

2. Tractament de cadenes de text

Fins ara, el tipus de dada caràcter no ha estat gaire rellevant a l'hora d'exposar exemples significatius de codi font de programes. És força habitual que en el vostre dia a dia tracteu amb conjunts de dades numèriques individuals, cadascuna independent de les altres: dates, distàncies, intervals de temps, quantitats, etc. Per tant, són dades que té sentit emmagatzemar i processar mitjançant un programa per automatitzar-ne el càlcul. En canvi, segurament no és gaire habitual que constantment useu conjunts de caràcters individuals i independents: lletres ‘a’, ‘b’, ‘c’, etc. La veritable utilitat dels caràcters és poder usar-los agrupats en forma de paraules o frases: un nom, una pregunta, un missatge, etc. Per tant, si un llenguatge de programació ha de ser útil, s'ha de tenir en compte aquesta circumstància.

El tipus de dada compost **cadena de text** o **String** serveix per representar i gestionar de manera senzilla una seqüència de caràcters.

String

En Java, el tipus de dada compost cadena de text s'identifica amb la paraula clau **String**.

2.1 La classe "String"

En llenguatge Java, la manipulació de qualsevol dada de tipus compost, incloent-hi les cadenes de text, té un seguit de particularitats i una terminologia força diferent de la dels tipus primitius o els *arrays*. Si bé no és necessari conèixer-ne tots els detalls, abans de veure com manipular cadenes de text o altres tipus de dades compostos és important aprendre un seguit de termes específics, ja que apareixen amb freqüència a la documentació de Java.

Lús dels termes classe i objecte està estretament lligat a la característica de llenguatge Java de ser orientat a objectes.

En Java, exceptuant el cas de l'*array*, s'usa el terme **classe** per referir-se a un tipus de dada compost.

A les variables de tipus de dades compostos també els podem assignar directament valors, com a les de tipus primitius. Ara bé, amb algunes particularitats en la nomenclatura.

S'usa el terme **objecte** per referir-se al valor assignat a una variable d'un tipus de dades compost.

En llenguatge Java, la classe que representa una cadena de text s'anomena explícitament **String**. Així, doncs, a partir d'ara, en lloc de parlar del “tipus de dada compost **String**”, direm sempre “la classe **String**”, i en lloc de dir que

Per convenció de codi, els noms de les classes al Java s'escriuen sempre usant **UpperCamelCase**.

una variable “és del tipus de dada compost `String`” es dirà que “és de la classe o que pertany a la classe `String`”. Per norma general, també es parlarà de “l’objecte assignat a la variable `holaMon`” en lloc d’“el valor assignat a la variable `holaMon`”, o d’“un objecte de la classe `String`” en lloc d’“un valor de tipus `String`”.

En realitat, els conceptes de classe i objecte van molt més enllà del que s’ha exposat aquí, però això ja ens val per poder explicar com manipular cadenes de text de manera que se segueixi estrictament la terminologia Java tal com s’usa a les seves fonts de referència.

2.1.1 Inicialització d’objectes “String”

Treballar amb classes no modifica en absolut la sintaxi per declarar una variable, però sí que pot tenir algunes implicacions a l’hora d’inicialitzar-la, ja que cal assignar un objecte. Com es representa un objecte pot variar dependent de la classe emprada. Afortunadament, en el cas de la classe `String`, la sintaxi per declarar i inicialitzar una variable és idèntica a com es faria amb un tipus primitiu qualsevol, ja que és possible representar un objecte mitjançant un literal directament. Els literals assignable a una variable de la classe `String` prenen la forma de qualsevol combinació de text envoltada entre cometes dobles, “...”. Un exemple de declaració i inicialització d’una variable que conté una cadena de text seria:

¹ `String holaMon = "Hola, món!";`

Aquesta instrucció declara una variable anomenada `holaMon`, pertanyent a la classe `String`, i li assigna l’objecte que representa la cadena de text “Hola, món!”.

