
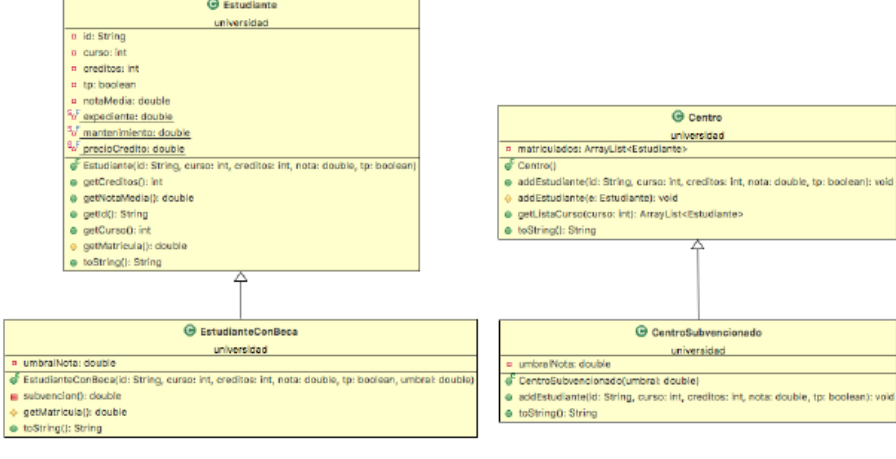


<div>  <div> DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN </div> </div> <div> PROGRAMACIÓN ORIENTADA A OBJETOS 9 de marzo de 2023 </div>	
<div> <div>APELLIDOS, Nombre</div> <div></div> </div>	<div> <div>TITULACIÓN Y GRUPO</div> <div></div> </div>
	<div> <div>MÁQUINA</div> <div></div> </div>

NORMAS PARA LA REALIZACIÓN DEL EXAMEN:

- Al inicio del contenido de cada fichero creado deberá aparecer un comentario con los datos del/de la estudiante (**nombre, titulación, grupo y código del equipo** que está utilizando).
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, *no hay que pararse en él indefinidamente*. Se pueden abordar otros.
- Está permitido:**
 - Consultar los apuntes (CV), la API (laboratorio), la guía rápida de la API (CV).
 - Añadir métodos privados a las clases.
- No está permitido:**
 - Utilizar dispositivos electrónicos (móviles, tablets, ...)
 - Utilizar otra documentación electrónica o impresa.
 - Intercambiar documentación con otros/as compañeros/as.
 - Utilizar soportes de almacenamiento.
 - Añadir métodos no privados a las clases.
 - Añadir variables o constantes a las clases.
 - Modificar la visibilidad de las variables, constantes y métodos que aparecen en el diagrama UML.
 - Modificar el código suministrado.
- Una vez terminado el examen, se debe **subir** (a la tarea creada en el CV para ello) un **fichero comprimido** (.zip o .rar) de la **carpeta src** que está en la carpeta **del proyecto** creado.
- Para la **corrección** del examen se utilizarán **programas de detección de copias/plagios**.
- Con **posterioridad** a la realización del examen, el profesorado podrá convocar a determinados/as alumnos/as para realizar **entrevistas personales** con objeto de comprobar la autoría de las soluciones

La Universidad ha solicitado el desarrollo de una aplicación para almacenar y gestionar la lista de estudiantes matriculados de cada centro. Se creará un proyecto llamado `prMatriculas`, en el que se realizará la implementación de las clases del siguiente diagrama, incluidas en el paquete `universidad`.



- (1 pts.)** La clase `Estudiante` contiene variables de instancia para almacenar los datos de un estudiante: un identificador único (`String`), el curso más alto del que está matriculado, el número de créditos matriculados (`int`), un atributo lógico (`boolean`) que indica si es

estudiante a tiempo parcial (`true`) o no (`false`) y la nota media del último curso (`double`). Además, la clase contiene tres constantes que almacenan, respectivamente, los gastos de apertura del expediente académico (59.0 euros, aplicables sólo en la matrícula del primer curso), los gastos de mantenimiento de la tarjeta de estudiante (5.7 euros) y el precio por crédito de docencia (22.5 euros). La clase deberá proporcionar:

- Un constructor al que se le proporciona el identificador, el curso, el número de créditos, la nota media y un valor lógico que indique la condición de estudiante a tiempo parcial. El constructor lanzará una excepción del tipo `RuntimeException` si no se cumplen las siguientes condiciones:
 - El curso es un número entre 1 y 4.
 - El número de créditos es mayor o igual que el mínimo fijado por la Universidad para un curso académico, y que se establece del siguiente modo: para los estudiantes a tiempo completo es 60 créditos en el primer curso y 48 en los restantes. Para los estudiantes a tiempo parcial es 24 en todos los cursos.

