Ejercicios Curso Java

Formación Sopra Steria
Formación JAVA

Bloque Java Básico

Versión 1.2 del lunes, 9 de julio de 2018

Destinatario(s)

Historial

Versión	Fecha	Origen de la actualización	Redactado por	Validado por
1.0	18/06/2018		María Nadal Silvia Carpena	
1.1	27/06/2018	Capítulo 8	María Nadal Silvia Carpena	
1.2	09/07/2018	Actualización expresiones lambda	Silvia Carpena	

Índice

1. Capítulo 3: Bloques básicos en Java

1. **PAQUETES** – Se deberá crear un paquete llamado com.sopra.javabasico.capitulo3.
2. **CREACIÓN DE CLASE** – Se deberá crear una clase llamada Capitulo3.java en com.sopra.javabasico.capitulo3.
3. **DOCUMENTACIÓN MEDIANTE JAVADOC** – Se documentarán la clase, métodos, constructores y distintas variables utilizadas mediante comentarios de Javadoc para poder crear la documentación del código.
4. **MAIN** – Se implementará en la clase anterior el método *main* que muestre por pantalla la frase "Estoy aprendiendo a programar en Java".
5. **ARGUMENTOS** – Se implementará en la clase anterior una funcionalidad de que muestre por pantalla el argumento que se haya recibido: "Me has mandado el texto: …".
6. **CREACIÓN DE OBJETOS e IMPORTACIONES** – Se deberá crear un objeto de la clase Random (del paquete java.util) e implementar una funcionalidad que imprima por pantalla un número aleatorio del 0 al 9. ¿Qué sucedería al importar, además, el paquete java.util utilizando el carácter comodín, * (asterisco)?
7. **VARIABLES DE CLASE** – Se deberán crear cinco variables de clase en la clase Capitulo3.java. Dos de ellas deben ser primitivas de datos a elección, otra, debe tener el tipo de dato Random; y las otras dos variables deben ser estáticas y de tipo String (una de ellas debe inicializarse con el valor "Esto es un String").
8. **CONSTRUCTORES** – Se deberá crear un constructor público para la clase Capitulo3.java que inicialice los tres primeros atributos creados en el punto anterior.
9. **BLOQUES INICIALIZADORES DE INSTANCIAS** – Se crearán dos bloques inicializadores de instancias. El primero, al final de la clase, deberá mostrar por pantalla el mensaje "Este bloque es un inicializador de instancia" y el segundo, al principio de la clase, deberá imprimir por pantalla la variable *num*. ¿Qué ocurre con cada uno de los bloques?
10. **CREACIÓN DE OBJETOS** – Se deberá crear un objeto de la clase Capitulo3 en el método *main*, pasando como parámetros los valores que se desee. (Si uno de los argumentos del constructor de la clase Capitulo3 es de tipo numérico, deberá introducirse en modo octal o hexadecimal).
11. Se deberán mostrar por pantalla (dentro del método *main*) los valores de los dos String creados y de los tres atributos de la clase. Para el atributo de tipo Random, se mostrará por pantalla un número aleatorio del 0 al 5.
12. **VARIABLES LOCALES** – Se deberá crear una variable local dentro del constructor de la clase e inicializarla. ¿Es posible acceder a esta variable fuera del constructor?
13. **ORDENANDO ELEMENTOS EN UNA CLASE** – Llegados a este punto, sería adecuado echar un vistazo al orden de los elementos de la clase, ¿sigue el formato explicado en la teoría del Capítulo 3?
14. **DESTRUYENDO OBJETOS** – Por último, se crearán dos objetos Capitulo3 en el método *main*, y después, se asignará el valor null a ambos. ¿Algún objeto ha sido recogido por el *Garbage Collector*? Es necesario asegurarse de ello haciendo uso del método finalize()

2. Capítulo 4: Operadores y sentencias

1. Se deberá crear un paquete llamado com.sopra.javabasico.capitulo4. Para cada uno de los ejercicios que hay a continuación, se creará una nueva clase Java con el nombre Ejercicio0X.java, siendo x el número que corresponda al ejercicio.
2. **VARIABLES** – Se creará un programa que implemente una funcionalidad para calcular el área de un rectángulo y que imprima por pantalla el texto "El rectángulo de … por … tiene un área de …". Recibirá el tamaño de los dos lados como argumentos.
3. **IF** – Se deberá comprobar el número de argumentos que reciba el programa. Si no recibe argumentos, se deberá avisar al usuario. En caso contrario, se deberá indicar cuántos ha recibido.
4. **ELSE IF** – Se deberá comprobar el número de argumentos que reciba el programa. Si no recibe argumentos, se deberá avisar al usuario. Si recibe hasta 4, se deberá indicar cuántos se han recibido. Si recibe más, se deberá avisar al usuario.
5. **FOR EACH** – Se deberán listar todos los argumentos que se reciban.

