Pràctica 4 (P4)

Recorregut i cerca, taules/arrays i classes

Laboratori de Programació I – Curs 18-19

Grau d'Enginyeria Informàtica

Departament de Matemàtiques i Informàtica

Facultat de Matemàtiques i Informàtica

UNIVERSITAT DE BARCELONA

- En tots els problemes, cal que identifiqueu totes les **seqüències** existents i amb quin **esquema algorísmic** cal tractar-les (recorregut o cerca). La identificació es fa en un comentari abans de la composició iterativa corresponent.

- Heu de resoldre **TOTS** els exercicis fent servir **mètodes** que es criden des del mètode main(). Aquests mètodes s'han de definir bé com a mètodes de classe (primers exercicis de la llista) o bé com a mètodes d'objecte (part final de la llista).

- **Tan aviat com s'expliqui JUnit, heu de fer servir test unitaris per tal de testejar els mètodes implementats**. Reviseu les transparències de JUnit del Campus Virtual.

1. Crea un programa que converteixi una frase (acabada en «fi») en mono vocàlica. Per a fer-ho caldrà llegir una frase i crear un mètode `monovocalica` que donada una cadena de caràcters (una paraula) faci la transformació mono vocàlica amb la vocal "i" (**FraseMonovocalica.java**).

```
static String monovocalica (String paraula)
```

Un exemple d'execució és:

```
$ java FraseMonovocalica

Entra una frase:
Hola, com estas fi

La frase monovocalica es: Hili, cim istis
```

2. Volem generar un alfabet que sigui útil per a twitter i sms que són serveis que requereixen pocs caràcters. Crea un programa que donada una frase acabada en «fi» transformi aquesta frase amb les següents condicions: elimini les vocals de totes les paraules i elimini signes de puntuació. (". " "," ":" ";" "-" " ' **(TransformaFrase.java)**. Utilitzeu els mètodes `transforma, esVocal, esSigne`, que també haureu d'implementar.

```
static String transforma(String paraula)

static boolean esVocal(char lletra)

static boolean esSigne(char lletra)
```

Un exemple d'execució és:

```
$ java TransformaFrase

Entra una frase acabada amb el mot fi:

  Hola, adios, hasta luego fi

La frase Transformada es: Hl ds hst lg
```

3. Crea un programa que a l'iniciar-se creï una taula/array amb els dies de la setmana ("Dilluns", "Dimarts", "Dimecres", etc..). A continuació es demanarà per consola un número del 1 al 7 i es mostrarà el dia que correspon a aquell número (Exemple: Num: 3 -> "Dimecres"). Escriu un mètode `nomDiaSetmana` tal que donat un nombre n retorni el dia de la setmana corresponent. (**DiesSetmana.java**).

```
static String nomDiaSetmana(int n)
```

4. Crea un programa que a l'iniciar-se creï una taula amb els dies de la setmana ("Dilluns", "Dimarts", "Dimecres", etc..). A continuació es demanarà per consola un dia de la setmana i el programa dirà quin número de dia de la setmana és (Exemple: Dia: "Dissabte" -> 6). Escriu un mètode `nombreDiaSetmana` tal que donat el nom d'un dia de la setmana retorni el nombre del dia corresponent. **(BuscaDiaSetmana.java)**

```
static int nombreDiaSetmana(String nom)
```

5. Crea un programa que sol·liciti n enters a l'usuari, els emmagatzemi a una taula i digui si els enters estan ordenats de forma creixent. Escriu un mètode `ordenats` que retorni true si els enters estan ordenats de forma creixent, i false en cas contrari. Cóm modificaries el mètode sense utilitzar la taula per emmagatzemar els enters? **(EntersOrdenats.java)**

```
static boolean ordenats(int [] taula)
```

