

MÓDULO INICIAL. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

Relación de Problemas N° 3

Proyecto BancoV2 (BancoV1, listas, herencia, excepciones)(labs)

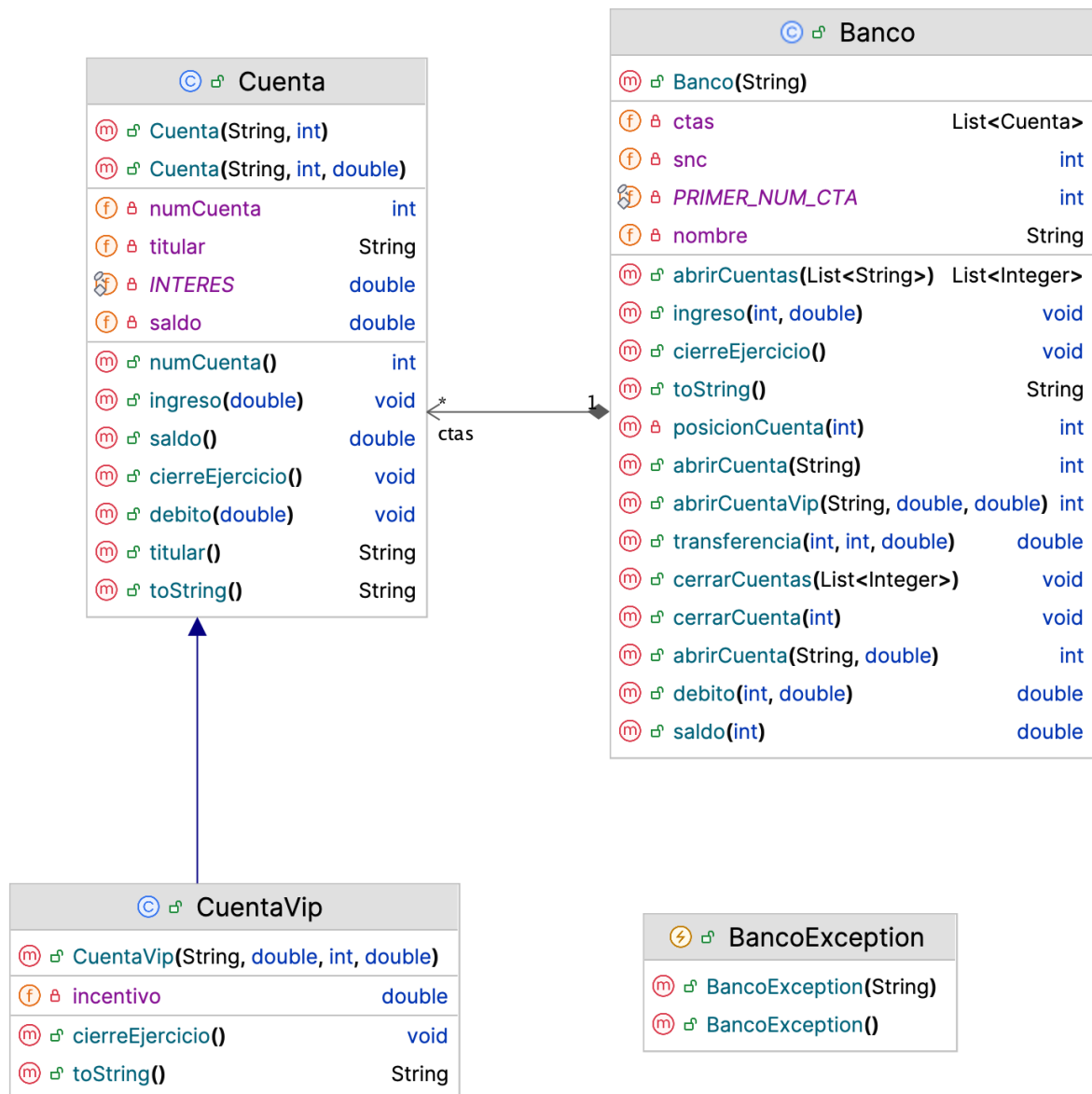
Modifica el ejercicio del proyecto BancoV1 de la siguiente forma (Haz una copia aquí de los ficheros del proyecto BancoV1):

