Sesión L03- Abstracción y Modularización

01 Preparamos el directorio de trabajo

Para realizar esta práctica vamos a crear el directorio *L03-Abstraccion*, dentro de nuestro directorio de trabajo (Practicas). Por lo tanto tendremos que ir a dicho directorio, y crear el subdirectorio correspondiente a esta práctica:

```
> cd Practicas
> mkdir L03-Abstraccion
> cd L03-Abstraccion
```

Este será el subdirectorio en el que vamos a trabajar durante el resto de la sesión. Por lo tanto cualquier fichero o directorio que creemos en esta sesión estará dentro de *L03-Abstraccion.*

Copia las carpetas que os hemos proporcionado (en el directorio de plantillas: "***library-L03***" y ***mail-system-L03***") en tu zona de trabajo (en la subcarpeta **L03-Abstraccion**). Cada una de las carpetas contiene un proyecto Maven.

Crea el directorio"***diagramas-L03***" (tal y como mostramos en la imagen de la derecha), la cual usaremos en el ejercicio 3, y copia en ella el fichero ***tablas-L03-draw.io*** del directorio de plantillas.

Directorio de Trabajo

```
/home/pa/Practicas
├── .git
├── .gitignore
├── L01-Git
├── L02-Clases-Objetos
└── L03-Abstracion
    ├── library-L03
    ├── mail-system-L03
    └── diagramas-L03
        └── tablas-L03-draw.io
```

01 Uso de la herramienta draw.io

Cuando hablamos de orientación a objetos, el **estado** de un objeto en un instante determinado es el conjunto de valores de sus campos (también llamados **atributos,** propiedades o variables de instancia).

Los **atributos**, definidos en una clase, son comunes a cualquier objeto que pertenezca a dicha clase, aunque el estado de cada objeto (valor de sus variables) irá cambiando con el tiempo al invocar a los métodos correspondientes.

Los **métodos**, definidos en la clase a la que pertenece un objeto, determinan su **comportamiento.** La signatura de dicho comportamiento, dada por el nombre del método más sus parámetros y el tipo de dato que devuelve) nos indica "QUÉ" hace ese objeto, ocultando el "CÓMO", o lo que es lo mismo, ocultando los detalles de implementación.

Para definir una clase de forma abstracta, indicaremos sus atributos y sus métodos. Los métodos proporcionan **abstracción** a nuestra aplicación, ayudándonos a entender lo que hace dicha aplicación (qué), sin tener que mirar los detalles de código (cómo). Recuerda que la abstracción es sinónimo de "ocultar detalles". Cuanto más detalles ocultemos, el nivel de abstracción será mayor. Por ejemplo, un paquete tiene un nivel de abstracción mayor que una clase.

La **modularización**, por otro lado, nos permitirá dividir el problema en partes, de forma que tengamos diferentes niveles de abstracción: nivel de proyecto, nivel de paquetes, nivel de clases, nivel de objetos,... Si te fijas en los proyectos maven de esta semana y en los de la sesión anterior, verás que todos ellos contienen como mínimo un paquete (de forma que cualquier clase pertenece a algún paquete. Hablaremos de los paquetes con más detalle en las siguientes sesiones.

Usaremos diagramas de clases y diagramas de objetos para representar de forma abstracta los elementos del problema a resolver. El diagrama de clases representa una visión ESTÁTICA de nuestro programa, mientras que el diagrama de objetos representa una visión DINÁMICA.

Para crear los diagramas usaremos la herramienta on-line gratuita: www.draw.io (también os podéis descargar la versión de escritorio). Es muy sencilla de usar y dispone de muchos tipos de elementos que podemos usar en diferentes tipos de diagramas.

Para empezar a trabajar seleccionaremos: "Guardar en dispositivo", y crearemos un nuevo **diagrama en blanco**.

Para representar las **clases** puedes usar el elemento UML mostrado en la **Figura 1**.

Recuerda que el tipo del **campo** se especifica después del nombre de la variable seguido de dos puntos, y en el caso de los **métodos**, el tipo del valor que devuelve cada método se indica también al final precedido de dos puntos (si el método no devuelve nada usaremos "void").

