//# Este compendio estará compuesto: de los resúmenes al inicio de cada ejercicio de la asignatura de Acceso a Datos y de la correspondiente tarea que será adjuntada en un pastebin para mayor facilidad de lectura. Cualquier duda o anotación deberá ser dirigida hacia la autora de este documento.

➔ Ejercicio #1:

Resumen:

- Veremos una introducción breve a Java.
- Comprenderemos cómo está montado el espacio de trabajo.
- Hablaremos de las clases y métodos en Java.
- Crearemos una clase `Greeter` con un método `sayHello`.
- Probaremos el *test* automático de la tarea.
- Distinguiremos *ejecutar* `Main.java` y *ejecutar tests*.
- Haremos `git add`, `git commit` y `git push` para subir los cambios al repositorio.

Tarea:

Esta tarea no requiere de código alguno, tan solo se debe de ejecutar el test correspondiente y entregar.

➔ Ejercicio #2:

Resumen:

- Escribiremos una nueva clase con un nuevo método sencillo.
- Lo invocaremos como hicimos con `Greeter` y `.sayHello()`.
- *(Y como siempre debemos hacer, subiremos los cambios al repositorio)*.

Tarea:

https://pastebin.com/5eStWfzN

➔ Ejercicio #3:

Resumen:

- Escribiremos una nueva clase con un nuevo método sencillo.
- Lo invocaremos como hicimos con `Greeter` y `.sayHello()`.
- *(Y como siempre debemos hacer, subiremos los cambios al repositorio)*.

Tarea:

https://pastebin.com/sJBm3y1U

➔ Ejercicio #4:

Resumen:

- Distinguiremos entre función y procedimiento.
- Implementaremos un método `byeWorld` en nuestra clase `Greeter`, que devolverá un String.
- Invocaremos dicho método desde `Main`.

Tarea:

https://pastebin.com/5RfYA7zs

➔ Ejercicio #5:

Resumen:

- Escribiremos **3** métodos distintos en `Greeter`.
    - Dos de ellos devolverán un número entero.
    - El tercero *invocará* a los otros dos y sumará esos valores.
- Veremos que un método puede invocarse desde *otro método*.
- Invocaremos dicho método desde `Main`.

Tarea:

https://pastebin.com/enS7DWTL

➔ Ejercicio #6:

Resumen:

- Escribiremos una nueva clase `SimpleMathDemo`.
- Escribiremos **4** métodos, similares a la tarea anterior.
    - Tres de ellos devolverán un número entero.
    - El cuarto *invocará* a los otros tres y sumará esos valores.
- Invocaremos dicho método desde `Main`.

Tarea:

https://pastebin.com/hDcCeqhv

➔ Ejercicio #7:

Resumen:

- Veremos ejemplos de *Unreachable statement*.
- Repasaremos brevemente la sintaxis del bucle `for` simple.
- Escribiremos un método en `SimpleMathDemo` que tiene un `return` dentro de un bucle.
- Veremos cómo se comporta invocándolo desde `Main`.

Tarea:

https://pastebin.com/8ZiK35np

➔ Ejercicio #8:

Resumen:

- Entenderemos que los métodos admiten *parámetros*.
- Veremos cómo se declaran los *parámetros formales*.
- Veremos cómo se envían (al invocar el método) los *parámetros actuales*.
- Escribiremos un ejemplo en `SimpleMathDemo` y lo invocaremos desde `Main.java`.

Tarea:

https://pastebin.com/hay3dmqh

➔ Ejercicio #9:

Resumen:

- Veremos brevemente cómo un método puede recibir más de 1 parámetro.
- Escribiremos un método en `SimpleMathDemo` que *recibe* 2 números enteros y los multiplica.

Tarea:

https://pastebin.com/cLUG2ZJL

➔ Ejercicio #10:

Resumen:

- Implementaremos otro método `printProduct` que reciba *distintos* parámetros, de *distintos* tipos.
- Comprenderemos el concepto de *sobrecarga* (`overload`).

Tarea:

https://pastebin.com/Q7UXfXHL

➔ Ejercicio #11:

Resumen:

- Un pequeño comentario sobre `Main`.