1. Añade a la clase `Cuenta` el método público `void cierreEjercicio()` que incremente el saldo de la cuenta en un 2 por ciento por los intereses del ejercicio (define una constante de clase para almacenar ese porcentaje, 0.02).
2. Crea la clase `CuentaVip` que se comporta igual que la clase `Cuenta` (es decir que extiende la clase `Cuenta`), pero que además:
 - a. Dispone de un atributo privado de tipo `double` denominado `incentivo`, que determinará una cantidad que el banco ingresará en esa cuenta al cierre del ejercicio por tratarse de una cuenta vip.
 - b. Dispone de un constructor en el que además del titular, el saldo y el número de cuenta se proporciona el valor del incentivo, `CuentaVip(String tit, double s, int n, incentivo double)`.
 - c. Dispone de un método público `void cierreEjercicio()`. El cierre de ejercicio de una cuenta vip es como el de un objeto de la clase `Cuenta`, pero tras incrementar el saldo con los intereses se ingresa o añade en la cuenta el valor de `incentivo`.
 - d. Redefine el método público `String toString()` para que además de mostrar la información como un objeto de la clase `Cuenta`, muestre el incentivo entre símbolos '\$' (mira el resultado en el ejemplo proporcionado).
3. Añade a la clase `Banco` el método público `public int abrirCuentaVip(String tit, double saldoInicial, double incentivo)`, que crea una `CuentaVip` con titular `tit`, saldo `saldoInicial` e incentivo `incentivo`, la mete en la lista `ctas` y devuelve el número de cuenta asignada. Su comportamiento es idéntico al método `abrirCuenta`.
4. Añade a la clase `Banco` el método público `void cierreEjercicio()` que cierre el ejercicio de todas sus cuentas.
5. Añade a la clase `Banco` el método público `List<Integer> abrirCuentas(List<String> titulares)` que añada una cuenta de la clase `Cuenta` con saldo 0 por cada titular que aparece en la lista `titulares`. Este método debe devolver una lista con los números de cuenta de todas las cuentas que se han creado.
6. Añade a la clase `Banco` el método público `void cerrarCuentas(List<Integer> numCuentas)` que elimine todos los objetos de la clase `Cuenta` cuyos números de cuenta aparecen en la lista `numCuentas`.
7. Añade a la práctica la excepción `BancoException` no comprobada. Sustituye todas las llamadas a `RuntimeException` por llamadas a `BancoException`.
8. Modifica el método `cerrarCuentas` para que si una cuenta de las que se pasa en el array argumento no existe en el banco, la ignore y siga con el resto.

Utiliza el programa de prueba `TestBancoB` para verificar tu implementación.

El resultado debe ser el siguiente:

```
TubbiesBank: [[(Po/1001) -> 500.0]$300.0$ [(Dixy/1002) -> 500.0] [(Tinky Winky/1003) -> 500.0]$100.0$
[(Lala/1004) -> 500.0] ]
[[Po/1001) -> 600.0]$300.0$ [(Dixy/1002) -> 400.0] [(Tinky Winky/1003) -> 400.0]$100.0$ [(Lala/1004) ->
600.0] ]
[[Dixy/1002) -> 600.0] [(Tinky Winky/1003) -> 400.0]$100.0$ [(Lala/1004) -> 600.0] ]
[[Dixy/1002) -> 600.0] [(Tinky Winky/1003) -> 400.0]$100.0$ [(Lala/1004) -> 600.0] [(Dora/1005) -> 0.0]
[(Botas/1006) -> 0.0] [(Pedro/1007) -> 0.0] ]
[1005, 1006, 1007]
[[Dixy/1002) -> 600.0] [(Tinky Winky/1003) -> 400.0]$100.0$ [(Lala/1004) -> 600.0] ]
[[Dixy/1002) -> 612.0] [(Tinky Winky/1003) -> 508.0]$100.0$ [(Lala/1004) -> 612.0] ]
[[Tinky Winky/1003) -> 508.0]$100.0$ [(Lala/1004) -> 612.0] ]
```



Proyecto NPIV4 (NPIV3, split) (labs)

Copia en este proyecto todas las clases del proyecto NPIV3. Añade a la clase NPI el método

```
public double evaluaStr(String expr)
```

que dada una cadena que representa una expresión en notación polaca inversa, la evalúa devolviendo el resultado. La cadena tendrá espacios en blanco entre cada operador, función o número.

Cambia la clase Main por la siguiente:

```
import npv.NPI;
public class Main {
    public static void main(String [] args) {
        // 5 + (6.4 - 2) * raiz(3.1);
        // NPI: 5 6.4 2 - 3.1 raiz * +
        NPI cpi = new NPI();
        double res = cpi.evaluaStr("5 6.4 2 - 3.1 raiz * +");
        System.out.println("Resultado " + res);
    }
}
```

Proyecto NPIV5 (NPIV4, tipo algebraico) (labs)

Crea el proyecto NPIV5 y copia todas las clases de NPIV4.

Con la implementación que tenemos actualmente de la clase NPI no podemos evaluar expresiones que utilicen más de las cuatro variables que tenemos. Por ejemplo, la expresión

$$3*(24 - 4*(5+2)) \text{ que en notación polaca inversa es } 3 \ 24 \ 4 \ 5 \ 2 \ + \ * \ - \ *$$

no se puede evaluar.

Para solucionar este inconveniente, vamos a crear una estructura `Stack` y unas modificaciones a la clase `NPI`.

Define en el paquete `stack` el tipo algebraico `Stack`. Un stack o pila es una estructura que colecciona elementos (en este caso, por simplificar, van a ser valores reales (`double`)) de manera que el último dato introducido en la pila es el primero que sale.

Las operaciones que se pueden hacer con un stack son las siguientes:

- push: Añade un elemento en la cima del stack
- pop: Elimina la cima del stack
- top: Devuelve el elemento en la cima del stack sin eliminarlo
- isEmpty: Chequea si el stack está vacío



Se va a definir el `Stack` como un tipo algebraico: Una pila puede estar vacía (`Empty()`) o tener un elemento en la cima seguido de otra pila (`Node(double, Stack)`). Como se ve, es un tipo algebraico recursivo.

Define la interfaz sellada `Stack` que implemente esos cuatro métodos por defecto y que solo permita que la implementen los record `Empty` y `Node`.