Val la pena comentar que hi ha el concepte de cadena de text “buida”, és a dir, que no conté absolutament cap caràcter. Es representa simplement no posant res entre les cometes dobles. Expressat amb codi seria:

¹ `String cadenaBuida = "";`

2.1.2 Manipulació d’objectes

A diferència dels tipus primitius, qualsevol objecte emmagatzemat en una variable pertanyent a una classe (és a dir, qualsevol valor emmagatzemat en una variable d’un tipus de dada compost), no es pot manipular directament mitjançant operadors de cap tipus. No hi ha operadors que manipulin o comparin directament objectes.

En alguns casos, la restricció en l’ús d’operacions ja pot resultar evident de manera intuïtiva. Aquest és el cas dels operadors aritmètics o lògics. Per exemple, amb quin resultat avaluaria una divisió de cadenes de text (objectes de la classe `String`) o la seva negació lògica? Ara bé, en el cas dels operadors relacionals, tot i que

sí que pot tenir sentit intentar comparar dues cadenes de text, usar-los també és incorrecte. Per tant, les expressions següents no són vàlides.

- `unString * unAltroString`
- `unString && unAltroString`
- `unString < unAltroString`
- `unString == unAltroString`
- etc.

En el cas de la igualtat cal tenir cura especial, ja que el compilador de Java no ho considera un error de sintaxi.

En Java, la manipulació d'objectes es fa mitjançant la **invocació de mètodes** sobre les variables en què es troben emmagatzemats.

Podeu trobar la llista de mètodes de la classe String en el web
<http://download.oracle.com/javase/6/docs/api/java/lang/String.html>.

La llista de mètodes que poden ser invocats sobre una variable on s'emmagatzema un objecte depèn exclusivament de la seva classe. Classes diferents permeten la invocació de diferents mètodes. Per exemple, la classe String especifica el conjunt de mètodes que és possible invocar sobre una variable on hi ha assignada una cadena de text.

Per generar una instrucció en què es crida un mètode, s'usa l'identificador de la variable, seguida d'un punt, i llavors el nom del mètode que volem, escollit entre la llista que ofereix la classe a la qual pertany la variable. Depenent del mètode utilitzat, pot ser necessari indicar un seguit d'informació addicional, en forma de llista de valors entre parèntesis, separats per comes (`..., ..., ...`). Aquesta llista és el que s'anomenen els seus **paràmetres**. Si el mètode no requereix cap paràmetre, només cal obrir i tancar parèntesis, `()`. La sintaxi seria:

¹ `identificadorVariable.nomMètode(paràmetre1, paràmetre2, ..., paràmetreN);`

Per convenció de codi, els noms dels mètodes de les classes del Java estan escrits sempre usant *lowerCamelCase*.

Escriure una invocació d'un mètode sobre una variable és equivalent a escriure una expressió que avalua un cert resultat. Cada mètode diferent fa una operació o transformació diferent sobre l'objecte emmagatzemat en aquella variable. Per saber què fa cadascun i a quin tipus de dada pertany el seu resultat, no queda més remei que mirar la documentació de Java.

La invocació d'un mètode té la màxima precedència en avaluar una expressió.

La màxima precedència d'una invocació d'un mètode té especialment sentit en aquest cas, ja que fins que el seu resultat concret no s'ha avaluat, no es disposa de cap valor a partir del qual es pugui continuar avaluant una expressió més complexa. Per exemple, en el codi següent, en què s'usa el mètode `length()`, que avalua la mida d'un objecte de la classe String, el resultat és 43, per l'ordre de precedència

dels operadors. Fins que no s'ha avaluat `length()` i s'ha esbrinat que la mida del text és 22, ni tant sols seria possible iniciar el càlcul de la multiplicació. Això evidencia que la seva precedència ha de ser la màxima.

```
1 public class Precedencia {  
2     public static void main (String[] args) {  
3         //El text té una mida de 22 caràcters.  
4         String text = "Hi havia una vegada...";  
5         //ordre: 1) mètode, 2) multiplicació, 3) resta.  
6         int dobleMidaMenysUn = 2 * text.length() - 1;  
7         System.out.println(dobleMidaMenysUn);  
8     }  
9 }
```