6. Crea un programa que llegeixi n nombres enters que poden estar duplicats, el programa ha d'imprimir la llista d'enters sense duplicitats. Escriu un mètode `eliminaDuplicats` que s'encarregui d'eliminar els duplicats. Quina diferència veieu en les propostes de mètodes següents? Feu la implementació utilitzant un d'aquests mètodes (**Duplicats.java**):

```
static  void eliminaDuplicatsv0 (int [] vector)

static int [] eliminaDuplicatsv1(int [] vector)

static void eliminaDuplicatsv2(int [] vector1, int[] vector2)
```

7. Crea un programa que donada la seqüència de múltiples de 7 inferiors a 10000, calculi el nombre de vegades que surt el dígit 0 **(Nombre0.java)**. Per a resoldre-l, crea un mètode `nZeros` que retorni el nombre de 0's que té el nombre que rep per paràmetre. Un cop implementat el programa, pensa com hauria de canviar la declaració del mètode per a que es pugui utilitzar per a qualsevol dígit.

```
static int nZeros (int n)
```

8. Crea un programa que donada la seqüència de múltiples de 7 inferiors a 10000, calculi el nombre de vegades que surt cadascun dels dígits [0..9]. Per a resoldre-l, crea un mètode anomenat `quantsDigits()` que donat un nombre n i una taula de dígits, `taulaDigits`, que ens

retorni a la taula el nombre de vegades que surt cadascun dels dígits en aquest nombre. **(NombreDigits.java)**

```
static void quantsDigits (int n, int [] taulaDigits)
```

9. Problema de multiplicació de matrius:

a) Crea un programa que demani per pantalla un número **n**. A continuació es crearan dues matrius (dues taules de 2 dimensions) de **n** files x **n** columnes i es demanarà a l'usuari els números de les dues matrius per omplir-les. Finalment, el programa crearà una tercera matriu on guardarà el resultat de multiplicar les dues matrius inicials. Escriu un mètode que retorni la multiplicació de les matrius , m1 i m2, que rep com a paràmetres **(MultiplicaMatrius.java)**

```
static int [][] multiplica(int [][]m1, int [][]m2)
```

NOTA: L'algorisme per multiplicar dues matrius és el següent. Donades dues matrius A i B de 2 files i 2 columnes, tal que:

A = | a11 a12 | B = | b11 b12 |
| a21 a22 | | b21 b22 |

La matriu C resultant de AxB és:

C = | c11 c12 |
| c21 c22 |

On:

c11 = (Fila 1 de A) x (Columna 1 de B) = ((a11 x b11) + (a12 x b21))
c21 = (Fila 2 de A) x (Columna 1 de B) = ((a21 x b11) + (a22 x b21))
c12 = (Fila 1 de A) x (Columna 2 de B) = ((a11 x b12) + (a12 x b22))
c22 = (Fila 2 de A) x (Columna 2 de B) = ((a21 x b11) + (a22 x b21))

Penseu que aquest exemple és d'una matriu 2x2, en resoldre l'exercici heu d'aplicar la formulació general de multiplicació de matrius de **n**x**n**.

b) En aquest apartat has de resoldre el mateix exercici de multiplicació (que has fet abans fent servir un mètode `static`) però amb classes.

Crea la classe Matriu (**Matriu.java**) amb els atributs `int [] [] matriu, int nFiles, int nCols` i amb els següents mètodes d'objecte:

```
// Crea una matriu de f files i c columnes (constructor de la classe Matriu)
Matriu(int f, int c)

// Omple la matriu amb dades introduïts des de teclat
void ompleMatriu(Scanner sc)

// Mostra per pantalla la matriu m
public String toString()
```

Crea la classe GestioMatriu (**GestioMatriu.java**) on estarà el mètode `main()` i heu d'implementar el mètode estàtic `multiplicaMatrius`:

```
// Suposem que a i b es podem multiplicar perquè
// el nombre de columnes de a és igual al nombre de files de b. La matriu resultant es retorna
static Matriu multiplicaMatrius(Matriu a, Matriu b)
```

Des del mètode `main()` has de fer les crides necessàries als mètodes per a crear objectes de la classe Matriu, omplir-los, mostrar-los i finalment multiplicar les dues matrius. Com ja saps dues matrius `a` i `b` es poden multiplicar si el nombre de columnes de `a` és igual al nombre de files de `b`.

