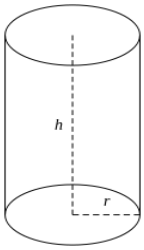
TAREA PRÁCTICA SOBRE LA UNIDAD 5 – PARTE (I): IMPLEMENTACIÓN DE LA CLASE DEPOSITO

La tarea consiste en la implementación de un **pequeño programa para simular el funcionamiento elemental de un conjunto de depósitos de estructura cilíndrica**. La aplicación permitirá crear objetos de tipo `Deposito` con unas características determinadas:



- **Radio de la base del depósito**.
- **Altura del depósito**.
- **Caudal de salida del grifo del depósito**.

Además, estos objetos de tipo `Deposito` contendrán **información de estado** como:

- El **nivel actual del depósito**.
- La **cantidad total de litros almacenados por el depósito desde que se fabricó**.
- La **cantidad total de litros vertidos por el depósito desde que se fabricó**.

Por otro lado, se pretende también que la propia clase, independientemente de la existencia o no de objetos, contenga **información global** del siguiente tipo:

- **Cantidad total de litros almacenados por todos los depósitos en el momento actual**.
- **Cantidad global de litros introducidos en todos los depósitos hasta el momento**.
- **Cantidad global de litros vertidos por todos los depósitos hasta el momento**.
- **Número de depósitos totalmente llenos en el momento actual**.
- **Número de depósitos totalmente vacíos en el momento actual**.

1. Clase `Deposito`.

Implementa la clase `Deposito` definiendo los atributos necesarios para representar las siguientes características de un depósito de estructura cilíndrica:

- Habrá tres primeros atributos que tendrán un valor que se definirá al crearse el objeto y ya nunca cambiarán su valor durante toda la vida del objeto (constantes). Podrían definirse algo así como la "naturaleza" o **características intrínsecas del objeto**:
  - **Altura del depósito** (en **metros**).
  - **Radio de la base del depósito** (en **metros**).
  - **Caudal de salida del grifo del depósito** (en **litros/segundo**).
- Otros atributos sí irán cambiando a lo largo de la vida del objeto y de alguna manera representarán el **estado del objeto** en cada momento:
  - El **nivel actual del depósito** (en **litros**).
  - La **cantidad total de litros almacenados** en el depósito desde que se fabricó (en **litros**).
  - La **cantidad total de litros vertidos** por el depósito desde que se fabricó (en **litros**).
- Por último, también habrá otros atributos que pertenecerán a la clase más que a un objeto en particular, y tendrán sentido siempre, independientemente de que existan o no instancias de la clase (**atributos estáticos**):
  - **Cantidad total actual de litros almacenados por todos los depósitos** que existan

en ese momento.
    - **Cantidad global de litros introducidos en todos los depósitos** hasta el momento.
    - **Cantidad global de litros vertidos por todos los depósitos** hasta el momento.
    - **Número de depósitos completamente llenos** en el momento actual.
    - **Número de depósitos completamente vacíos** en el momento actual.
- Además de esos atributos estáticos variables, la clase también va a contener **constantes** que serán de utilidad a la hora de realizar comprobaciones o asignaciones por defecto. Estas constantes serán públicas y tendrá acceso a ellas cualquier programador que utilice la clase:
    - **Mínima altura** permitida a la hora de crear un nuevo depósito (**0,20 metros**).
    - **Máxima altura** permitida a la hora de crear un nuevo depósito (**veinte metros**).
    - **Mínimo radio** permitido a la hora de crear un nuevo depósito (**0,20 metros**).
    - **Máximo radio** permitido a la hora de crear un nuevo depósito (**diez metros**).
    - **Máxima relación entre el radio y la altura**: **0,5**. Es decir, que el radio nunca podrá ser superior a la mitad de la altura. O lo que es lo mismo, la relación radio/altura debe ser siempre menor o igual que 0,5 (radio/altura<=0,5).
    - **Mínimo caudal de salida** permitido a la hora de crear un nuevo depósito (**0,001 litros/segundo**).
    - **Máximo caudal de salida** permitido a la hora de crear un nuevo depósito (**1,0 litros/segundo**).
    - **Valor por defecto para la altura** de un depósito (se utilizará en el constructor sin parámetros: **un metro**).
    - **Valor por defecto para el radio** de un depósito (se utilizará en el constructor sin parámetros: **medio metro**).
    - **Valor por defecto para el caudal de salida** del grifo de un depósito (se utilizará en el constructor sin parámetros: **0,100 litros/segundo**).