2.1.3 Accés als caràcters d'una cadena de text

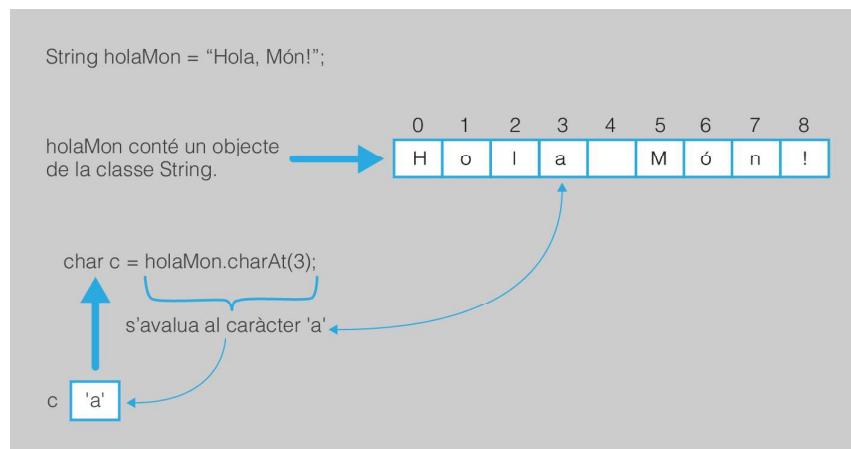
Per la manera com organitzen els conjunts de caràcters que contenen, les cadenes de text són molt semblants als *arrays*. En el fons, no deixen de ser una seqüència de tipus primitius, en aquest cas caràcters (`char`), en què cada lletra ocupa una posició concreta. Per tant, molts dels aspectes generals aplicables als *arrays* en el Java també es poden aplicar a les cadenes de text. Novament, l'accés es fa per l'índex de la posició que ocupa cada caràcter, començant des de 0 fins a la seva llargària - 1.

Ara bé, atès que les cadenes de text s'emmagatzem en Java en variables de la classe `String` en forma d'objectes, l'accés o manipulació de les seves dades s'ha de fer mitjançant la invocació d'algun dels mètodes definits en la classe `String`. Per al cas de l'accés a caràcters individuals, el mètode que cal usar és l'anomenat `charAt(numPosició)`. Aquest mètode requereix un únic paràmetre de tipus enter en què s'indica quina posició cal consultar, i avaluва igual el caràcter en aquella posició.

Per exemple, veure quin caràcter és a la tercera posició d'una cadena de text es faria de la manera següent. Fixeu-vos que la invocació d'un mètode és equivalent a avaluar una expressió:

```
1 ...  
2 String holaMon = "Hola, món!";  
3 char c = holaMon.charAt(3);  
4 //A la variable "c" hi ha el caràcter 'l'  
5 System.out.println("A la posició 3 hi ha el caràcter " + c);  
6 ...
```

La figura 2.1 mostra el comportament de la invocació en el mètode `charAt`. Fixeu-vos que, en el fons, aquest és equivalent a avaluar una expressió. El seu resultat pot ser immediatament assignat a una variable, o usat en una altra expressió.

FIGURA 2.1. Avaluació de la invocació del mètode charAt

Com amb els *arrays*, la posició també ha de ser un valor vàlid o es produirà un error. Per evitar això, en Java les cadenes de text també incorporen un mecanisme per establir quina és la seva llargària, similar a l'atribut `length` dels *arrays*. Es tracta del mètode `length()`. Alerta, fixeu-vos que, en tractar-se d'un mètode sense paràmetres, immediatament al final porta dos parèntesis, `()`. En el cas dels *arrays*, aquests parèntesis no s'usen.

Per exemple, el codi següent escriu en línies diferents els caràcters individuals de la cadena de text “Hola, món!”. Proveu que és així.

```

1 //Mostra els caràcters individuals en una cadena de text.
2 public class MostraCaracters {
3     public static void main (String[] args) {
4         String holaMon = "Hola, món!";
5         //Es recorren les posicions de la cadena de text una per una.
6         for (int i = 0; i < holaMon.length(); i++) {
7             System.out.println(holaMon.charAt(i));
8         }
9     }
10 }
```