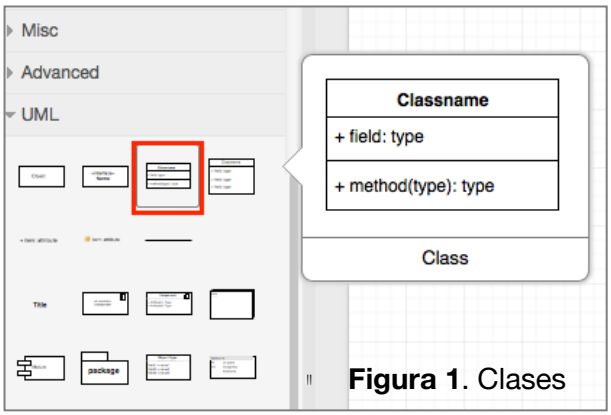


Figura 1. Clases

Tanto los campos como los métodos pueden ir precedidos por los símbolos “+” (si son “públicos”), o “-“, en el caso de que sean privados. Recuerda que de momento, los **campos** siempre van a ser “**privados**”, y por lo tanto sólo serán accesibles a través de los métodos *accesor* y *mutator* de la clase a la que pertenecen. Los **métodos** de la clase, normalmente serán

públicos (un método privado no es accesible desde fuera de la clase que lo contiene). De igual forma una clase pública puede ser accedida desde otro paquete.

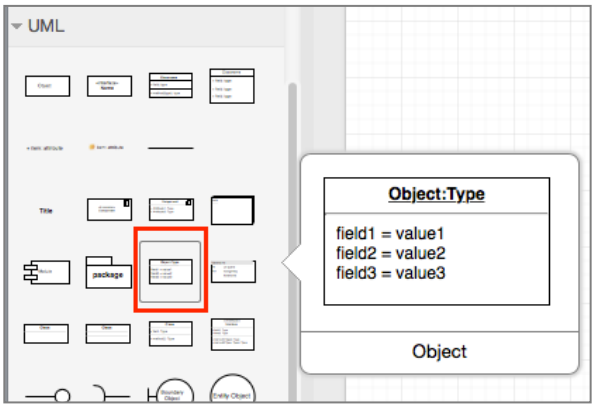


Figura 2. Objetos

Para representar los **objetos** puedes usar el elemento UML mostrado en la **Figura 2**. Por defecto, las figuras se representan con bordes rectos. Para que los objetos tengan un aspecto similar al que se propone en las transparencias, una vez que has creado el objeto, puedes modificar su apariencia y marcar la casilla "*rounded*" (desde la pestaña Style que se muestra en la parte derecha del dibujo, cuando se selecciona el objeto correspondiente). Ver **Figura 3**

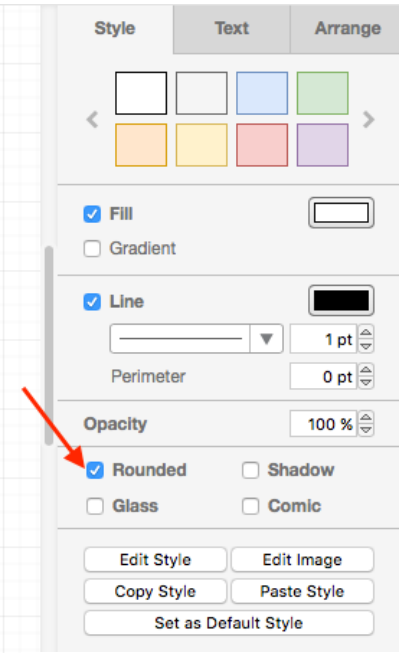


Figura 3. Aspecto de los objetos

Puedes cambiar el aspecto de de los elementos: colorear de forma diferente las clases y los objetos, cambiar el grosor de las líneas, poner sombras,...

Esta herramienta es interesante porque te puede servir para crear casi cualquier tipo de diagrama, por ejemplo, podemos representar diagramas de entidad/relación, usados en Base de Datos.