- Métodos de acceso a los atributos `getCreditos()`, `getNotaMedia()`, `getID()` y `getCurso()`.

- El método `getMatricula()`, que calcula y devuelve el importe de la matrícula del estudiante: *expediente + mantenimiento + creditos * precioCredito*, teniendo en cuenta que los gastos de apertura del expediente sólo se suman a los estudiantes de primer curso.

- El método `toString()`, que devuelve una representación textual del objeto con el formato que se muestra en el siguiente ejemplo, donde aparecen en **negrita** (por orden) los valores de los atributos *id*, *notaMedia*, *curso* y *créditos*. En azul aparece el importe de la matrícula del estudiante.

```
{388-77-5692 (media: 6.7); curso 4; 70 créditos; 1580.7 euros}
```

- (2 ptos.)** La clase `EstudianteConBeca` hereda de la clase `Estudiante` y representa a estudiantes que obtienen un descuento en su matrícula, cuya cuantía dependerá de sus notas. La clase tiene un atributo, `umbralNota`, cuyo valor indica la nota media mínima que ha debido sacar el estudiante en el curso anterior para obtener la beca. La clase deberá proporcionar:
 - Un constructor al que se le proporcionan un umbral de nota y todos los parámetros necesarios para crear un objeto `Estudiante`. Si la nota media del estudiante no supera el umbral, lanzará una excepción del tipo `RuntimeException`.

- La redefinición del método `getMatricula()` que devuelve la matrícula del estudiante descontándole el importe subvencionado, calculado por el método privado `subvencion()`.

- El método `subvencion()` devuelve la bonificación que se aplica a una matrícula. Para establecer cuál le corresponde al estudiante, se divide el rango de notas que van desde el umbral hasta 10 en tres tramos. El primero de ellos tiene una bonificación de 75 euros, el segundo de 125 y el tercero de 150. El estudiante obtendrá la subvención correspondiente al tramo en el que esté su nota. Por ejemplo, si el umbral es 8.5, los tramos serían [8.5 – 9.0] [9.0 – 9.5] y [9.5 -10.00], un estudiante con un 9.4 de nota media obtendría una bonificación de 125 euros (2º tramo), si su nota es de 8.75 la subvención sería en cambio de 75 euros (1er. tramo). Si el umbral bajase a 8.00, los tramos serían [8.0 – 8.66666666...]

[8.66666666...– 9.33333333...] y [9.33333333... - 10.00], ambos estudiantes obtendrían una bonificación mayor. El de 9.4 estaría ahora en el 3er tramo y tendría por tanto un descuento de 150 euros, mientras que el de 8.75 recaería en el 2º tramo y le corresponderían 125 euros.

- El método `toString()`, que devuelve una representación textual del objeto con el formato que se muestra en el siguiente ejemplo, donde aparecen en **negrita** (por orden) los valores de los atributos *id*, *notaMedia*, *umbralNota*, *curso* y *créditos*. En azul aparece el importe de la matrícula del estudiante.

```
{591-96-2640 (media: 9.9, umbral: 8.5); curso 3; 52 créditos; 1025.7 euros}
```

- (0.5 ptos.)** Crea una aplicación (clase distinguida `PruebaEstudiante`) para probar las clases anteriores. Realiza las siguientes acciones:
 - Crea un objeto `e1` de la clase `Estudiante` que represente a un estudiante a tiempo completo, con un identificador “785-36-6518”, de primer curso, con una nota media de 6.6 y 24 créditos. Muéstralo por pantalla. El resultado de la ejecución será:

```
Exception in thread "main" java.lang.RuntimeException: ERROR: estudiante 785-36-6518 no alcanza el mínimo número de créditos.
```

- Comenta las líneas anteriores y crea dos objetos `e2` y `e3` de la clase `Estudiante`. El primero será un estudiante a tiempo parcial con un identificador “718-15-9116”, de segundo curso, con una nota media de 6.5 y 24 créditos. El segundo será un estudiante a tiempo completo de primer curso, con una media de 8.9 y 66 créditos. Muestre ambos objetos por la consola. El resultado será:

```
{718-15-9116 (media: 6.5); curso 2; 24 créditos; 545.7 euros}
{366-35-7779 (media: 8.9); curso 1; 66 créditos; 1549.7 euros}
```

- Crea un objeto `e4` de la clase `EstudianteConBeca` con los mismos datos que `e3` y un umbral de nota de 8.0. Muéstralo por pantalla. El resultado será:

```
{366-35-7779 (media: 8.9, umbral: 8.0); curso 1; 66 créditos; 1424.7 euros}
```

- (4 ptos.)** La clase `Centro` almacena una lista de estudiantes y dispondrá de los siguientes métodos.