Finalment, heu de tenir en compte que un punt molt important de l'ús dels mètodes és el fet que aquests **es comporten exactament igual que expressions**. Per tant, no és possible fer assignacions sobre aquests. En el cas de les variables de la classe `String` pot ser fàcil confondre's, ja que el seu comportament és molt semblant als *arrays*, però no del tot. Tot i que si partiu del funcionament d'un *array* podria semblar lògic fer el següent, en realitat és un error. Seria com si, operant amb valors enters, intentéssiu assignar un valor al resultat d'una expressió suma:

```

1 ...
2 holaMon.charAt(3) = 'b'
3 //Erroni. No és realment igual que fer "array[3] = ...;".
4 //Seria més aviat com intentar fer "(3 + 4) = a;".
5 ...
```

De fet, la classe String de Java no ofereix cap mètode per canviar el valor del caràcter en una posició determinada. **Repte 1:** feu un programa que mostri per pantalla una cadena de text, però escrita del revés. Per exemple, si es parteix de la cadena “Hola, món!”, per pantalla ha de mostrar ”!nóm ,aloH”.

2.1.4 Concatenació de cadenes de text

Si bé s'ha dit que els objectes no es poden manipular mitjançant operadors, únicament mitjançant la invocació de mètodes, això no és del tot cert. Hi ha un únic cas en què sí que és possible aplicar l'operació de suma entre objectes. Es tracta de la concatenació de cadenes de text (objectes de la classe `String`). De fet, aquest és un cas realment molt excepcional, ja que, si us hi fixeu, també es permet fer aquesta operació entre cadenes de text i dades de diferents tipus primitius, cosa que en teoria tampoc està permesa. Cap altra mena d'objectes permet en Java aquesta mena de comportament.

El resultat de la concatenació amb cadenes de text sempre dóna com a resultat una nova cadena de text, que es pot usar tant per generar missatges mostrats directament per pantalla com per fer assignacions a variables de la classe `String`. Per tant, recordeu que es pot fer:

```
1 //Mostra el resultat d'una divisió simple entre reals.  
2 public class Dividir{  
3     public static void main(String[] args) {  
4         double dividend = 18954.74;  
5         double divisor = 549.12;  
6         double resultatDivisio = dividend/divisor;  
7         //S'assigna el resultat de la concatenació a una variable de la classe  
8         //String.  
9         String missatge = "El resultat obtingut és " + resultatDivisio + ".";  
10        System.out.println(missatge);  
11    }  
12}
```

Cal que tingueu present que us trobeu davant una excepció, incorporada pels creadors del Java per facilitar la vostra tasca com a programadors a l'hora de crear missatges de text.

2.1.5 "Arrays" de cadenes de text

De la mateixa manera que es poden declarar i usar *arrays* de tipus primitius, també es poden declarar i usar *arrays* d'objectes, com les cadenes de text. La sintaxi en aquest aspecte no varia en absolut, si bé heu de tenir en compte que la part de la instrucció vinculada a la definició del tipus ara indica el nom de la classe. Per exemple, per declarar un *array* amb capacitat per a 5 cadenes de text es faria de la manera següent, segons si s'usa el mecanisme amb valors concrets o per defecte:

```
1 //Inicialització a valor concret.  
2 //Un literal d'una cadena de text es representa amb text entre cometes dobles.  
3 String[] arrayCadenaConcret= {"Valor1", "Valor2", "Valor3", "Valor4", "Valor5"  
4     "};  
5 //Inicialització a valor per defecte.  
6 String[] arrayCadenaDefecte = new String[5];
```

El cas de la inicialització a valors per defecte té una particularitat, i és que les posicions prenen un valor especial que indica que la posició està “buida”, anomenat `null`. Alerta, ja que aquest valor és un cas especial i no és equivalent a una cadena buida o sense text (representada amb el literal ””). Mentre una posició és “buida”, es compleix que:

- Si es mostra per pantalla, apareix el text: “null”.
- Qualsevol invocació d'un mètode sobre seu dóna un error.