Para la evaluación de una expresión en notación polaca inversa puede utilizarse un stack de la siguiente manera:

- Partimos de un stack vacío.
- Recorremos la expresión de izquierda a derecha:
 - o Si encontramos un número lo introducimos en el stack
 - o Si encontramos un operador
 - o Sacamos la cima del stack (segundo argumento)
 - o Sacamos la nueva cima del stack (primer argumento)
 - o Aplicamos el operador con estos argumentos
 - o Introducimos el resultado en el stack
- Al finalizar, la cima del stack tendrá el resultado

Modifica la clase `NPI` para que, en lugar de utilizar las variables `x`, `y`, `z`, `t`, utilice un stack.

Proyecto Mastermind (herencia, scanner, equals, excepciones)

Se desea implementar un juego (Mastermind, en el paquete mastermind) en el que el usuario ha de adivinar una combinación secreta de cifras no repetidas creada aleatoriamente por el programa. El usuario tendrá la posibilidad de hacer varios intentos para adivinar la combinación secreta, respondiendo el programa a cada uno de estos intentos con el número de cifras acertadas, indicando cuántas de éstas están en la posición correcta y cuántas aparecen en la combinación secreta, aunque en una posición diferente. Por ejemplo, si la combinación secreta es 4235 y el jugador intenta 4350 el programa responderá con que el número de cifras colocadas es 1 (la cifra 4) y el de descolocadas es 2 (las cifras 3 y 5). Para ello:

1) Cread el record Movimiento cuyas instancias mantengan información sobre un intento (como un String) y dos números enteros, los cuales indicarán el número de cifras bien colocadas y descolocadas. Se pide:

2) Definid la clase MastermindException que en lo sucesivo se utilizará para crear excepciones en caso de que se produzca algún tipo de error. Esta excepción será de no obligado tratamiento.

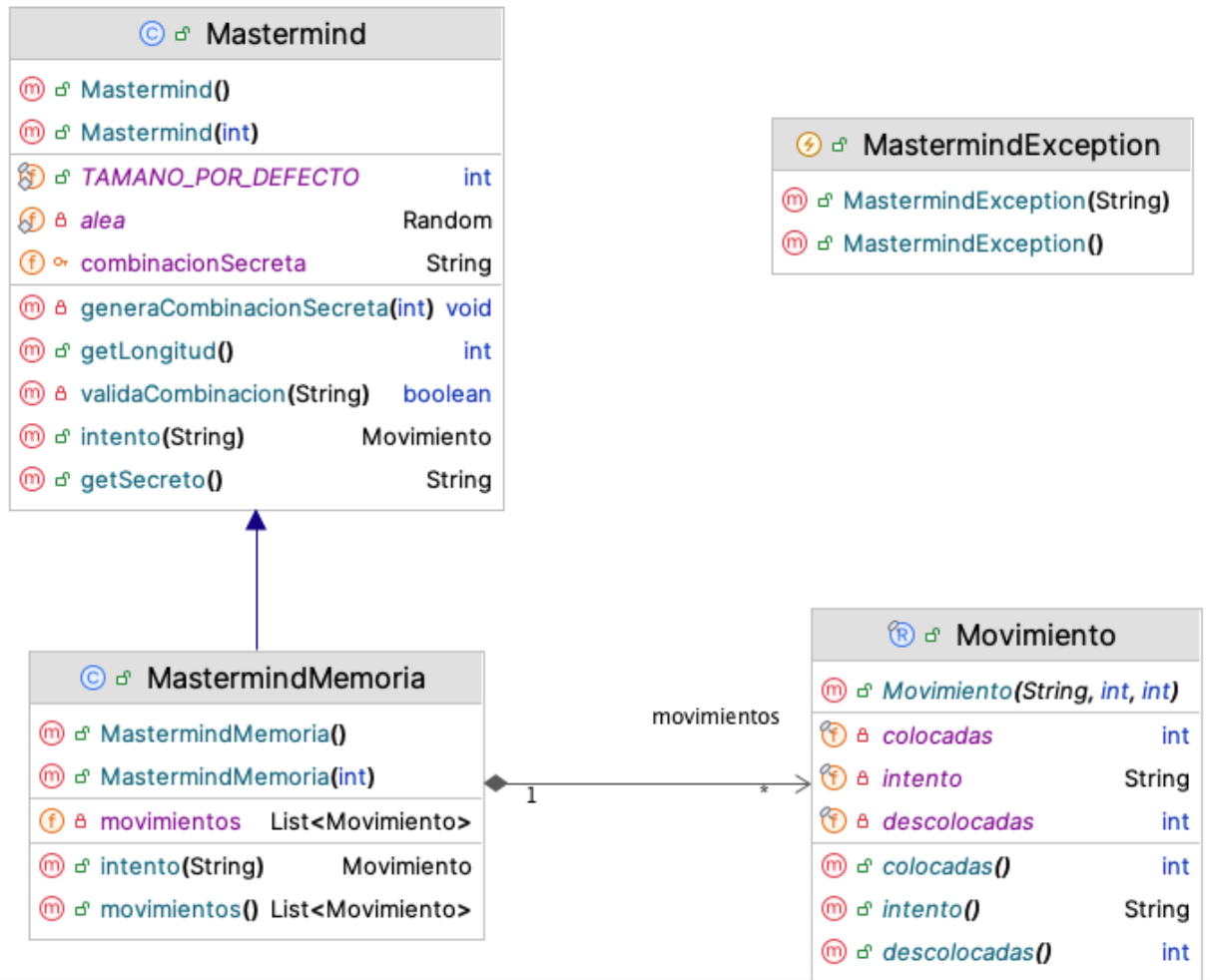
3) Cread la clase Mastermind, la cual contendrá una combinación de cifras secreta (un String), de manera que cualquier usuario de esta clase puede intentar averiguar dicha combinación. Ninguna combinación de cifras podrá contener cifras repetidas. Se pide:

- Un constructor que crea una instancia del mastermind dado el tamaño de las combinaciones de cifras. Habrá un segundo constructor sin argumentos que creará una instancia del juego con combinaciones de un tamaño por defecto (en este caso, 4). La combinación secreta se creará en el momento de la creación del mastermind. Un argumento no positivo o mayor que el número de cifras disponibles (10) producirá el lanzamiento de la excepción MastermindException.
- Definid el método `int getLongitud()` que proporcione la longitud de la combinación en este juego.
- Definid un método `private boolean validaCombinacion(String cifras)` que compruebe si una cadena de caracteres corresponde a un movimiento válido. Una cadena es un movimiento válido si tiene la longitud correcta, es decir, las cifras que exige el juego, todos los caracteres son números y no contiene cifras repetidas.
- Definid un método `public Movimiento intento(String cifras)` tal que dada una cadena de cifras comprueba si es una cadena válida y entonces devuelva un objeto Movimiento que contenga las cifras, el número de cifras que aparecen en la combinación secreta en la misma posición y el número de cifras en posiciones distintas. Si la cadena de cifras no es válida lanzará la excepción MastermindException.
- Definid un método `public String getSecreto()` que permita conocer la combinación secreta. Devolverá una cadena de caracteres con la combinación secreta.

4) Cread la clase MastermindMemoria, cuyos objetos se comportan igual que los de la clase Mastermind excepto que éstos recuerdan todos los movimientos válidos que se han intentado hasta el momento, no permitiendo que el usuario repita combinaciones de cifras. Las combinaciones repetidas provocan el lanzamiento de la excepción MastermindException. Se pide:

- Constructores con argumentos como los de la clase Mastermind.
- Redefine el método `Movimiento intento(String cifras)` de forma que tenga el comportamiento esperado.
- Definid un método `List<Movimiento> movimientos()` que proporcione una lista con los movimientos válidos ya realizados.

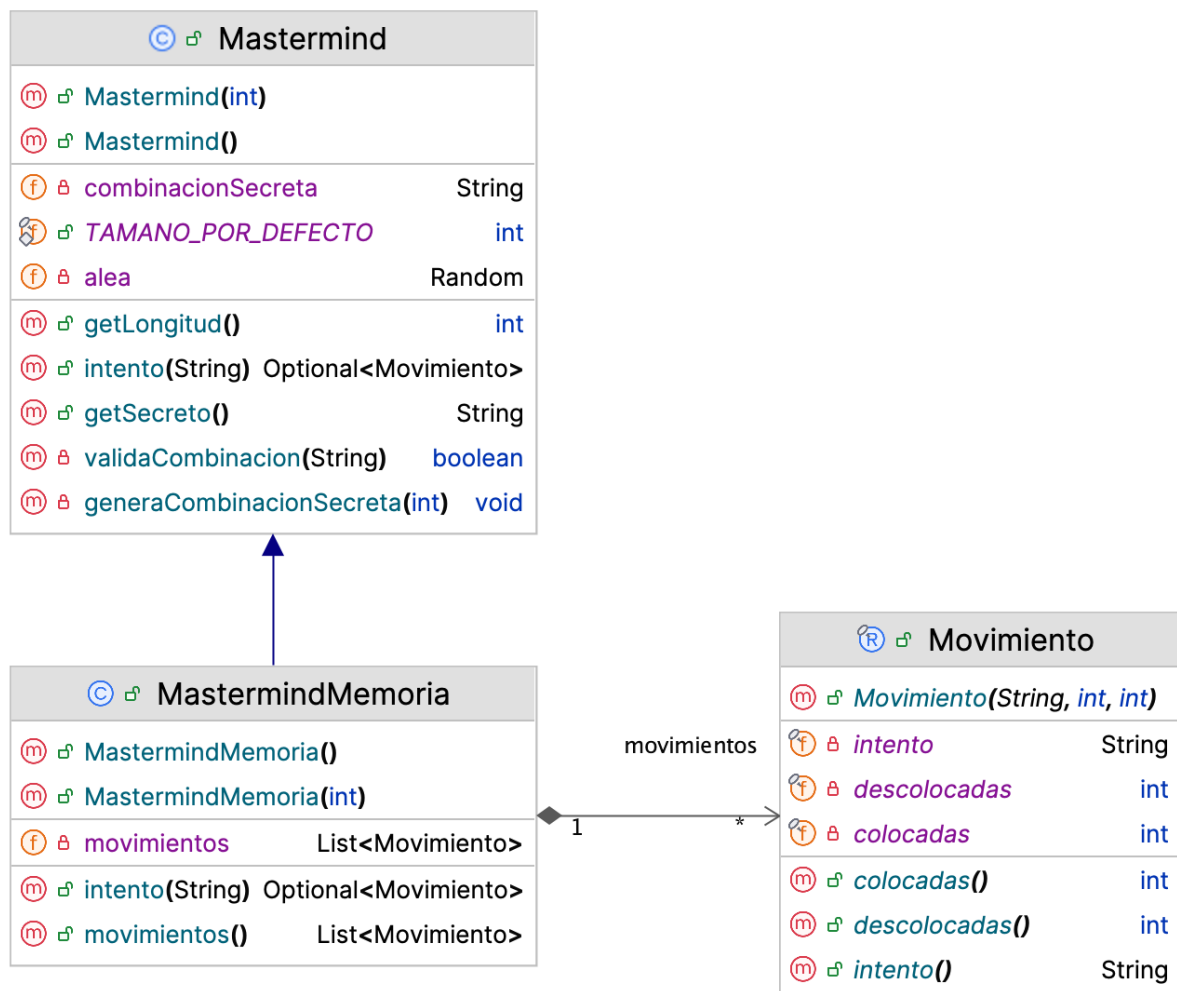
Probad las clases con el programa `TestMastermind` proporcionado.