2. Constructores

Implementa **tres constructores** para la clase `Deposito`:

- Un constructor que recibirá como parámetros la **altura** (en **metros**), el **radio** (en **metros**) y el **caudal de salida** (en **litros/segundo**). Ahora bien, si los parámetros no son válidos (no cumplen las condiciones que se piden), entonces se lanzará una excepción de tipo `IllegalArgumentException` y el objeto no será creado. Las condiciones mínimas exigibles para que un objeto de tipo depósito pueda ser creado es que su **altura esté entre los veinte centímetros y los veinte metros**, el **radio entre los veinte centímetros y los diez metros** y además el **radio nunca pueda ser mayor que la mitad de la altura**. Respecto al **caudal de salida**, su rango válido se encontrará **entre un mililitro/segundo como mínimo y un litro/segundo como máximo**. Todas estas restricciones estarán definidas en las constantes de clase que se han descrito en apartados anteriores y **deben utilizarse esas constantes para las comprobaciones en el constructor, y no valores literales**. La excepción `IllegalArgumentException` lanzada deberá contener un mensaje de error diferente en caso de que:
    - El radio o la altura no estén dentro de los rangos permitidos (“`Error: Parámetros de creación del depósito inválidos. El radio o la altura no están en el rango permitido.`”).
    - El radio es mayor que la mitad de la altura (“`Error: Parámetros de creación del depósito inválidos. No cumplen los requisitos de geometría.`”).
    - El caudal del grifo de salida no está dentro del rango permitido (“`Error: Parámetros de creación del depósito inválidos. EL caudal`

`de salida no está en el rango permitido.`”).

- Un segundo constructor que permita crear un objeto de tipo `Deposito` donde la **altura** (en **metros**) y el **radio** (en **metros**) se le pasarán como parámetros. El **caudal de salida** será el valor por omisión de **cien mililitros/segundo** para el grifo de salida.
- Un constructor **sin parámetros** que creará una instancia de un depósito con una **altura de un metro** y un **radio de cincuenta centímetros**, así como un **caudal de salida de cien mililitros/segundo** para el grifo de vaciado. Esos valores deben aparecer como atributos constantes de la clase y es obligatorio usarlos en el constructor (no usar valores literales).
- Ten en cuenta que **desde los constructores con menos parámetros deberá invocarse el constructor con mayor parámetros** para completar la inicialización. Es fundamental que lo hagas así para **evitar escribir código redundante o incoherente**. Sólo en el tercer constructor deberían hacerse las **comprobaciones** y las **inicializaciones** apropiadas. De este modo se harán siempre de la misma manera

3. Métodos “consultores” (“getters”)

Añade a la clase `Deposito` los siguientes **métodos consultores** (**getters**) para poder obtener la siguiente información :

1. `getCapacidad()`, que **que devuelva la capacidad del depósito** `(en` **litros**`)`.
2. `getCaudalSalida()`, que devuelva el **caudal de salida del grifo del depósito** (en **litros/segundo**).
3. `estaVacio()`, que indique si el depósito se encuentra **completamente vacío**.
4. `estaLleno()`, que indique si el depósito se encuentra **completamente lleno**.
5. `getNivelActual()`, que devuelva el **nivel actual del depósito** (en **litros**).
6. `getVolumenIntroducidoDesdeCreacion()`, que devuelva la **cantidad de litros totales introducidos en el depósito desde su fabricación** (en **litros**).
7. `getVolumenVertidoDesdeCreacion()`, que devuelva la **cantidad de litros totales vertidos por el depósito desde su fabricación** (en **litros**).
8. `getVolumenGlobalActual()`, que devuelva la cantidad de **litros totales almacenados entre todos los depósitos en el momento actual** (en **litros**).
9. `getVolumenGlobalIntroducido()`, que devuelva la cantidad de **litros totales introducidos en todos los depósitos que hayan sido creados hasta el momento** (en **litros**).
10. `getVolumenGlobalVertido()`, que devuelva la cantidad de **litros totales vertidos por todos los depósitos que hayan sido creados hasta el momento** (en **litros**)
11. `getNumDepositosLlenos()`, que devuelva el **número de depósitos completamente llenos** en ese momento.
12. `getNumDepositosVacios()`, que devuelva el **número de depósitos completamente vacíos** en ese momento .