Per accedir a una posició d'un `array` d'objectes, la sintaxi és igual que per als tipus primitius, usant un índex entre claus (`[i]`). Tot i així, els objectes tenen la particularitat de permetre la invocació de mètodes. En aquest aspecte, una posició es comporta com una variable qualsevol. Això es pot veure en l'exemple següent, especialment en el segon bucle, en què s'invoca el mètode `length()` directament sobre la cadena de text que hi ha a cada posició de l'`array`. Compileu-lo i executeu-lo per esbrinar què fa:

```
1 public class ArrayString {  
2     public static void main(String[] args) {  
3         String[] text = {"Hi", "havia", "una", "vegada..."};  
4         System.out.println("El text de l'array és:");  
5         for (int i = 0; i < text.length; i++) {  
6             System.out.println("Posició " + i + ": " + text[i]);  
7         }  
8         System.out.println("Les seves mides són:");  
9         for (int i = 0; i < text.length; i++) {  
10            System.out.println("Posició " + i + ": " + text[i].length());  
11        }  
12    }  
13 }
```

2.2 Entrada de cadenes de text

De la mateixa manera que pot ser necessari introduir dades de tipus primitius en el programa, de manera que s'hi pugui treballar, també pot ser necessària l'entrada de cadenes de text. Per al cas de l'entrada de cadenes de text, però, teniu dues possibilitats.

D'una banda, es pot usar el teclat tal com heu fet fins ara. De fet, aquesta és la finalitat més natural per a la qual s'empra el teclat, ja que es tracta d'un perifèric especialment dissenyat per escriure text. Per fer-ho cal que useu les instruccions de lectura que ja heu après anteriorment, però de manera que sigui possible llegir tant paraules individuals com frases de text completes.

D'una altra banda, hi ha una possibilitat nova: l'entrada mitjançant arguments en el mètode principal. Sempre que s'executa un programa al vostre ordinador és possible incloure directament un seguit d'arguments addicionals en forma de text. Si bé en els sistemes operatius moderns, basats en entorns gràfics de finestres, aquesta circumstància queda oculta, aquesta opció està present. Java permet

esbrinar si en executar el vostre programa s'han inclòs aquests arguments i, si és el cas, quins són.

2.2.1 Lectura des de teclat

El procés de lectura de cadenes de text des de teclat és molt semblant a la lectura de tipus primitius. Només cal saber quines són les instruccions adients per obtenir cadenes de text des de teclat, en aquest cas en forma d'objectes de la classe `String`. Abans de veure quines són aquestes instruccions, però, val la pena fer una petita revisió dels elements que intervenen en el procés de lectura de dades per teclat aprofitant que s'han introduït els conceptes de classe, objecte i mètode, sota una nova perspectiva.

Revisió de la lectura per teclat

Torneu a fer una ullada a un exemple simple de lectura de dades per teclat. Per exemple, un programa que llegeix un valor enter i el mostra per pantalla, comprovant abans que el tipus del valor escrit sigui el correcte.

```
1 import java.util.Scanner;
2 //Llegeix un enter, comprovant que sigui correcte.
3 public class LecturaEnter {
4     public static void main (String[] args) {
5         Scanner lector = new Scanner(System.in);
6         int valor = 0;
7         System.out.print("Escriu un valor enter: ");
8         if (lector.hasNextInt()) {
9             valor = lector.nextInt();
10            System.out.println("El valor era " + valor + ".");
11        } else {
12            lector.next();
13            System.out.print("El valor no era enter.");
14        }
15        lector.nextLine();
16    }
17 }
```

Aquest codi té les particularitats següents:

- `Scanner` és una classe. Aquesta ofereix mètodes per llegir dades des del teclat.
- `lector` és una variable en què s'emmagatzema un objecte pertanyent a aquesta classe.
- `hasNextInt()`, `nextInt()` i `next()` són mètodes, oferts per la classe `Scanner`. De fet, totes les instruccions `next...` i `hasNext...` vinculades a la lectura de dades pel teclat són mètodes d'aquesta classe. En invocar-los, provoquen que es faci una lectura des del teclat i avaluen la dada llegida.
`Segons el tipus de dada que es vol llegir, cal usar un mètode o un altre.`

L'únic punt que encara és una incògnita ara mateix és com funciona la inicialització d'una variable de la classe Scanner, de manera que contingui un objecte. No és necessari que l'entengueu, simplement n'hi ha prou de saber que per a aquesta classe es fa així. Ara bé, és important tenir en compte que sense una correcta inicialització de la variable lector amb un objecte la lectura de dades no serà correcta.