Proyecto MastermindO (herencia, scanner, equals, Optional)

Vamos a hacer una modificación al proyecto Mastermind de manera que no crearemos la excepción MastermindException. En su lugar utilizaremos la clase Optional para comunicar el resultado de un intento del usuario. En este proyecto se debe copiar el paquete mastermind del proyecto Mastermind excepto la clase MastermindMemoria. Utilizaremos un nuevo programa de prueba TestMastermind.java.

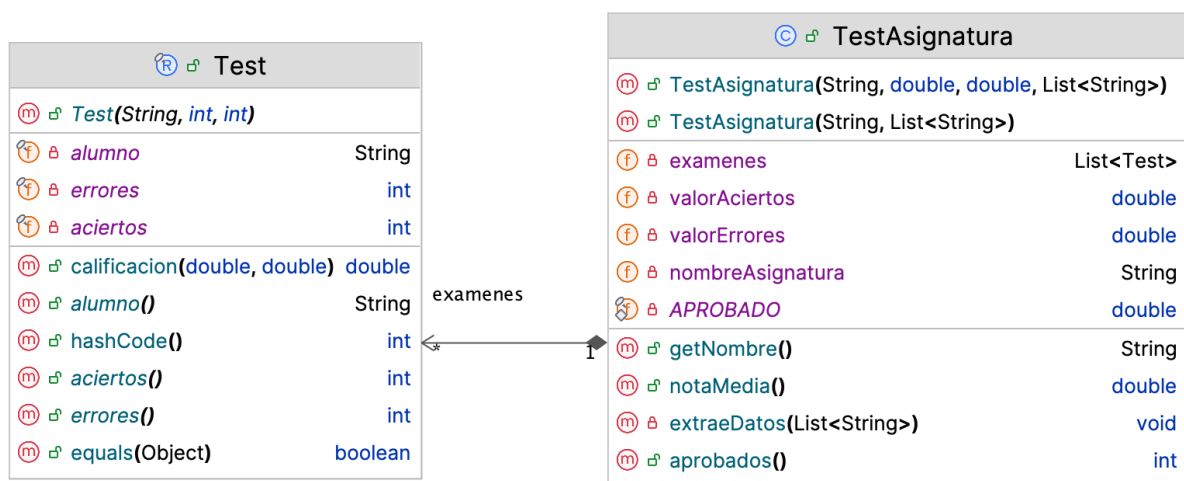
- En el constructor de la clase Mastermind, se debe sustituir la excepción MastermindException por la excepción IllegalArgumentException.
- En los métodos intento de las clases Mastermind y MastermindMemoria se debe evitar el lanzamiento de excepciones. Este método ahora devolverá un Optional<Movimiento> de manera que si el intento es correcto se devolverá el opcional con un movimiento y si no lo es se devolverá el opcional vacío.
- Se proporciona el programa de prueba TestMastermind.java para probar el juego.



Proyecto Test (listas, scanner, equals)(labs)

Se va a crear una aplicación para anotar las calificaciones obtenidas por alumnos en exámenes tipos test de una asignatura. Para ello se crearán las clases `Test` y `TestAsignatura` en un paquete denominado `tests`.

- 1) El record `Test` mantiene información sobre el examen tipo test de un estudiante. Tiene como argumentos nombre del alumno (`String`), y el número de aciertos y errores (ambos de tipo `int`) en las respuestas contestadas del test. En el record se debe incluir:
 - a) Redefinición de `equals`. Dos tests se consideran iguales cuando corresponden al mismo estudiante, independientemente de que el nombre incluya mayúsculas o minúsculas.
 - b) El método `public double calificacion(double valAc, double valErr)`, devolverá la calificación del test, considerando el número de aciertos por la valoración de cada uno de ellos más el número de errores por la valoración de cada uno. La valoración de cada acierto y error se proporcionan como parámetros.
- 2) La clase `TestAsignatura` deberá incluir información sobre el nombre de la asignatura (`String`), una lista de `tests`, la valoración de las preguntas acertadas y la valoración de las preguntas falladas. En este caso, la clase incluirá:
 - a) Dos constructores. Uno con cuatro argumentos donde el primero determine el nombre de la asignatura, el segundo la valoración de cada acierto, el tercero la valoración de cada error y el cuarto argumento es una lista de cadenas de caracteres donde cada posición incluye información sobre el resultado (aciertos y errores) de un estudiante. El formato de cada posición será:
`Apellidos, Nombre: aciertos + errores`
Si la valoración de las acertadas no es positiva o la valoración de las falladas no es negativa o 0 se lanzará una excepción `RuntimeException`.
El segundo constructor solo incluirá dos argumentos, uno con el nombre de la asignatura y otro con la lista de cadenas de caracteres. En este caso, se supondrá que valoración de aciertos será 1, y la valoración de errores, 0. En ambos casos, a partir de la lista de cadenas de caracteres habrá que generar la lista de objetos `Test`. Si alguna línea es errónea se muestra un mensaje de error por consola y se sigue con la siguiente.
NOTA. Se recomienda crear un método privado, `void extraeDatos(List<String>)` que se encargue de extraer la información de la lista de datos y completar la lista de `tests`.
 - b) Un método `public double notaMedia()` para calcular la nota media de todos los exámenes.
 - c) Un método `public int aprobados()` para devolver el número de aprobados, considerando como tal al examen que supere la calificación de 5.