Recordeu que heu de crear una classe amb JUnit per a testejar els mètodes creats (**GestioMatriuTest.java**). Amb el testeig d'un parell de mètodes és suficient.

10. Es vol gestionar la informació de com a molt 25 bicicletes. Per allò es crea la classe **Bicicleta i TaulaBicicleta.** Una bicicleta té com atributs el *model*, *pes*, si te *suspensió*, i *preu* (**Bicicleta.java),** mètodes:

    //Constructor amb paràmetres
    `Bicicleta(String model, double pes, String s, double preu)`

    //Mètode per a mostrar una bicicleta (permet mostrar una bici fent us de System.out.println)
    `public String toString()`

    La classe **TaulaBicicleta** (**TaulaBicicleta.java**) conté dos atributs, *taulaBicicletes* que emmagatzema un conjunt de bicicletes (objectes de la classe Bicicleta), i *nBicis* que conté el nombre actual de bicicletes (inicialment 0). Aquesta classe també conté els següents mètodes:

    //Constructor de la classe
    `TauBicicleta(int max)`

    //Afegeix la bicicleta *b* a la taula de bicis
    `void afegirBici(Bicicleta b)`

    //Mostra totes les bicis que guarda la
    `void mostrarBicis()`

    //Retorna la bicicleta amb minim cost per kilo
    `Bicicleta getBiciMinimoCostePerKilo()`

    //Retorna la bicicleta amb pes inferior a *peso*
    `Bicicleta getBiciMinimoPes(int peso)`

    La classe **MenuBicicletes** (**MenuBicicletes.java)** conté la funció principal main() i mostra a l'usuari el següent menú:

    1. Afegir bicicleta.
    2. Mostrar bicis
    3. Obtenir el model de la bicicleta que tingui mínim cost per kilogram.
    4. Obtenir la bicicleta de pes inferior a un pes entrat per l'usuari, si és que n'hi ha alguna.
    5. Sortir.

    La opció 1 del menú sol·licita informació d'una bicicleta i afegeix aquesta a la taula de bicicletes.

    L'opció 2 del menú mostra totes les bicis que s'han afegit.

    L'opció 3 del menú mostra per pantalla el model de la bicicleta que té mínim cost per kilogram.

    L'opció 4, sol·licita a l'usuari un pes, i mostra la bicicleta que té un pes inferior a l'introduït. Si no hi ha una bicicleta amb pes inferior, es mostra un missatge dient-lo.

    Finalment, l'opció 5 tanca el programa.

11. Crea un programa per a gestionar un conjunt de paraules **(GestioParaules.java)**. Per això, defineix dues classes Paraula (**Paraula.java**) i TaulaParaules (**TaulaParaules.java**).

La classe **Paraula** conté els atributs *paraula* (String) que emmagatzema la paraula, i *nOcurr* (int) que indica el nombre d'ocurrències de la paraula.

La classe **TaulaParaules** conté dos atributs, *taulaParaules* que emmagatzema un conjunt de paraules (objectes de la classe Paraula), i *nParaules* que conté el nombre actual de paraules (inicialment 0).

El programa **GestioParaules** ha de permetre mitjançant un menú:

1. Afegir paraula a la taula.
2. Mostrar paraules de la taula.
3. Saber la paraula més freqüent de la taula.
4. Sortir.

Per exemple, si s'afegeixen les paraules: Porta, Llapis, Teclat i Llapis. L'opció mostrar paraules mostrarà:

```
Paraula NOcurr
------------------
Porta    1
Llapis   2
Teclat   1
```

i l'opció paraula més freqüent retornarà la paraula Llapis dient que té 2 ocurrències.