4. Método llenar

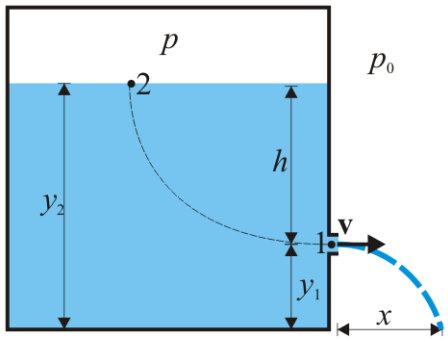
Implementa un método `llenar` que permita **rellenar el depósito** una cantidad de litros determinada que se pasará como parámetro. Este método **actualizará el atributo de nivel en función del valor del parámetro que se le pase**. Ahora bien, podría darse **un caso de error** que debería lanzar una **excepción** si **la cantidad de líquido con la que se intenta llenar es superior a la que puede admitir el depósito en ese momento**. En tal caso el depósito se llenará completamente, pero al rebosarse hará que se lance una excepción `IllegalArgumentException`. En este caso, el mensaje asociado a la excepción sería `"Error: Depósito lleno. Se ha desbordado en xxx litros."`, donde xxx indicaría la cantidad en exceso que se ha intentado rellenar y que no se ha podido (lo que habría rebosado).

Si el número de litros que se pasan al método es un número negativo, también se hará que se lance una excepción `IllegalArgumentException`. En este caso, el mensaje asociado a la excepción sería `"Error: Parámetro de cantidad de llenado inválido (litros negativos)."`

5. Método abrirGrifo

Implementa un método `abrirGrifo` que reciba como parámetro una cantidad de **segundos** y haga que el depósito vierta la cantidad de litros correspondiente al tiempo que se ha indicado como parámetro. Si el grifo está abierto el tiempo suficiente, el depósito se acabará vaciando completamente. El método debe **devolver la cantidad de litros** que se han vertido durante el tiempo que el grifo ha estado abierto. Si el número de segundos que se pasan al método es un número negativo, se hará que se lance una excepción `IllegalArgumentException`. En este caso, el mensaje asociado a la excepción sería `"Error: Parámetro de tiempo inválido (tiempo negativo)."`

6. Método vaciar



Implementa un método `vaciar`, que **vacíe completamente el depósito**. Este método debe calcular y **devolver la cantidad de segundos que se han necesitado para llevar a cabo el vaciado**

7. Método `toString`

Implementa un método `toString` que represente el **estado de un depósito**. El `String` final devuelto por el método debería:

1. Contener la siguiente **información del depósito** en ese momento:
    - **Capacidad del depósito** (en litros).
    - **Nivel actual del depósito** (en litros).
    - **Volumen almacenado histórico** desde que se construyó el objeto (en litros).
    - **Volumen vertido histórico** desde que se construyó el objeto (en litros).
2. Mostrar esa información con el siguiente formato: " **Capacidad: XXX litros - NivelActual: YYY litros - AlmacenadoTotal: ZZZ litros - VertidoTotal: WWW litros** ", donde XXX será la capacidad del depósito expresada en litros y con dos decimales; YYY será el nivel actual del depósito expresado en litros y con dos decimales; ZZZ será el volumen total almacenado desde que se construyó el depósito, expresado en litros con dos decimales, y WWW el volumen total vertido desde que se construyó el depósito, expresado en litros con dos decimales. Por ejemplo:

```
Capacidad:100,53 litros - NivelActual:38,00 litros- AlmacenadoTotal:50,00 litros- VertidoTotal:12,00 litros
```

8. Documentación javadoc

Documenta apropiadamente el código con **comentarios javadoc** para poder generar una documentación útil y apropiada:

Documentación a nivel de clase. Debes incluir una descripción lo más completa posible para documentar la clase en general.