Probad estas clases con el programa de prueba `PruebaTest` proporcionado.

La salida al ejecutar el programa anterior es

Asignatura: Programación Orientada a Objetos

Aprobados en el test: 6

Nota media en el test: 6.0

Proyecto BusV1 (listas, interfaces, IO)(labs)

Se va a crear una aplicación para controlar los autobuses de línea que hay en una ciudad. Para ello se crearán las clases `Bus`, `Servicio`, `EnMatricula`, `PorLinea` y la interfaz `Criterio`, todas en el paquete `buses`.

- Mientras no se indique lo contrario, las variables de instancia serán privadas y los métodos públicos.
- Pueden añadirse los métodos privados que se consideren.

Primera parte

Clase `Bus`

- Crea la clase `Bus` que mantiene información de un autobús de línea del cual se conocen el código del autobús (`int codBus`), la matrícula (`String matricula`) y la línea a la que pertenece (`int codLinea`). La clase tendrá un constructor cuyos argumentos son el código del autobús y la matrícula.
- Crea el método `void setCodLinea(int codLinea)` que asigna el código de la línea a la que pertenece el autobús. Crea los métodos `int getCodBus()`, `String getMatricula()` y `int getCodLinea()` que devuelven el código del autobús, la matrícula y el código de la línea a la que pertenece respectivamente.
- Dos autobuses serán considerados iguales si coinciden su código de autobús y su matrícula. La letra de la matrícula podrá estar indistintamente en mayúsculas o minúsculas.
- La representación de un autobús (método `toString()`) debe mostrar las tres variables de instancia. Por ejemplo:

```
Bus (675,2959CDN,25)
```

Clase `Servicio`

- Crea la clase `Servicio` que colecciona los autobuses que hay en una ciudad. Esta clase contiene un `String` que indica el nombre de la ciudad y una colección (una lista llamada `buses`) de autobuses. Define un constructor que tiene como argumento el nombre de la ciudad del servicio. Este constructor debe crear la lista para almacenar los autobuses.
- Define el método `String getCiudad()` que devuelve el nombre de la ciudad y otro `List<Bus> getBuses()` que devuelve la lista de buses.
- Para incluir autobuses en la lista define el método `void leeBuses(String file) throws IOException`, que lee los datos de los autobuses del fichero `file`. El fichero de entrada contiene los datos de cada autobús en una línea del fichero con el siguiente formato (ver el fichero `buses.txt`):

```
<codBus>,<matricula>,<codLinea>
```

Cuando en la lectura de los datos de un autobús se produzca algún tipo de error, bien porque falten datos o porque alguno sea incorrecto, se deberá mostrar en consola el error y se sigue con el resto del fichero de entrada.

Clase `MainPrueba`

- Crea una aplicación `MainPrueba` en el paquete por defecto que cree un servicio para la ciudad de Málaga con los autobuses del fichero `buses.txt`. Esta aplicación mostrará por consola la ciudad y todos los autobuses, uno a uno.

Segunda parte

Interfaz Criterio

Se desea obtener una selección de autobuses atendiendo a diferentes criterios de selección. Así, nos podrán interesar los autobuses que pertenecen a una línea determinada, es decir, que tienen un código de línea dado, los que contienen una cadena de caracteres en su matrícula, etc.

- Define la interfaz `Criterio` que contenga el método boolean `esSeleccionable (Bus bus)` que indica si el autobús pasado como argumento se debe seleccionar según este criterio.

Clase EnMatricula

- Crea la clase `EnMatricula` que implementa un `Criterio` y mantiene una cadena de caracteres que se le pasa en el constructor.
- Implementa el método de selección de manera que el autobús argumento será seleccionable si su matrícula contiene la cadena dada en el constructor.
- Redefine el método `String toString()` para que muestre

```
Autobuses cuya matricula contiene xxx
```

Clase PorLinea

- Crea la clase `PorLinea` que implementa un `Criterio` y mantiene un código de línea que se le pasa en el constructor.
- Implementa el método de selección de manera que el autobús argumento será seleccionable si su código de línea coincide con el dado en el constructor.
- Redefine el método `String toString()` para que muestre

```
Autobuses de la linea xxx
```

Clase Coincide

- Crea la clase `Coincide` que implementa un `Criterio` y mantiene un autobús que se le pasa en el constructor.
- Implementa el método de selección de manera que el autobús argumento será seleccionable si el bus coincide con el dado en el constructor.
- Redefine el método `String toString()` para que muestre