Tarea:

Esta tarea no requiere de código complicado, solo la creación de otra clase y el borrado de todo lo que había en el interior del Main.java. https://pastebin.com/EeuZ2dtE

➔ Ejercicio #12:

Resumen:

- Escribiremos un método nuevo en `SimpleMathDemo` que reciba **3** valores enteros y *devuelva* el mayor de ellos.

Tarea:

https://pastebin.com/wDnPXSRE

➔ Ejercicio #13:

Resumen:

- Escribiremos un método nuevo en `SimpleMathDemo` que reciba **4** valores enteros y *devuelva* el menor de ellos

Tarea:

https://pastebin.com/0hKGAmjJ

➔ Ejercicio #14:

Resumen:

- Escribiremos un método nuevo en `SimpleMathDemo` que reciba **2** valores enteros, `start` y `length`.
- Devolverá un número entero **aleatorio** en el rango definido por esos valores.

Tarea:

https://pastebin.com/NVs0k9nA

➔ Ejercicio #15:

Resumen:

- Hablaremos de la *instanciación* de una clase.
- Comprenderemos qué es el método **constructor**.
- Crearemos una nueva clase `Person` con un método **constructor** escrito por nosotros, que admita un parámetro.

Tarea:

https://pastebin.com/EJPZT3W5

➔ Ejercicio #16:

Resumen:

- Introduciremos los *atributos*.
- Añadiremos un *atributo* `name` a nuestra clase `Person`.
- Lo inicializaremos en el método **constructor**.

Tarea:

https://pastebin.com/AzRHsBXa

➔ Ejercicio #17:

Resumen:

- Hablaremos de métodos `getters` y `setters`.
- Añadiremos un método `getter` a nuestra clase `Person`.
- Añadiremos un método `greeting()` que devuelva un `String` compuesto.

Tarea:

https://pastebin.com/WAdQXmVx

➔ Ejercicio #18:

Resumen:

- Introduciremos el concepto de POJO.
- Crearemos un POJO llamado `Animal`, con la agilidad que nos otorga IntelliJ IDEA.

Tarea:

https://pastebin.com/ztyBBUDG

➔ Ejercicio #19:

Resumen:

- Crearemos una clase `Coin` que represente una moneda.
- Tendrá un atributo que represente la **recompensa** por *ganar una tirada*.
- Tendrá un método que hará un *print* y represente una tirada aleatoria.

Tarea:

https://pastebin.com/4VVraZtQ

➔ Ejercicio #20:

Resumen:

- Crearemos una clase Mouse que simbolice el ratón del ordenador.
- A través de un atributo, podrá estar configurada para zurdos.
- Tendrá dos métodos que simbolizan los clicks principal y secundario.
    - En función de la configuración harán dos print distintos.

Tarea:

https://pastebin.com/UCjkWs0z

➔ Ejercicio #21:

Resumen:

- Repasaremos el concepto de *array*.
- Repasaremos la sintaxis para declarar e inicializar un *array* en Java.
- Crearemos una clase `TravelStops`:
    - Almacenará en un atributo privado un *array*.
    - Tendrá tres métodos para imprimir sus valores.

Tarea:

https://pastebin.com/DGSaQuuk

➔ Ejercicio #22:

Resumen:

- Añadiremos un método para imprimir el *enésimo* elemento de `TravelStops`.

Tarea:

https://pastebin.com/beA3JUg9

➔ Ejercicio #23:

Resumen:

- Añadiremos un método para imprimir *todas* las paradas de `TravelStops`.

Tarea:

https://pastebin.com/W5mnbGx6

➔ Ejercicio #24:

Resumen:

- Añadiremos un método para *actualizar* (modificar) una parada de `TravelStops`.

Tarea:

https://pastebin.com/qnXNqs5k

➔ Ejercicio #25:

Resumen:

- Hablaremos de `ArrayList`.
- Veremos algunas de sus ventajas.
- Implementaremos un `NewTravelStops`, esta vez usando un `ArrayList`.