¿Qué pasa si no se recibe ninguno?

6. **FOR** – Siguiendo con el mismo ejemplo, ahora se deberán listar mostrando su posición ("0 - xxxx")

¿Qué pasa si no se recibe ninguno?

7. **WHILE** – Ahora se tienen que listar todos los argumentos hasta que se encuentre la palabra "fin", pero esta no debe ser incluida en la lista.

¿Qué pasa si no se recibe ninguno?

8. **DO WHILE** – Igual que antes, pero se deberá imprimir también la palabra de fin.

¿Qué pasa si no se recibe ninguno?

9. **SWITCH** – Se deberá implementar una funcionalidad que reciba el número de mes y devuelva el número de días que tiene (ignorando los años bisiestos). Sólo se debe hacer algo si se recibe 1 sólo argumento.
10. **CARACTERES** – Se deberá imprimir un rectángulo por la salida estándar del tamaño que se indique en los argumentos recibidos. El primer argumento será el ancho, el segundo el alto. Si el área del rectángulo pedido es múltiplo de 3, rellénalo con #, si el resto es 1, con % y si es 2 con X.
11. **OPERADOR TERNARIO** – Se deberá calcular el valor absoluto de un float.
12. **JUEGO** – Se va a jugar a piedra-papel-tijeras contra el ordenador. Se tendrá que explicar al jugador cómo se juega, pedirle que dé su jugada (**P**iedra, pape**L**, **T**ijeras, **S**alir), generar una jugada aleatoria para el ordenador y decidir quien ha ganado.

3. Capítulo 5: API de Java

1. Se deberá crear un paquete llamado com.sopra.javabasico.capitulo5. A lo largo del ejercicio, se crearán tantas clases como se crean convenientes para solucionar los problemas propuestos.
2. **STRING y STRINGBUILDER -** Dada una persona, se debe generar su nombre en el siguiente formato Nombre APELLIDO1 APELLIDO2 e imprimirlo por pantalla.
3. **STRING y STRINGBUILDER –** Se deberá generar el trigrama de dicho nombre: Inicial del nombre, inicial del primer apellido y final del último apellido (puede tener 1 o 2).
4. **ARRAYLIST** - De una lista de personas, se deberán sacar las estadísticas de cuántas tienen el mismo trigrama.
5. **ARRAYS y STRING** – Se deberá crear una funcionalidad que retire los acentos de una palabra (las palabras están en castellano y en minúsculas)
6. Se deberá generar la secuencia "1-2-3-...-30000" en un String y comparar tiempos y uso de memoria entre las distintas técnicas de concatenación aprendidas.
7. **DATETIME** - Se deberá hacer un programa que saque por pantalla la fecha y hora actual con varios formatos y en varios idiomas.
8. **DATETIME** - Ahora se deberán volver a convertir esas fechas formateadas (Strings) en fechas (Date).
9. **DATETIME** - Se deberán generar nuevas fechas, suma, resta, meses, días, …

4. Capítulo 6: Métodos y encapsulamiento

1. Se deberá crear un paquete llamado com.sopra.javabasico.capitulo6.
2. **STATIC** – Se crea una clase StaticTest con un entero estático y otro "normal", y dos métodos que incrementen cada uno de ellos y los saque por pantalla. Se deberá crear un método main en esa misma clase que cree varios objetos StaticTest. La finalidad de este ejercicio es ir jugando con los objetos, llamando a los métodos de incremento de cada uno de ellos una o varias veces.
3. **CLASES** con **ATRIBUTOS** – Se crea una clase Ciclista y se definen en ella unos cuantos atributos: nombre, ciudad, calle, número, usa casco.
4. Se crea una clase Luz con los atributos color, estado, nivel de batería.
5. Se crea una clase Bicicleta con los atributos color, número de marchas, velocidad actual, marcha actual.
6. **CONSTRUCTORES** – Se añade a cada clase un constructor que reciba todos los datos obligatorios para crear sus objetos.
7. **ATRIBUTOS** – A los atributos de las bicicletas, se deberá añadir una luz delantera y una luz trasera. Y a los ciclistas, se deberá añadir una lista de bicicletas.
8. **SETTERS** y **GETTERS** – Se añaden a cada clase métodos para acceder a los valores de sus atributos.
9. **MÉTODOS** - ¿Qué acciones se pueden hacer sobre cada uno de estos objetos? Se deberán crear métodos para apagar y encender las luces, arrancar, acelerar, frenar, subir y bajar de marcha, comprar y vender una bici.