Con respecto a las **relaciones** entre clases y objetos, recuerda que para las clases usaremos la relación de dependencia etiquetada como "uses" y para los objetos una flecha (puedes elegir la que aparece con nombre "*association 3*")

Una vez que tengas completo el diagrama, la herramienta permite exportar el dibujo en formato png, o en pdf, entre otros, así como guardarla en Dropbox, Google Drive..., o en el disco duro del ordenador.

Deberás guardar cada uno de los diagramas creados en el disco duro, en formato xml (extensión ***.drawio***). desde la opción ***File → Save as...***), y también en formato png (opción ***File → Export as***). En el cuadro de texto ***Where***, debéis seleccionar ***Descargar***: os descargará el fichero en la carpeta de descargas.

02 Modularización y abstracción: diagramas de clases y objetos

Cuando crees tu proyecto java desde cero, tendrás que decidir cómo modularizar tu aplicación (cuántos paquetes y qué van a contener, y cuántas y qué clases vas a necesitar implementar). Y después tendrás que definir de forma abstracta cada una de las clases (determinar las propiedades y los comportamientos, indicando las signaturas de los constructores y métodos para cada una de las clases). Y finalmente implementarás los detalles de las clases. Fíjate que para resolver el problema SIEMPRE tenemos que hacerlo desde más nivel de abstracción a menos.

En nuestro código tendremos un conjunto de instancias de las clases (objetos), que irá cambiando a lo largo de la ejecución del mismo (es decir, en un instante determinado, durante la

ejecución del proyecto intervendrán un cierto número de objetos X, y en otro momento el número de objetos y/o su estado será diferente). Piensa en el ejemplo de las transparencias de obtener un listado de los pacientes de un hospital. La lista de pacientes no será la misma si la imprimimos hoy, o si la imprimimos el mes que viene (puede que haya más o menos pacientes, y que los pacientes sean diferentes).

Para modularizar nuestra aplicación, y definir de forma abstracta las clases, podemos usar diagramas de clases (que nos proporcionan una vista estática del proyecto) y diagramas de objetos (que constituyen una vista dinámica).

Ten en cuenta que en el diagrama de clases no sólo es importante reflejar todas ellas, sino también las **RELACIONES** entre ellas. De momento sólo conocemos una relación de dependencia entre clases, en concreto la relación ***use***. Debes tener claro para qué sirve y en qué consiste exactamente esa relación (es decir, por qué necesitamos incluir esa relación entre dos clases concretas.

La idea es que practiques con las definiciones abstractas de clases y objetos. Para este ejercicio debes usar la carpeta con nombre ***diagramas-L03***, dentro del directorio `Practicas/L03-Abstraccion`, en donde guardarás los siguientes diagramas de clases y objetos, que deberás crear con la herramienta ***draw.io***:

- Crea un diagrama de clases y un diagrama de objetos para el proyecto ***laboratory-L02*** de la práctica anterior (ejercicio 4). Guarda el diagrama de clases con el nombre ***lab-clases.drawio*** (con la opción *Save as*), y el diagrama de objetos en un fichero con el nombre ***lab-objetos.drawio***. Exporta los diagramas en formato **png** con los nombres ***lab-clases.png*** y ***lab-objetos.png***, respectivamente (usa la opción *Export as*).

  Para realizar el diagrama de objetos debes tener en cuenta que queremos obtener dicho diagrama después de ejecutar el método ***pa.Enrollment.main()***. En este caso, la propiedad *students* del objeto *labPA*, contiene una colección de objetos (hablaremos de las colecciones de objetos en sesiones posteriores). Puedes indicar el valor de dicha propiedad en el diagrama de objetos como: ***students = <st1, st2, st3>***.

- Crea un diagrama de clases y un diagrama de objetos para el proyecto ***mail-system-L03*** proporcionado. Guarda los diagramas en los ficheros ***mail-clases.drawio*** y ***mail-objetos.drawio*** (con la opción *Save as*). Exporta los diagramas en formato **png** (con los nombres ***mail-clases.png*** y ***mail-objetos.png*** (con la opción *Export as*).