A la classe **GestioParaules** has d'implementar els següents mètode:

```
//Mostra un menú amb 4 opcions i retorna l'opció introduïda per l'usuari.
static int menu (Scanner sc)
```

A la classe **TaulaParaules** has d'implementar els mètodes:

```
//Afegeix la paraula p a la taula de paraules. Si ja estava la paraula a la taula, incrementa
//les ocurrències de la paraula.
void afegirParaula(Paraula p)

 //Mostra les paraules (i la seva freqüència).
void mostrarParaules()

//Retorna la paraula més freqüent de la taula.
Paraula paraulaMesFrequent()

//Busca la paraula pa a la taula de paraules, retorna l'índex de la paraula en la taula
// o -1 si no es troba la paraula a la taula.
int buscarParaula(Paraula pa)
```

Recordeu que heu de crear una classe amb JUnit per a testejar els mètodes creats a **TaulaParaulesTest.java**. Amb el testeig d'un parell de mètodes és suficient.

12. Crea un programa que gestioni les butaques de la sala d'un cinema. La classe **Cinema** (**Cinema.java**) té com atributs el nombre de files, *numFiles*, el nombre de butaques per fila, *numButaquesF*, i les butaques (*boolean [][] butaques*). La classe Cinema té els mètodes:

```
//Inicialment, totes les butaques del cinema han d'estar lliures.
Cinema (int numFiles, int numButaquesF)
```

// *fila* i *columna* indiquen la butaca a ocupar. Retorna -1 si la butaca no existeix, -2 si aquesta butaca ja estava ocupada, i 0 per a confirmar que la butaca s'ha ocupat correctament
`int ocuparButaca(int fila, int columna)`

// *fila* i *columna* indiquen la butaca a lliurar. Retorna -1 si la butaca no existeix, -2 si aquesta butaca ja estava lliure, i 0 per a confirmar que la butaca s'ha lliurat correctament
`int lliurarButaca(int fila, int columna)`

// *Posa totes les butaques com a lliures*
`void lliurarButaques()`

// *Mostra l'estat de totes les butaques*
`void mostrarButaques()`

Per exemple, per un cine amb 2 files i 2 butaques per fila, i amb totes les butaques lliures es mostraria:

```
Num butaca:     0       1
Fila 0:         L       L
Fila 1:         L       L
```

A la classe **GestioCinema.java** el programa demanarà a l'usuari el nombre de files de la sala (*numFiles*) i el nombre de butaques per cada fila (*numButaquesF*) del cinema, i cridarà al constructor de Cinema amb aquests valors.

A continuació, l'usuari podrà, mitjançant un **menú** (visualitzat mitjançant el mètode `menuOpcions`), fer les següents operacions:

1. Ocupar butaca.
2. Lliurar butaca.
3. Lliurar totes les butaques.
4. Mostrar butaques.
5. Sortir.

A l'opció de menú 1, el programa demanarà a l'usuari la fila i el nombre de butaca, i la posarà com a ocupada. Veure a dalt els paràmetres i retorn del mètode `ocuparButaca`.

A l'opció de menú 2, el programa demanarà fila i butaca i la posarà com a lliure. Veure avall els paràmetres i retorn del mètode `lliurarButaca`.

A l'opció de menú 3, el programa posarà totes les butaques com a lliures.

A l'opció de menú 4, el programa mostrarà l'estat, Lliure (L) o Ocupat (O), de totes les butaques.

Finalment, a l'opció 5 el programa acaba.

13. Modifica l'exercici anterior creant una nova classe **Sala** (**Sala.java**), que emmagatzema la informació relativa a les butaques d'una sala del cine. Ara la classe **Cinema** (**Cinema.java**) emmagatzemarà informació sobre les diferents sales d'un cine (amb un màxim de 10 sales). Crea un programa **(GestioCinema.java)** que permeti a l'usuari, mitjançant un menú, fer:

    1. Afegir sala al cine.
    2. Ocupar butaca d'una sala.
    3. Lliurar butaca d'una sala.
    4. Lliurar totes les butaques del cine.
    5. Mostrar totes les sales del cine.
    6. Sortir.