a. Los métodos que enciendan y apaguen las luces tendrán dos variantes: para el ciclista este método seleccionara la bici (de su colección) de la cual apagar/encender las luces. En la clase bicicleta estos métodos apagarán o encenderán las luces; es decir, cambiaran su estado lógico.
b. Arrancar la bici pondrá la marcha actual y la velocidad a 1.
c. Acelerar incrementara la velocidad actual.
d. Frenar disminuirá la velocidad actual.
e. Subir/bajar de marcha aumentará o disminuirá la marcha actual.
f. Comprar/vender bici añadirá o borrará una bici de la lista de bicis en posesión del ciclista.

Para implementar los métodos habrá que usar los getters y setters creados anteriormente para modificar el estado de las variables, también se deberán imprimir por pantalla los cambios realizados. (ej: subirMracha() imprimirá un mensaje parecido a "Se ha subido la marcha".)

10. **OBJETOS** – Se deberá escribir un programa en el que se declare un ciclista con su bici.
11. **MENSAJES** – Ese ciclista deberá encender las luces de su bici, arrancar, acelerar a 10 km/h y usar la segunda marcha. Se implementará un método *main* en una nueva clase que realice las diversas acciones imprimiendo por pantalla los pasos realizados por el ciclista.

5. Capítulo 7: Diseño de clases

5.1. Ejercicio 1: Huerto

Supóngase que se tiene un huerto urbano en las ventanas de casa y se quiere un sistema que sirva para gestionarlo.

Se desea realizar una combinación de plantas aromáticas (como la menta o la albahaca), plantas de hoja (lechuga, canónigos, …), de raíz (rabanitos, zanahorias), de fruto (tomates, pimientos), leguminosas (judías, garbanzos, …), …

Cada especie tiene sus propios requisitos de espacio (volumen de sustrato que necesitan, distancia entre plantas, …), de riego, sus tiempos de germinación, etc.

Hay cultivos que son compatibles (pueden plantarse en la misma maceta) y otros que no. Suelen ser fácilmente combinables una planta de raíz, con una de hoja con una de fruto (lechuga con tomate y zanahoria).

En el huerto se tienen disponibles varios tipos de jardineras, de diferentes tamaños y formas (rectangular, tubular).

Este ejercicio pretende mostrar el inicio de un proyecto Java basado en un escenario "real", así como los pasos previos a seguir antes de comenzar a escribir código. Para ello, se hará uso de ciertos conceptos no vistos hasta el momento, pero muy utilizados en escenarios reales, como es el caso de UML (Unified Modeling Language).

1. Se deberá crear un Diagrama de Clases UML (en papel) para representar un huerto.

Una buena forma de empezar es leer un par de veces el texto que expone las necesidades que se muestran en los primeros párrafos del ejercicio y subrayar los sustantivos, ya que podrían ser una pista de las clases que se van a necesitar.

Una vez se tenga una lista de nombres, podría ser buena idea plantearse preguntas como las siguientes: ¿Es una clase, o un atributo de otra clase? ¿Es una clase o es un objeto? Si es una clase, ¿con qué otras clases se relaciona?

2. El siguiente paso será compartir el modele UML de cada persona con el resto del grupo y llegar a un modelo común (en papel).

Es posible que en este punto se discutan, por ejemplo, ciertos nombres: ¿Maceta o Jardinera?, si hace falta o no una clase con un método main, cuál es la relación entre el Huerto, las Macetas (o Jardineras) y las Plantas.

3. Una vez se haya creado un diagrama de clases consistente, se deberá implementar el modelo en Java, creando un nuevo proyecto MacetoHuerto usando como raíz el paquete com.sopra.javabasico.capitulo7.macetohuerto (es posible repartir el trabajo, y que cada persona se encargue de implementar una cosa). En primer lugar se crearán las clases y los atributos de cada clase, así como sus constructores y sus métodos getters y setters.