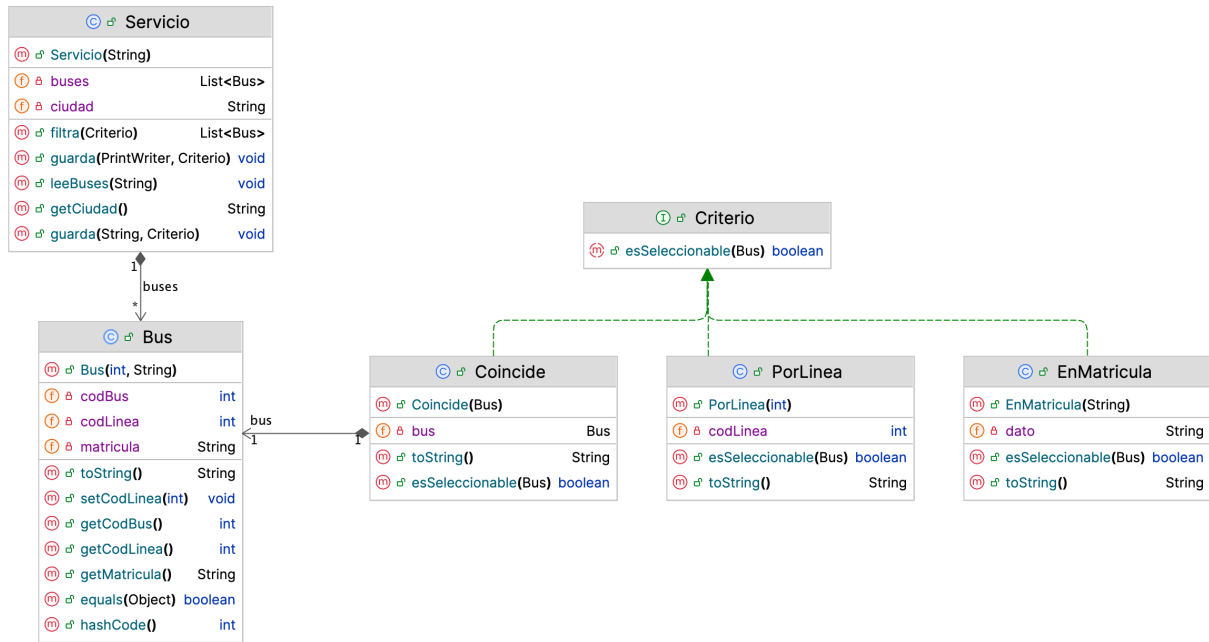
```
Autobús xxxxxxxx
```

Clase Servicio

Continuamos con la clase `Servicio` para añadir nuevos métodos.

- Crea el método `List<Bus> filtra (Criterio criterio)` que devuelve una lista con aquellos autobuses del servicio que cumplen el criterio que se pasa como argumento.
- Se pretende guardar en un fichero los autobuses que seleccionemos con el método anterior. Para ello se van a crear dos métodos. El primero `void guarda (String file, Criterio criterio) throws FileNotFoundException`, guarda en el fichero `file` los autobuses que cumplan el criterio dado como segundo argumento. Para ellos se ayudará de otro método `void guarda (PrintWriter pw, Criterio criterio)` que guarda los autobuses seleccionados por el criterio en el dispositivo de salida proporcionado como primer argumento.

A continuación, se muestra un diagrama de clases del ejercicio completo.



Se proporciona la clase `MainBuses` que hace una prueba de todo lo anterior. La salida del programa `MainBuses` debe ser la mostrada abajo. Además, debe crear dos ficheros, el primero, `linea21.txt`, con los autobuses de la línea 21, el segundo, `contiene29.txt`, con los buses que contiene el 29 en la matrícula:

Malaga

Error, faltan datos en 636,1338RFE
Error en dato numérico en 714,4956XFU,71

Autobuses de la línea 21

Bus(653,1540HIH,21)
Bus(652,7658PDM,21)
Bus(590,3774ARP,21)
Bus(654,3513FZV,21)
Bus(523,8297WHJ,21)
Bus(634,5009KVE,21)
Bus(575,9452NKW,21)
Bus(645,6675NFF,21)

Autobuses cuya matricula contiene 29

Bus(463,2934IAC,3)
Bus(619,2298VGL,10)
Bus(656,2965JGB,20)
Bus(523,8297WHJ,21)
Bus(477,2906HCE,22)
Bus(675,2959CDN,25)
Bus(593,3729MMI,32)
Bus(713,0293PVU,71)

Proyecto LibreriaV4 (LibreriaV3, equals)

Se van a realizar algunas modificaciones al proyecto de librería en su tercera edición para que pueda conocerse cuando dos libros son iguales. Para ello, crear una copia del proyecto LibreriaV3 en el proyecto LibreriaV4 y modificar la clase Libro:

- Clase Libro: dos libros son iguales si lo son su autor y su título independientemente de la tipografía. No se tiene en cuenta cualquier otro dato.
- Clase Libreria: modificar lo necesario para que ahora se utilice `equals` para buscar un libro.