    Per a resoldre l'exercici, crea els mètodes (pensa a quina classe pertanyen, **Sala** o **Cinema**):

//Afegeix una sala al cine, *nombreF* i *nombreB* indiquen el nombre de files i nombre de butaques per fila de la sala, respectivament.
```
void afegirSala(int nombreF, int nombreB)
```

//Marca com ocupada una butaca d'una sala, *f* indica el nombre de fila de la butaca i *b* el nombre la butaca dintre de la fila. Retorna 1 en cas de haver-hi ocupat correctament la butaca i -1 en cas de estar-hi ja ocupada la butaca.
```
int ocuparButaca(int f, int b)
```

//Marca com ocupada una butaca d'una sala, *f* indica el nombre de fila de la butaca i *b* el nombre la butaca dintre de la fila. Retorna 1 en cas de haver-hi lliurat correctament la butaca i -1 en cas de estar-hi ja lliure la butaca.
```
int lliurarButaca(int f, int b)
```

//Marca com lliures totes les butaques d'una sala.
```
void lliurarTotes()
```

//Mostra les butaques d'una sala
```
void mostrarSala()
```

14. Defineix una classe Rectangle (**Rectangle.java**) amb els atributs *identificador*, *base*, i *alçada*, i amb els següents mètodes:

```
//constructor
Rectangle (int id, double base, double alcada)

//calcular l'àrea del rectangle.
double getArea()

//calcular el perímetre del rectangle.
double getPerimetre()
```

La classe **Rectangles** guarda un conjunt (una taula) de rectangles **(Rectangles.java)**. Fixeu-vos en els mètodes de la classe **TaulaBiclicleta** per a implementar els mètodes de la classe **Rectangles**. Llegiu també les descripcions de cadascuna de les opcions del menú que es troben avall.

Des de la classe **GestioRectangles** (**GestioRectangles.java)** es mostrarà un menú, on l'usuari podrà:

1. Afegir un nou rectangle a la taula de rectangles.
2. Modificar rectangle.
3. Eliminar rectangle.
4. Veure rectangle.
5. Sortir.

A l'opció 1 del menú es demanarà (des del main()) a l'usuari la mida de la base i l'alçada. A continuació es crearà un nou rectangle i es guardarà al conjunt de rectangles.

L'opció 2 del menú permet modificar un rectangle. El programa mostrarà el conjunt de rectangles (els seus identificadors) i a continuació demanarà l'identificador del rectangle a modificar. Per acabar, es demanarà la nova mida de la base i l'alçada del rectangle i guardarà la informació actualitzada.

L'opció 3 del menú permet eliminar un rectangle. El programa mostrarà el conjunt de rectangles (els seus identificadors) i a continuació demanarà l'identificador del rectangle a esborrar. Finalment s'esborrarà el rectangle del conjunt.

L'opció 4 del menú permet visualitzar un rectangle. El programa mostrarà el conjunt de rectangles (els seus identificadors) i a continuació demanarà l'identificador del rectangle a visualitzar. Per acabar es mostrarà tota la informació del rectangle demanat: base, alçada, àrea i perímetre.

L'opció 5 permet sortir del programa.

15. Defineix una classe **Cotxe** (**Cotxe.java**) amb els atributs *identificador (id)*, *marca*, *model*, *color*, *esllogat*, *preu* de lloguer per dia, i *nombre* de dies llogat. La classe **Lloguer (Lloguer.java)** guardarà un conjunt de cotxes i permetrà afegir o esborrar cotxes a aquest conjunt, llogar un cotxe o lliurar un cotxe llogat, i veure els cotxes disponibles i llogats.

    A continuació implementa un programa **(GestioCotxes.java)** per a gestionar una casa de lloguer de cotxes. Les opcions del menú seran:

    1. Afegir cotxe.
    2. Veure cotxes disponibles.
    3. Veure cotxes llogats.
    4. Esborrar cotxe.
    5. Llogar cotxe.
    6. Lliurar cotxe.
    7. Sortir

    L'opció 1 del menú permet afegir un cotxe. Demanarà les dades del nou cotxe i l'afegirà al conjunt de cotxes.