En Java es posible crear subpaquetes para organizar las clases (por ejemplo, uno para macetas, otro para plantas, …)

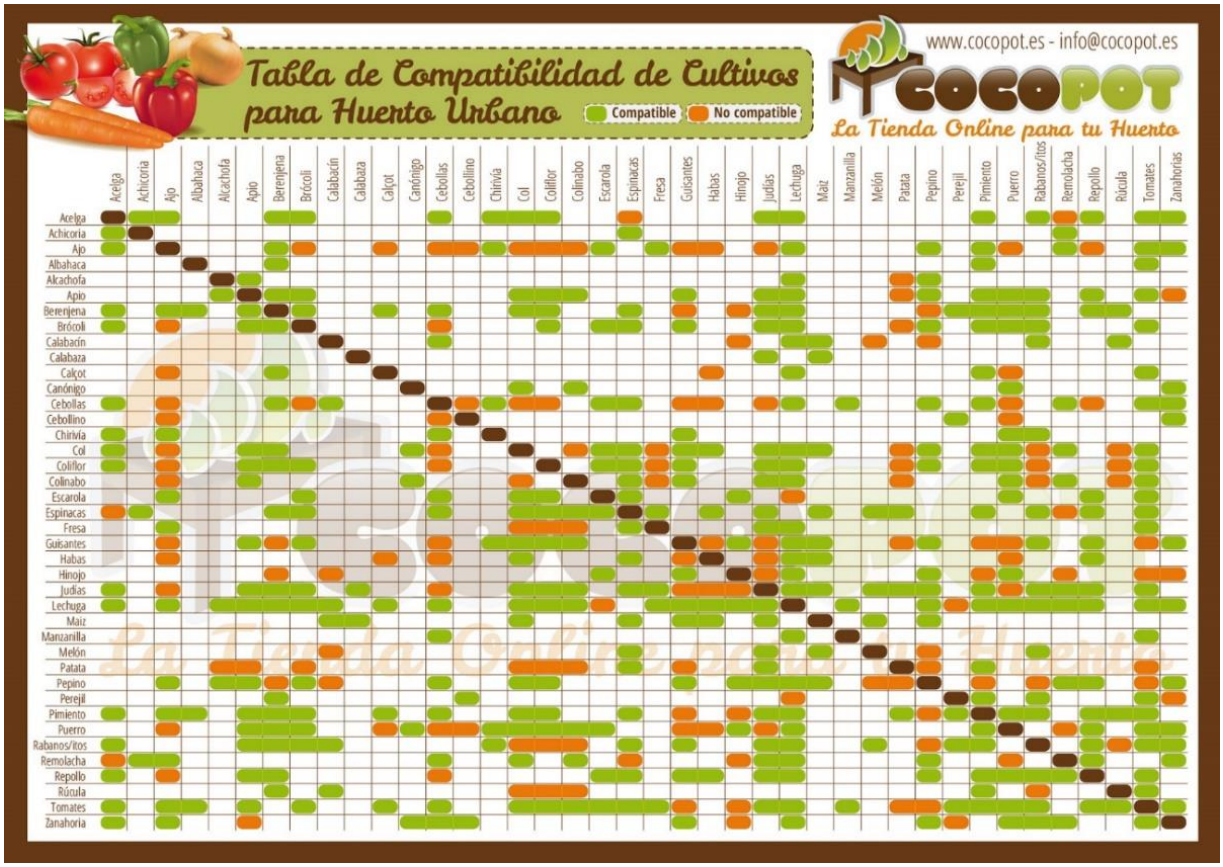
4. Posteriormente se pensará en los métodos que se incluirán en cada una de las clases. ¿Qué se quiere hacer con el huerto? El objetivo final será que el sistema sea capaz de mostrar en cuál de las macetas se debe plantar cada planta (no hace falta encontrar un sistema óptimo).

Para ello, será necesario asegurarse de que la planta que se quiere plantar cabe en la maceta; es decir, que la maceta tiene sustrato y superficie libres (no ocupados por otras plantas), y que además, la nueva planta es compatible con las que ya hay plantadas en la maceta.