Proyecto UniversoV2 (UniversoV1, IO)(challenge)

Utilizando el proyecto UniversoV1 (habrá que publicarlo y añadir dependencias al pom.xml), vamos a crear otro proyecto donde los datos de las partículas se lean desde un fichero. Para ello, en el paquete universo vamos a crear el record DatosUniverso. Se dispone también de una aplicación MainFichero y de una serie de ficheros con los datos de un universo y su configuración en el directorio data/universos.

Cada fichero de configuración tiene el siguiente formato

```
<numero de particulas>
<radio universo>
<incremento de tiempo>
<sleep de animaciones>
<pos1x> <pos1y> <vel1x> <vel1y> <masa1> <radio1>
<pos2x> <pos2y> <vel2x> <vel2y> <masa2> <radio2>
...
<posnx> <posny> <velnx> <velny> <masan> <radion>
```

<numero de particulas> es el número de partículas que aparecen al final de este fichero.

<radio universo> es el radio visible del universo.

<Incremento de tiempo> es el delta de tiempo de cada animación.

<sleep de animaciones> lapsus de tiempo de una animación en pantalla.

<posix> la abscisa de la posición de la partícula i-ésima

<posiy> la ordenada de la posición de la partícula i-ésima

<velix> la abscisa del vector velocidad de la partícula i-ésima.

<veliy> la ordenada del vector velocidad de la partícula i-ésima.

<masai> la masa de la partícula i-ésima.

<radioi> el radio de la partícula i-esima.

Record DatosUniverso

Este record tendrá tres variables de instancia, `Universo universo`, `double dt` y `int sleep`. La primera es un universo, la segunda es el delta de tiempo de cada animación y la tercera es el lapsus de tiempo que se mantiene una animación en pantalla.

- El método

```
public static DatosUniverso leeDatos(String fichero) throws IOException
```

permite leer del fichero dado los datos de un universo, su delta de tiempo transcurrido en una animación y el lapsus de tiempo en el que se mantiene una animación en pantalla.
- El primer argumento de la aplicación se utiliza para nombrar al fichero con el que se quiere trabajar.
- La clase aplicación `MainFichero` puede usarse para ejecutar la animación. Esta clase necesita que se pase como argumento de la aplicación el nombre del fichero.

Proyecto AlturasV1 (listas, interfaces, IO)(*mandatory*)

Se va a crear una aplicación para manipular las alturas medias de la población de los distintos países del mundo. Para ellos se van a crear las clases `Pais` y `Mundo` en el paquete `alturas`. Todos los métodos que se piden se considerarán públicos y todas las variables de instancia privadas.

El record `Pais`

El record `Pais` mantiene información de un país, el continente al que pertenece y la altura media de sus habitantes. Así un país tendrá un nombre de país (`String nombre`), un nombre de continente (`String continente`), y la altura media de sus habitantes (`double altura`).

Clase `Mundo`

La clase `Mundo` mantiene información de los países del mundo mediante una lista de países (`List<Pais> paises`).

- a) Define un constructor que inicialice adecuadamente la estructura.
- b) Define el método `List<Pais> getPaises()` que devuelve la lista de países.
- c) Define el método `void leePaises(String file) throws IOException` que lee los países del fichero cuyo nombre se pasa como argumento.
El fichero tiene el formato (mirar el fichero `alturas.txt`)

`<nombre país>,<continente>,<altura>`

Interfaz `Seleccion`

- a) Para seleccionar algunos países crea una interfaz con un único método `boolean test(Pais pais)` que determina si el país argumento es seleccionable o no.

Clase `MayoresQue`

Esta clase implementa la interfaz `Seleccion` de manera que solo selecciona los países cuya media de altura de la población es mayor que un valor dado. Para ellos tendrá una variable de instancia (`double alturaMin`) que indicará la altura mínima exigida.

- a) Define un constructor que proporcione la altura mínima exigida.
- b) Redefine el método `boolean test(Pais pais)` para que solo sea cierto para aquellos países cuya altura sea mayor o igual a la mínima exigida.

Clase `MenoresQue`

Esta clase implementa la interfaz `Seleccion` de manera que solo selecciona los países cuya media de altura de la población es menor que un valor dado. Para ellos tendrá una variable de instancia (`double alturaMax`) que indicará la altura máxima permitida.

- a) Define un constructor que proporcione la altura máxima permitida.
- b) Redefine el método `boolean test(Pais pais)` para que solo sea cierto para aquellos países cuya altura sea menor a la máxima permitida.

Clase EnContinente

Esta clase implementa la interfaz `Seleccion` de manera que solo selecciona los países que contengan un determinado texto en el nombre del continente. Para ellos tendrá una variable de instancia (`String texto`) que indicará el texto que deben contener.

- Define un constructor que proporcione el texto que deben contener.
- Redefine el método `boolean test(Pais pais)` para que solo sea cierto para aquellos países cuyo continente contengan el texto dado.

Clase Mundo

Para seleccionar países según un criterio, se define el método `List<Pais> selecciona(Seleccion sel)` que devuelve los países que cumplen el test que define `sel`.

Clase MainMundo

Crear una aplicación que cree un objeto de la clase `Mundo` y lea los países del fichero `alturas.txt`. Luego seleccione los países con talla media mayor o igual que `1.77` y los imprima en consola uno a uno. Posteriormente seleccione los países con talla media menor que `1.77` y los imprima igualmente en consola. Por último, seleccione los países del continente `Europe` y también los imprima en consola.