Lectura de paraules individuals

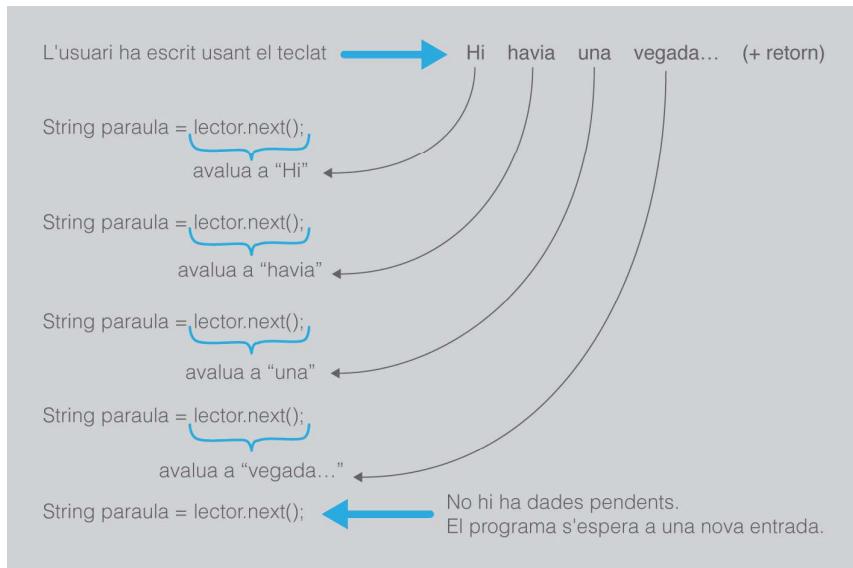
Igual que fins ara s'han llegit tipus primitius individuals entrats per teclat, també és possible llegir paraules.

El mètode de la classe Scanner vinculat a la lectura d'una cadena de text composta d'una única paraula és next().

El seu comportament és exactament el que heu vist fins al moment per llegir tipus primitius, amb l'única diferència que la invocació d'aquest mètode sobre un objecte de la classe Scanner avalua una cadena de text. Recordeu que si en una mateixa línia escriviu més d'una paraula, successives invocacions a aquest mètode no bloquejaran el programa, sinó que aniran avaluant les successives dades pendents de llegir. Aquest comportament queda esquematitzat a la figura 2.2.

Cal tenir en compte que, en aquest context, quan es parla de *paraula*, en realitat s'està referint a qualsevol combinació de lletres o nombres que no conté cap espai enmig. Per tant, aquest mètode pot llegir els textos individuals “Hola,”, “món!”, “1024”, “3.1426”, “base10”, etc. Ara bé, el resultat d'una lectura usant aquest mètode sempre avalua una cadena de text, mai un tipus numèric. Fins i tot quan el valor llegit és un nombre.

FIGURA 2.2. Lectura de seqüències de paraules



Un fet remarcable en el cas de l'entrada de cadenes de text des de teclat és que no cal fer cap comprovació de tipus prèvia a la lectura (mètodes `hasNext...`) , ja que tota dada escrita mitjançant el teclat sempre és pot interpretar com una cadena de text. Mai no es pot produir una errada en aquest sentit.

La millor manera de veure'n el comportament és amb un exemple. Compileu i executeu el programa següent, que llegeix fins a 10 paraules i les va mostrant per pantalla. Proveu d'escriure més d'una paraula a cada línia (una frase sencera) i observeu com les llegeix totes una per una abans d'esperar novament una entrada de dades. Fixeu-vos que les paraules queden discriminades en la lectura, en estar separades amb espais.

```
1 import java.util.Scanner;
2 //Llegeix un cert nombre de paraules (en aquest cas, 10).
3 public class LecturaParaules {
4     public static final int NUM_PARAULES = 10;
5     public static void main (String[] args) {
6         Scanner lector = new Scanner(System.in);
7         System.out.println("Escriu " + NUM_PARAULES + " paraules separades per
8             espais.");
9         System.out.println("Les pots escriure en línies de text diferent, si vols
10             .");
11         //Es van llegint paraules una per una.
12         //Recordar el comportament lectura de seqüències de dades pel teclat.
13         String paraula = lector.next();
14         System.out.println("Paraula " + i + ": Has escrit \" " + paraula + " \".");
15         //Es llegeix la resta de la línia i s'ignora el contingut.
16         lector.nextLine();
17     }
18 }
```