Tarea:

https://pastebin.com/Rtr8f0zp

➔ Ejercicio #26:

Resumen:

- Introduciremos el bucle *for-each*.
- Implementaremos un método en `NewTravelStops` que imprima todas las paradas.

Tarea:

https://pastebin.com/muqG7raF

➔ Ejercicio #27:

Resumen:

- Hablaremos del árbol de directorios del sistema operativo.
- Veremos cómo leer un fichero línea a línea en Java.
- Hablaremos del carácter de escapado (`\`).
- Crearemos una clase `HomeCinemaPreferences` que tendrá un método para leer el fichero `assets\cinemaPrefs.txt` e imprimir su contenido por pantalla.

Tarea:

https://pastebin.com/k21tsccv

➔ Ejercicio #28:

Resumen:

- Añadiremos dos atributos privados y dos `getters` a `HomeCinemaPreferences`.
- Escribiremos un nuevo método privado que *parsee* una línea del fichero.
- Usaremos el método para *parsear* todas las líneas del fichero.

Tarea:

https://pastebin.com/F70A0QGx

➔ Ejercicio #29:

Resumen:

- Añadiremos dos métodos `setters` a `HomeCinemaPreferences`.
- Escribiremos un nuevo método escribe las preferencias en el fichero `cinemaPrefs.txt`.

Tarea:

https://pastebin.com/Y61MQw5J

➔ Ejercicio #30:

Resumen:

- Echaremos un vistazo al formato XML.
- Añadiremos un método `saveExampleXML` a nuestra clase `HomeCinemaPreferences.java` que creará y guardará un XML de ejemplo (`assets\example.xml`), con una etiqueta raíz y dos nodos.
- Añadiremos `saveAsXML`, que guardará un XML (`cinemaPrefs.xml`) con los valores de las preferencias.

Tarea:

https://pastebin.com/3vZtUf2Z

➔ Ejercicio #31:

Resumen:

- Veremos cómo se puede recuperar la información de un archivo XML.
- Crearemos un nuevo constructor `HomeCinemaPreferences` que reciba un parámetro `boolean`:
    - Si es `false`, el método funcionará como hasta ahora, leyendo `cinemaPrefs.txt`.
    - Si es `true`, leerá `cinemaPrefs.xml`.
- Hablaremos de *deprecar* (marcar como obsoleto) código.

Tarea:

https://pastebin.com/TtxdzYXm

➔ Ejercicio #32:

Resumen:

- Echaremos un vistazo al formato JSON.
- Añadiremos un método `saveExampleJSON` a nuestra clase `HomeCinemaPreferences.java` que creará y guardará un JSON de ejemplo (`assets\example.json`), con dos atributos.
- Añadiremos `saveAsJSON`, que guardará un JSON (`cinemaPrefs.json`) con los valores de las preferencias.

Tarea:

https://pastebin.com/0JVAXiaw

➔ Ejercicio #33:

Resumen:

- Veremos cómo se puede recuperar la información de un archivo JSON.
- Veremos qué es un `Enum` en Java.
- Crearemos un nuevo constructor `HomeCinemaPreferences` que reciba un parámetro `Enum`:
    - Permitirá inicializar las preferencias desde `cinemaPrefs.txt`, `cinemaPrefs.xml` ó `cinemaPrefs.json` según su valor.
- *Deprecaremos* los constructores anteriores.

Tarea:

https://pastebin.com/phWhDFGX

➔ Ejercicio #34:

Resumen:

- Veremos cómo se representa una lista en JSON.
- Crearemos una clase sencilla con un método que sólo lee e imprime una lista de *strings* JSON.

Tarea:

https://pastebin.com/dH3Frc8v

➔ Ejercicio #35:

Resumen:

- Se pedirá una clase que tenga un método constructor y un método público que almacene un JSON con información de la repartición inicial de cartas de una baraja de naipes para `N` jugadores.

Tarea:

https://pastebin.com/pjgMAxDy

➔ Ejercicio #36:

Resumen:

- Se pedirá una clase que permita almacenar información de registro horario (hora de entrada y hora de salida) con un método, y, generar un JSON con la información acumulada.

Tarea:

https://pastebin.com/DzzVKaG9