- Un constructor sin argumentos que crea una lista vacía.

- El método `addEstudiante(...)`, que recibe como parámetro un identificador, un curso, un número de créditos una nota y un valor lógico que indica si el estudiante es a tiempo parcial o no. Este método creará un objeto `Estudiante` con estos parámetros y llamará al método protegido `addEstudiante(Estudiante e)` que lo añadirá a la lista de matriculados.

- El método `addEstudiante(Estudiante e)`, que si ya hay un estudiante en el centro con el mismo id (sin diferenciar mayúsculas de minúscula) lo sustituye por el nuevo, y si no, lo añade al final de la lista.

- El método `getListaCurso(...)` que crea y devuelve una lista con los estudiantes que están matriculados en el curso que recibe como parámetro.

- El método `toString()` que devuelve la representación textual de la lista según el formato dado por la clase `ArrayList`.

- (2.5 ptos.)** La clase `CentroSubvencionado` hereda de la clase `Centro` y permite almacenar en la lista de matriculados estudiantes con y sin beca. Esta clase tiene un atributo que representa la nota mínima (`umbral`) que debe tener un estudiante para que se le subvencione parcialmente la matrícula.

- El constructor de la clase recibe como parámetro el umbral de la nota.

- La clase redefine el método `addEstudiante()`, de modo que si la nota recibida como parámetro es mayor que el umbral, se crea y se añade a la lista un objeto `EstudianteConBeca`, pasándole todos los parámetros (`id`, `curso`, `créditos`, `nota`) y el umbral del centro. En caso contrario se crea y añade un objeto de la clase `Estudiante`.

- En ambos casos, se invocará al método protegido `addEstudiante(Estudiante e)` para añadirlo a la lista de matriculados.

- El método `toString()` que una cadena con el mismo formato que la clase `Centro`, anteponiendo una línea que muestre la nota umbral a partir de las que se conceden becas (en **negrita**). Ejemplo:

```
Nota mínima para beca: 8.5
[{388-77-5692 (media: 6.7); curso 4; 70 créditos; 1580.7 euros},...,{...} ]
```

En el campus virtual se proporciona el fichero `PruebaCentro.java` para probar el funcionamiento de las clases `Centro` y `CentroSubvencionado`. La salida del programa será la siguiente (se han introducido saltos de línea en el enunciado para mejorar la legibilidad):

```
{388-77-5692 (media: 6.7); curso 4; 70 créditos; 1580.7 euros},
{718-15-9116 (media: 6.5); curso 2; 63 créditos; 1423.2 euros},
{820-94-1888 (media: 5.5); curso 3; 28 créditos; 635.7 euros},
{366-35-7779 (media: 8.9); curso 1; 66 créditos; 1549.7 euros},
{591-96-2640 (media: 9.9); curso 3; 52 créditos; 1175.7 euros},
{626-40-6133 (media: 5.1); curso 1; 66 créditos; 1549.7 euros},
{466-12-6029 (media: 7.2); curso 1; 70 créditos; 1639.7 euros},
{123-18-2748 (media: 4.2); curso 3; 53 créditos; 1198.2 euros},
{608-60-9964 (media: 7.5); curso 2; 43 créditos; 973.2 euros},
{652-92-2849 (media: 7.2); curso 3; 53 créditos; 1198.2 euros},
{701-73-6031 (media: 9.3); curso 4; 48 créditos; 1085.7 euros},
{269-58-1651 (media: 4.5); curso 1; 60 créditos; 1414.7 euros}]
```

```
Nota mínima para beca: 8.5
[{388-77-5692 (media: 6.7); curso 4; 70 créditos; 1580.7 euros},
{718-15-9116 (media: 6.5); curso 2; 63 créditos; 1423.2 euros},
{820-94-1888 (media: 5.5); curso 3; 28 créditos; 635.7 euros},
{366-35-7779 (media: 8.9, umbral: 8.5); curso 1; 66 créditos; 1474.7 euros},
{591-96-2640 (media: 9.9, umbral: 8.5); curso 3; 52 créditos; 1025.7 euros},
{626-40-6133 (media: 5.1); curso 1; 66 créditos; 1549.7 euros},
{466-12-6029 (media: 7.2); curso 1; 70 créditos; 1639.7 euros},
{123-18-2748 (media: 4.2); curso 3; 53 créditos; 1198.2 euros},
{608-60-9964 (media: 7.5); curso 2; 43 créditos; 973.2 euros},
{652-92-2849 (media: 7.2); curso 3; 53 créditos; 1198.2 euros},
{701-73-6031 (media: 9.3, umbral: 8.5); curso 4; 48 créditos; 960.7 euros},
{269-58-1651 (media: 4.5); curso 1; 60 créditos; 1414.7 euros}]
```