Lectura de caràcters individuals

Hi ha casos en què només es vol llegir un caràcter, per tal de simplificar l'entrada de dades. Per exemple, com a sistema abreujat per escollir opcions d'un menú, o en un programa en què es donin diferents opcions etiquetades amb lletres individuals (com podria ser resoldre una pregunta de tipus test). En un cas com aquest, i atès que la classe `Scanner` no ofereix cap mètode per llegir caràcters individuals, el que cal fer és llegir l'entrada com una cadena de text, comprovar que aquesta només es compon d'un únic caràcter i extreure'l.

- **Per llegir una cadena de text hi ha el mètode `next()`, com tot just s'ha vist.**
- **Per establir la llargària d'una cadena de text es disposa del mètode `length()`. En aquest cas, cal veure si és 1.**
- **Per extreure un caràcter individual, tenim el mètode `charAt(numPosició)`. En aquest cas, la posició per consultar és la 0.**

En el programa següent d'exemple cal escollir la resposta correcta entre quatre opcions etiquetades amb una lletra. Per simplificar el codi, només es permet un sol intent. Executeu-lo i observeu atentament cadascuna de les passes tot just descrites per veure si s'ha llegit realment una única lletra i extreure-la.

```
1 import java.util.Scanner;
2 //Mostra una pregunta tipus test i mira si s'endevina.
3 public class LecturaCaracter{
4     public static final char RESPOTA_CORRECTA = 'b';
5     public static void main (String[] args) {
6         Scanner lector = new Scanner(System.in);
7         System.out.println("Endevina la pregunta.");
8         System.out.println("Quin dels següents no és un tipus primitiu?");
9         System.out.println("a) Enter");
10        System.out.println("b) Scanner");
11        System.out.println("c) Caràcter");
12        System.out.println("d) Booleà");
13        System.out.print("La teva resposta és l'opcíó: ");
14        //Es llegeix la cadena de text.
15        String paraula = lector.next();
16        //És una paraula d'un únic caràcter?
17        if (paraula.length() == 1) {
18            //S'extreu el caràcter de la cadena de text.
19            char caracter = paraula.charAt(0);
20            //És un caràcter vàlid? (a, b, c o d)
21            if ((caracter >= 'a')&&(caracter <= 'd')) {
22                //La resposta final és correcta?
23                if (caracter == RESPOTA_CORRECTA) {
24                    System.out.println("Efectivament, la resposta era '" +
25                        RESPOTA_CORRECTA + "'");
26                } else {
27                    System.out.println("La resposta '" + caracter + "' és incorrecta.");
28                }
29            } else {
30                System.out.println("'" + caracter + "' és una opció incorrecta.");
31            }
32        } else {
33            //No ho era.
34            System.out.println("'" + paraula + "' no és un caràcter individual.");
35        }
36        lector.nextLine();
37    }
}
```

Repte 2: modifiqueu el programa anterior per permetre fins a tres intents de resposta en lloc d'un.

Lectura de frases senceres

Fins al moment, només ha estat possible llegir elements individuals, un per un i separats per espais, dins d'una seqüència entrada per teclat. Ara bé, sovint és útil llegir frases senceres, compostes per un conjunt de paraules separades per espais. Per exemple, entrar un nom i cognoms, o una adreça postal. En aquest cas, voldríem desar tota la cadena de text dins d'una única variable.

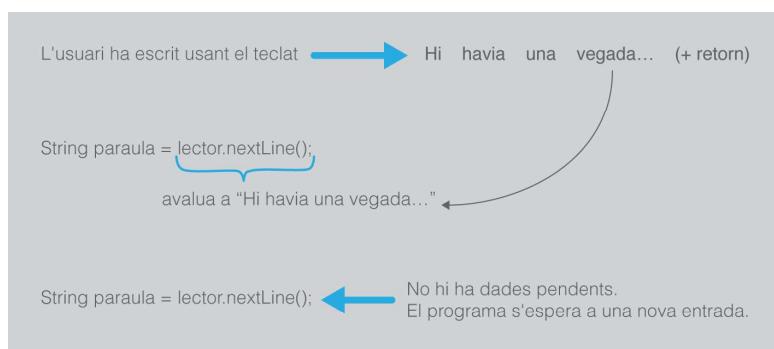
El mètode de la classe Scanner vinculat a la lectura d'una cadena de text en forma de frase en què hi ha diverses paraules separades per espais és nextLine().