  Para realizar el diagrama de objetos debes tener en cuenta únicamente el código del método ***pa.DemoMail.main()***. En este caso, el campo *items* del objeto *servidorDeCorreo* contiene una colección de objetos. Puedes indicarlo en el diagrama de objetos como en el caso anterior.

Sube a GitHub el trabajo realizado hasta el momento:

```
> git add .   //desde el directorio que contiene el repositorio git
> git commit -m"Terminados los diagramas de L03"
> git push
```

03 Detalles de las clases: implementación en Java

Importa el proyecto Maven ***library-L03*** en Eclipse (desde tu directorio *L03-Abstraccion*, recuerda que para importar el proyecto maven debes seleccionar la carpeta que contiene el fichero *pom.xml*). Se pide:

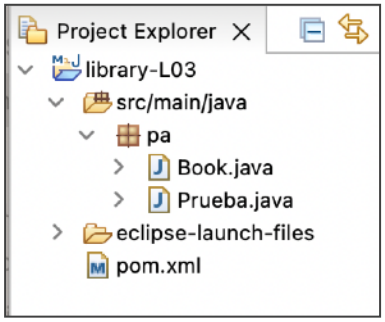
- Añade los métodos ***getAuthor()*** y ***getTitle()*** a la clase *Book* de forma que devuelvan el valor del campo *author* y *title,* respectivamente. Recuerda que este tipo de métodos se denominan métodos *accessor* o también métodos *getter.*

- Añade un campo ***pages*** a la clase *Book* para almacenar el número de páginas del libro. Debería ser de tipo ***int***. Añade un parámetro adicional en el constructor que represente el número de páginas. Añade también el *accesor* correspondiente al nuevo campo *pages*.
- Añade un campo adicional a la clase con el nombre ***refNumber***. Este campo almacenará el número de referencia del libro (es un identificador único para cada libro). Debería ser de tipo ***String***. Inicialízalo con una cadena de caracteres vacía (""), ya que su valor inicial no va a ser pasado como parámetro en el constructor de la clase.
- Define un método *mutator* para el campo ***refNumber*** (recuerda que estos métodos los utilizamos para cambiar el valor de alguna propiedad del objeto, en este caso, si dicho método únicamente cambia el valor del atributo, también se conoce como método *setter*). La signatura del método será: `public void setRefNumber(String ref)` de forma que se asigne el valor que se pasa por parámetro al campo *refNumber* de la clase. El método ***setRefNumber()*** solamente debe asignar el número de referencia pasado por parámetro si éste es una cadena con 3 caracteres como mínimo. Si el valor que se pasa por parámetro tiene menos de tres caracteres, entonces se debe imprimir un mensaje de error y dejar el campo sin modificar. El mensaje de error tendrá la forma `"ERROR: El número de referencia del libro <titulo> debe contener al menos tres caracteres"`. En donde `<titulo>` tendrás que reemplazarlo por el título del libro cuyo número de referencia sea incorrecto.
- Crea un método accessor para poder consultar el valor de la propiedad ***refNumber*** (**recuerda**: propiedad, campo y variable de instancia son sinónimos).
- Añade un nuevo método público ***printDetails()*** a la clase. Este método debe imprimir los detalles del autor, título, número de páginas y número de referencia en una ventana del terminal. Tienes que tener en cuenta que solamente mostraremos el número de referencia si la longitud de la cadena no es cero (utiliza el método *length()* aplicable a los objetos de tipo *String* para averiguar la longitud de la cadena). En caso de que todavía no hayamos asignado un número de referencia al libro, se deberá imprimir como número de referencia la cadena "ZZZ" en su lugar. Puedes formatear el mensaje como consideres oportuno utilizando sentencias ***System.out.println***.
- Añade un campo adicional con el nombre ***borrowed*** a la clase. Este campo contabilizará el número de veces que ha sido prestado el libro. Añade el método ***prestar()***, de forma que se incremente en 1 el número de préstamos cada vez que se invoque a dicho método. Implementa también su *accesor* con el nombre ***getBorrowed()*** de forma que devuelva el valor del campo correspondiente. Modifica finalmente el método ***printDetails()*** para que imprima también el valor de este campo en los detalles del libro.
- Añade un nuevo método público ***mostrarVecesPrestado()***. Dicho método mostrará por pantalla un mensaje (ver **Figura 4** en la última hoja) con el título de libro indicando el número de veces que se ha prestado (el mensaje estará en singular o en plural dependiendo de si se ha prestado una vez o más de una). En la Figura 4, al final del ejemplo, verás cómo debe ser dicho mensaje para cada caso.