- **Documentación de atributos públicos** indicando para cada atributo público una pequeña descripción.
- **Documentación a nivel de métodos**, incluyendo para cada método:
- **Descripción del método**.
- **Descripción del valor devuelto** (`@return`), si es que devuelve algo.
- **Descripción de cada uno de los parámetros** (`@param`).
- **Descripción de las posibles excepciones** que puede lanzar (`@throws`). En nuestro caso serán sólo de tipo `IllegalArgumentException`

Finalmente, tendrás que implementar algún programa principal que te permita probar todo lo que has hecho para saber si funciona correctamente.

CRITERIOS DE CORRECCIÓN DE LA TAREA

- Se tendrá en cuenta que todos los **identificadores** cumplan con el convenio sobre "**asignación de nombres a identificadores**" establecido para el lenguaje Java, para constantes, variables, métodos y clases. Se penalizará con **hasta 2 puntos** menos la no observación de este criterio.
- Cualquier **funcionalidad** pedida en el enunciado debe poderse probar y ha de funcionar correctamente, de acuerdo a las especificaciones del enunciado, incluido que **el proyecto completo pueda ser compilado y ejecutado**. Deberá utilizarse el programa principal que se proporciona. La no observación de este criterio podrá penalizarse con **hasta 5 puntos menos**, pues no se estará presentando el proyecto completo y operativo.
- La introducción de **datos erróneos o del tipo equivocado** no provocará nunca que la aplicación aborte, ya que todas las excepciones serán capturadas y tratadas convenientemente. Se penalizará con **hasta 3 puntos** menos la no observación de este criterio.
- Además, de los **comentarios javadoc**, se deben incluir todos **comentarios en el código** que se consideren oportunos para que éste quede autodocumentado. Un código pobremente documentado comentado no facilita su legibilidad. Se penalizará con **hasta 1 punto** un código con comentarios inapropiados, erróneos, imprecisos o inexistentes cuando se considere que debieran ser necesarios.
- Los **mensajes al usuario** deben ser siempre **claros, correctos ortográfica y gramaticalmente**, y ofrecer al mismo toda la información que le resulte relevante de forma completa. Se tendrá especialmente en cuenta que los comentarios tengan presente en todo momento que el usuario no está en el contexto del programador respecto al problema y a la solución, así que debemos preocuparnos de hacer explícito cuál es ese contexto. Por eso, debemos trabajar en la medida de lo posible asumiendo que el usuario no tiene por qué poseer conocimiento alguno sobre el problema, ni sobre la solución que aporta nuestra aplicación, salvo la información que le demos por medio de los mensajes. La **corrección ortográfica y gramatical**, así como la **coherencia en las expresiones lingüísticas**, tanto en los comentarios como en los mensajes al usuario es fundamental en todo programa. Se penalizarán con **hasta 2 puntos** menos las incorrecciones a este respecto.
- Es fundamental que **el código esté correctamente indentado**, se penalizará con **hasta 1 punto** la no contemplación de este criterio.

Rúbrica de corrección (Parte I):

La clase `Deposito` y sus atributos se han definido apropiadamente.	0,5
Los **constructores** de la clase `Deposito` se han implementado apropiadamente y funcionan correctamente.	1,0
Los métodos consultores (`getters`) de la clase `Deposito` se han implementado apropiadamente y funcionan correctamente.	0,5
El método `llenar` de la clase `Deposito` se ha implementado apropiadamente y funciona correctamente.	1,5
El método `abrirGrifo` de la clase `Deposito` se han implementado apropiadamente y funciona correctamente.	1,5
El método `vaciar` de la clase `Deposito` se ha implementado apropiadamente y funciona correctamente.	1,0
El método `toString` de la clase `Deposito` se ha implementado apropiadamente y funciona correctamente.	0,5
Se han incluido los comentarios `Javadoc` apropiados para la clase `Deposito` y la documentación se genera correctamente.	0,5

Defensa de la tarea

Los contenidos de esta tarea podrán ser contrastados por el profesorado mediante preguntas personalizadas y solicitud de modificaciones en el código para demostrar que el código entregado es fruto de un trabajo propio e individual. Si se observa que el alumnado no es capaz de defender su tarea como su legítimo autor, la tarea será evaluada negativamente.

Si durante la revisión y corrección posterior de la tarea se detectara algún indicio de copia, plagio, manipulación o cualquier otro tipo de actuación ilegítima, el alumnado implicado deberá defender de nuevo su tarea ante el profesorado.