electroválvula
* Evento producido (**Apertura** ó **cierre** de la electroválvula)
* Instante actual (En milisegundos)

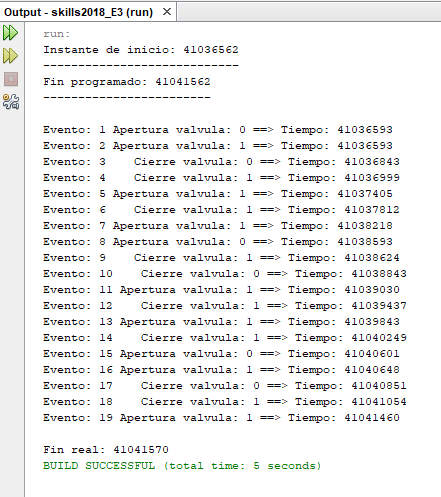
Para ello podría utilizarse la función System.out.println ():



**La simulación debe durar 5000 milisegundos**.

Los tiempos a utilizar son siempre en milisegundos y **están referidos a los milisegundos del día actual**. Así, las 02:15:00 horas corresponderían a 2x3600x1000 + 15\*60\*1000 milisegundos.

Si tuviésemos los datos del ejemplo anterior lo que debería aparecer sería algo así como:



Puede observarse que se muestran los eventos conforme se van produciendo y siempre referidos a los milisegundos actuales del día. Por tanto podría aparecer la apertura y el cierre de una electroválvula antes de que se produzca la apertura de la otra. Del mismo modo podría producirse la apertura de las 2 electroválvulas para, a continuación, producirse el cierre de las mismas en orden inverso a cómo se abrieron.

Junto al archivo ***Tiempos.bin*** se tiene otro archivo llamado ***Milisegundos.java*** que contiene una función *MiliSegundos\_De\_Hoy ()* que permite obtener los milisegundos actuales del día. Esa función devolverá 1 milisegundo en el primer instante del día y devolverá 86.399.999 milisegundos en el último instante del día. Para poder utilizar esa función es preciso importar las librerías de manejo de calendarios (java.util.Calendar):



La calificación de este Ejercicio 3 se especifica a continuación:

* **0.5** puntos si la simulación dura 5000 milisegundos. Se admite un error de 1% en el tiempo debido al retardo que provoca la función System.out.println ().
* **4.0** puntos si se consigue controlar de forma efectiva la apertura y cierre de las 2 electroválvulas según la temporización especificada en el fichero ***Tiempos.bin*** y durante los **5000** milisegundos especificados.

**ATENCIÓN**: No pueden utilizarse funciones que produzcan una pausa en la ejecución. Si se utilizase cualquier mecanismo (*pause* (), *sleep* (), *delay* (), bucles vacíos…etc.) que detuviese o ralentizase la ejecución del programa la calificación de este Ejercicio 3 sería:

**0.0**

# Esquema de puntuaciones

|  |  |
| --- | --- |
| Criterio | Valoración |
| Ejercicio 1: Obtención siguiente primo | 1.5 puntos |
| Ejercicio 1: Cálculos optimizados | 0.5 puntos |
|  |  |
| Ejercicio 2: Construcción matrices | 1.0 puntos |
| Ejercicio 2: Obtención nombre espía | 2.0 puntos |
| Ejercicio 2: No utilización bucles anidados | 0.5 puntos |
|  |  |
| Ejercicio 3: Duración de 5000 milisegundos | 0.5 puntos |
| Ejercicio 3: Simulación correcta de eventos | 4.0 puntos |