Para probar los cambios realizados en la clase *Book*, vamos a **crear una nueva clase** a la que llamaremos **Prueba** (en el paquete ***pa*****).** Para crear la nueva clase seleccionaremos el paquete **pa** del proyecto (pulsando una vez con el botón izquierdo del ratón sobre el elemento *pa*). Una vez seleccionado, pulsamos el botón derecho del ratón para mostrar el menú contextual, y seleccionamos ***New→Java Class...*** Modifica únicamente el nombre de la clase y llámala Prueba, el resto de campos del formulario déjalos como están. A continuación crea un método ***main()*** en la clase Prueba (recuerda que la signatura del método tiene que ser: `public static void main()`. Este método implementará lo siguiente:

- **Crea** tres libros, cuyos títulos son: "Juego de Tronos", "El nombre de la Rosa", y "Tutankamon", respectivamente. Los autores son, en este orden: "George R.R. Martin", "Umberto Eco", y "Christian Jacq". Finalmente el número de páginas para cada uno de ellos, son 1300, 987 y 876, respectivamente.
- Imprime por pantalla el mensaje: "Creados los tres libros" (utiliza la sentencia *System.out.println()* desde el main).
- **Añade** un número de referencia a cada libro (usando el *mutator* correspondiente). En concreto, serán "01", "001" y "0003", para el primero, segundo y tercer libro, respectivamente. A continuación **imprime** los detalles de cada libro.
- Ahora **cambia** los números de referencia del primer y segundo libro (usando el *mutator* correspondiente) y sustitúyelos por "0001", "0002". Vuelve a **imprimir** los detalles de los dos primeros libros.
- Solicita dos veces un **préstamo** del primer libro, y una vez para el segundo libro.
- Muestra por pantalla las veces que se ha prestado cada libro (usando el método que has implementado)
- Ejecuta el programa. Debes ver en pantalla la información mostrada en la **Figura 4** (al final del ejercicio).

A continuación mostramos los diferentes **niveles de abstracción** así como los **módulos** en los que hemos dividido nuestro proyecto maven:



Tenemos **3 niveles de abstracción**, ordenados de mayor a menor:
PROYECTO: (library-L03) →mayor abstracción = menos nivel de detalle
PAQUETE: (pa)
CLASE: (Book, Prueba) →menor abstracción = más nivel de detalle

Fíjate que **cada nivel** tiene una **modularización** diferente.

A Nivel de PROYECTO: tenemos **1 módulo**: library-L03
A nivel de PAQUETE: tenemos **1 módulo**: pa
A nivel de CLASE: tenemos **dos módulos**: Book, Prueba

Para ejecutar el programa puedes usar la Run configuration: ***library-L03-run***. También dispones de la run configuration ***library-L03-clean*** (para eliminar el directorio *target*).

Sube a GitHub el trabajo realizado hasta el momento:

```
> git add .   //desde el directorio que contiene el repositorio git
> git commit –m”Terminados los ejercicios de proyecto library de L03”
> git push
```