    L'opció 2 del menú permet veure cotxes disponibles. Mostrarà informació dels cotxes no llogats.

    L'opció 3 del menú permet veure cotxes llogats. Mostrarà informació dels cotxes llogats, mostrant per a cadascun d'ells els dies que està llogat.

    L'opció 4 del menú permet esborrar cotxe. Mostrarà informació dels cotxes disponibles, amb el seu *id* (els cotxes llogats no els podem eliminar). A continuació s'eliminarà el cotxe del conjunt.

    L'opció 5 del menú permet llogar cotxe. Mostrarà informació dels cotxes disponibles, cadascun amb el seu *id i* el seu preu de lloguer. A continuació es demanarà l'*id* del cotxe a llogar i el número de dies que es desitja llogar. Mostrarà el preu total del lloguer, i el cotxe quedarà com a llogat des d'aquell moment.

    L'opció 6 del menú permet lliurar cotxe. Mostrarà els cotxes llogats, cadascun amb el seu *id*. A continuació es demanarà l'*id* del cotxe a lliurar. El cotxe quedarà com a disponible des d'aquell moment.

    L'opció 7 permet sortir del programa.

16. Defineix una classe Monomi (**Monomi.java**) amb els atributs *coeficient* i *grau*. Implementa aquests mètodes de la classe:

    ```
    //Sumar amb un altre monomi.
    // Fixa't que aquest mètode retorna un nou monomi, resultat de sumar els dos monomis anteriors
    Monomi suma(Monomi monomiASumar)

    //Restar amb un altre monomi.
    //Fixa't que aquest mètode retorna un nou monomi, resultat de restar els dos monomis anteriors
    Monomi resta(Monomi monomiASumar)
    ```

    NOTA: Un monomi és de la forma ax^n, on a és el coeficient, x és el literal i n és el grau del monomi. Per exemple: $-3x^2$
    NOTA 2: Dos monomis només es poden sumar o restar si tenen el mateix grau.

    Defineix la classe **Polinomi**, que tindrà una llista de monomis. Contindrà els següents mètodes:

    ```
    //Afegir monomi.
    void afegir(Monomi monomi)

    //Obtenir grau del polinomi. Retorna el grau del polinomi, és a dir el grau del monomi amb grau més alt.
    int getGrau()
    ```

//Obtenir monomi de grau N
```
Monomi getMonomi(int grauMonomi)
```

//Sumar amb un altre polinomi.
//Fixa't que aquest mètode retorna un nou polinomi, resultat de sumar els dos polinomis anteriors
```
Polinomi suma(Polinomi polinomiASumar)
```

//Restar amb un altre polinomi.
//Fixa't que aquest mètode retorna un nou polinomis, resultat de restar els dos polinomis anteriors
```
Polinomi resta(Polinomi polinomiASumar)
```

Per acabar crea un programa (**GestioPolinomis.java**) que demani dos polinomis per pantalla i mostri els resultats de la seva resta i la seva suma. Per demanar un polinomi, s'ha de demanar primer el grau del polinomi (grauPolinomi) i s'ha d'anar demanant cada coeficient de cadascun dels termes.

Per exemple: Grau del polinomi? -> 3
Coeficient pel terme de grau 0? -> -3 (-3)
Coeficient pel terme de grau 1? -> 5 (5x)
Coeficient pel terme de grau 2? -> -6 ($-6x^2$)
Coeficient pel terme de grau 3? -> 8 ($8x^3$)

El polinomi resultant introduït és -> $8x^3 -6x^2 +5x -3$

17. Modifica l'exercici 10 (bicicletes) per a treballar amb atributs privats (`private`) i mètodes getters i setters publics (`public`).

18. Modifica l'exercici 11 (paraules) per a treballar amb atributs privats (`private`) i mètodes getters i setters publics (`public`).