Cuidado, ¡no todos los tipos de planta tienen los mismos requisitos! Las plantas de raíz también tienen un requisito de profundidad.

Para simplificar, se considerará que cada planta ocupa la superficie correspondiente al cuadrado de la distancia necesaria entre plantas. Por ejemplo, si las zanahorias necesitan 10cm de distancia entre planta y planta, plantar una zanahoria requiere $100cm^2$ disponibles.

Para saber si dos plantas son o no compatibles, se mirará la tabla siguiente. Si no se indica, se tomará la parte "…suelen ser fácilmente combinables una planta de raíz, con una de hoja con una de fruto…" del enunciado.



5.2. Ejercicio 2 : Lambdas

1. Se deberá crear un paquete llamado com.sopra.javabasico.capitulo7.lambdas
2. **LAMBDAS** - Crear una nueva clase SeleccionarNumeros con un atributo array de enteros (positivos y negativos)
3. **LAMBDAS -** Crear un método imprimirNumeros que reciba como parámetros de entrada una lista de enteros y un predicado de la clase Predicate de java.util . Este método debe recorrer la lista e imprimir solo aquellos números que cumplan con la condición del predicado.
4. **LAMBDAS -** Crear el método main y ejecutar el método anterior con los parámetros necesarios para devolver:
5. Una lista con los números positivos.
6. Una lista con los números negativos.
7. Una lista con los números pares.
8. Una lista con los números impares y negativos.
9. **LAMBDAS -** Crear una interfaz CombinarNumeros con un método cNum con dos enteros como parámetros de entrada y que tengo un entero como salida.
10. **LAMBDAS -** Instanciar la interfaz en el main de SeleccionarNumeros y asignarle una función lambda para cada uno de estos casos:
    a. Realizar una suma e imprimir por pantalla el resultado.
    b. Restar 1 a los dos números y multiplicarlos, mostrando el resultado por pantalla.
    c. Mostrar por pantalla un mensaje dependiendo de los dos números de entrada (el valor de retorno de la expresión lambda será "null"):
        i. Si los dos números son positivos ("Dos positivos")
        ii. Si los dos números son negativos ("Dos negativos")
        iii. Si un número es positivo y otro negativo ("Son diferentes")

6. Capítulo 8: Excepciones

1. **EXCEPCIONES** – Se crea un paquete llamado com.sopra.javabasico.capitulo8
2. La finalidad de este ejercicio será calcular la raíz cuadrada de la división de dos números escritos cada uno uno en una línea de un fichero cuya ruta recibirá el programa como argumento. Una vez hecho el cálculo, el resultado de este será escrito en un en un fichero de salida que se pasará al programa como segundo argumento. Es posible averiguar cómo funciona la entrada y salida de ficheros en Java llevando a cabo una breve investigación en Internet o en su documentación oficial de manera previa a la realización de este ejercicio.
   Los pasos a seguir serán los siguientes por tanto:
   - Recuperar la ruta del fichero
   - Leer las líneas del fichero
   - Convertirlas en números
   - Calcular la división y la raíz cuadrada del resultado de esta división
   - Escribir el resultado en el fichero de salida

3. **EXCEPCIONES** – Se deberán tratar todos los posibles errores, mostrando un mensaje explicativo para el usuario por consola. Solo se generará el fichero de salida si el resultado es correcto. Los casos a probar serán los siguientes:
    - El fichero de entrada no existe
    - El fichero de entrada no tiene 2 líneas
    - El contenido de las líneas no es un entero
    - El denominador es cero
    - El resultado de la división es negativo
    - El fichero de salida es de sólo lectura
4. **EXCEPCIONES** – A continuación, en vez de únicamente informar al usuario sobre los problemas que han tenido lugar durante la ejecución del programa, se deberán solucionar dichos problemas. ¿Cómo?
    - Si no es posible leer los números del fichero, se pedirán al usuario por la consola
    - Si el denominador es 0, se cambiará a 1 (informando de ello al usuario por la consola)
    - Si el resultado de la división es negativo, se sacará el valor absoluto (informando de ello al usuario por la consola)
    - Si no es posible escribir en el fichero de salida, se intentará escribir en otro fichero de salida (mismo nombre que el que dice el usuario, pero añadiéndole un ".err" al final) (informando de ello al usuario por la consola)
    - Si tampoco es posible, se sacará por consola el resultado, y un aviso de que no se ha podido escribir en el fichero