Figura 4. Traza de ejecución del ejercicio 3. Recuerda que en rojo mostramos las salidas por pantalla de las sentencias *System.out.println()* que debes usar en el método *main()*.

```
Creados los tres libros

ERROR: El número de referencia del libro Juego de Tronos debe contener al menos
3 caracteres

Detalles de Juego de Tronos
--------------------------
   -- Autor: George R.R. Martin
   -- Título: Juego de Tronos
   -- Nº páginas: 1300
   -- Nº referencia: ZZZ
   -- Nº préstamos: 0
--------------------------


Detalles de El nombre de la rosa
--------------------------
   -- Autor: Umberto Eco
   -- Título: El nombre de la rosa
   -- Nº páginas: 987
   -- Nº referencia: 001
   -- Nº préstamos: 0
--------------------------


Detalles de Tutankamon
--------------------------
   -- Autor: Chistian Jacq
   -- Título: Tutankamon
   -- Nº páginas: 876
   -- Nº referencia: 0002
   -- Nº préstamos: 0
--------------------------

Cambiamos las referencias de los dos primeros libros

Detalles de Juego de Tronos
--------------------------
   -- Autor: George R.R. Martin
   -- Título: Juego de Tronos
   -- Nº páginas: 1300
   -- Nº referencia: 0001
   -- Nº préstamos: 0
--------------------------


Detalles de El nombre de la rosa
--------------------------
   -- Autor: Umberto Eco
   -- Título: El nombre de la rosa
   -- Nº páginas: 987
   -- Nº referencia: 0002
   -- Nº préstamos: 0
--------------------------

El libro Juego de Tronos se ha prestado 2 veces
El libro El nombre de la rosa se ha prestado 1 vez
El libro Tutankamon no se ha prestado ninguna vez
```

Se muestra ZZZ

Las nuevas referencias tienen 4 caracteres

mostramos el número de préstamos realizados

04 Tipos de variables

Edita el fichero ***tablas-L03.drawio*** proporcionado. Verás que contiene tres tablas: una para cada tipo de variable del proyecto maven *library-L03*. Rellénalas y añade todas las filas que necesites.

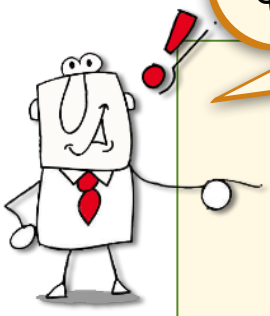
Después de rellenar las tablas, guarda el fichero, y a continuación expórtalo en **formato *png***.

Recuerda que las variables de clase las hemos mencionado en clase de teoría, pero las explicaremos más adelante.

Sube a GitHub el trabajo realizado hasta el momento.

RECUERDA!

¿Qué conceptos y cuestiones me deben quedar CLAROS después de hacer la práctica?



- Cuántos niveles de abstracción tienen cada uno de mis proyectos de prácticas
- Cuál es la descripción abstracta a nivel de clase de todas las clases de mis prácticas
- Cuántos módulos forman cada uno de los niveles de abstracción de mis prácticas.
- A la hora de resolver un problema, cómo debemos proceder: de más nivel de abstracción a menos o al revés. ¿por qué?
- Qué representan un diagrama de clases y un diagrama de objetos, y diferencias entre ambas vistas. ¿Puedo representar un diagrama de objetos sin tener en cuenta el tiempo?
- IMPORTANTE!! Cuando dibujemos un diagrama de clases, es imprescindible, además de usar el tipo de flecha adecuado, indicar el TIPO DE RELACIÓN entre las clases (de momento sólo conocemos la relación USE, pero más adelante introduciremos otros tipos de relaciones)
- Un diagrama de clases, al tener un carácter estático, tendremos que hacerlo ANTES de implementar, mientras que un diagrama de objetos depende de la implementación, por lo que lo usaremos para ilustrar el comportamiento de nuestro programa en un instante de tiempo determinado.
- Cómo puedo saber, viendo el código, si estoy invocando a un método interno (de la propia clase) o externo (de otra clase).
- Para qué sirve la palabra reservada "this"
- Cómo crear nuevas clases con Eclipse
- Para qué sirve el método main() de una clase
- Qué implica que un campo, método y/o clase sean públicos o privados
- Qué diferencia hay entre una variable de instancia, una variable local y un parámetro. Tienes que ser capaz de identificarlas en el código.
- ¿Cómo puedes reconocer un tipo primitivo de un tipo objeto en el código. ¿Cómo se almacenan las variables de tipos primitivos y de tipos objeto?
- Los parámetros de un método java siempre se pasan por valor. ¿Qué pasa cuando un parámetro es de tipo primitivo o de tipo objeto cuando se modifica el valor del parámetro en el código de dicho método?