Fins al moment, aquest mètode ha estat usat per descartar tota la llista d'elements pendents de llegir del teclat (en el cas que s'hagi escrit més d'un element en una única línia). En realitat, aquest mètode el que fa és llegir tots els elements pendents de llegir a la línia actual i avaluar la cadena de text que els conté tots. Per tant,

serveix per llegir conjunts d'elements de cop, independentment dels espais en blanc que els separen.

A efectes pràctics amb vista a la lectura de cadenes de text, si mai no es combina l'ús de `nextLine()` amb altres mètodes de lectura d'elements individuals (`next()`, `nextInt()`, etc.) és possible usar-lo per llegir frases senceres directament. Cada cop que s'invoca, el programa es bloqueja i espera que l'usuari entri una frase sencera i pitgi la tecla de retorn. Llavors, avalua tota la frase escrita (totes les paraules amb espais inclosos). En aquest cas, i al contrari que amb la resta de mètodes de lectura, no és possible escriure més d'un element dins una mateixa línia de text, ja que llegeix tota la línia al complet des de la darrera lectura fins a trobar un salt de línia. Aquest comportament s'esquematitza a la figura 2.3. Contrasteu-lo amb el mostrat a la figura 2.2.

FIGURA 2.3. Lectura de frases



El programa d'exemple següent mostra com es llegeix una línia de text completa escrita pel teclat, de manera molt semblant a l'exemple anterior de llegir paraules individuals. Tot i així, fixeu-vos que hi ha algunes diferències en el codi i en el comportament. Concretament, mai no es donarà el cas que una lectura pel teclat no ocasioni el bloqueig del programa a l'espera que escriviu quelcom. Absolutament sempre s'aturarà a esperar que escriviu un text i pitgeu la tecla de retorn. Proveu el programa per comprovar que és així.

```
1 import java.util.Scanner;
2 //Llegeix frases escriptes pel teclat.
3 public class LecturaFrase {
4     public static final int NUM_FRASES = 4;
5     public static void main (String[] args) {
6         Scanner lector = new Scanner(System.in);
7         System.out.println("Escriu " + NUM_FRASES + " frases.");
8         System.out.println("Per acabar una frase, pitja la tecla de retorn.");
9         for (int i = 0; i < NUM_FRASES; i++) {
10             //Es van llegint frases una per una.
11             String frase = lector.nextLine();
12             System.out.println("Frase " + i + ": Has escrit \"" + frase + "\".");
13         }
14         //Ara no cal llegir la resta de cap línia, ja que sempre es llegeixen
15         //línies senceres...
16     }
17 }
```

2.2.2 Argument del mètode principal

Aquesta opció permet establir una entrada de dades al programa sense haver d'interrompre'n l'execució per haver de preguntar quelcom a l'usuari i esperar una entrada per teclat.

Una pregunta que us podeu plantejar és quina és la utilitat d'aquest sistema quan ja hi ha l'entrada pel teclat. Un exemple de situació en què això és molt útil és en programes que es comportaran de manera diferent segons una entrada de dades, però en els quals no es vol, o no es pot, esperar que l'usuari estigui pendent. Per exemple, suposeu un programa que voleu que s'executi cada nit per fer còpies de seguretat d'un seguit de fitxers. D'una banda, no té sentit que el programa estigui sempre en marxa comprovant si ha arribat l'hora de fer la còpia. És més lògic programar-ne l'execució periòdica amb alguna eina oferta pel sistema operatiu. Ara bé, és evident que no és factible que algú estigui pendent del teclat cada vegada que s'executa, però perquè funcioni correctament ha de disposar de certa informació que tampoc no pot estar escrita dins el codi font, ja que el valor dependrà de cada usuari que l'utilitzi: la llista de directoris que es vol copiar, on es desa la còpia, etc. Per tant, és útil disposar d'una manera simple d'entrar dades en un programa sense que hagi de ser sempre pel teclat.

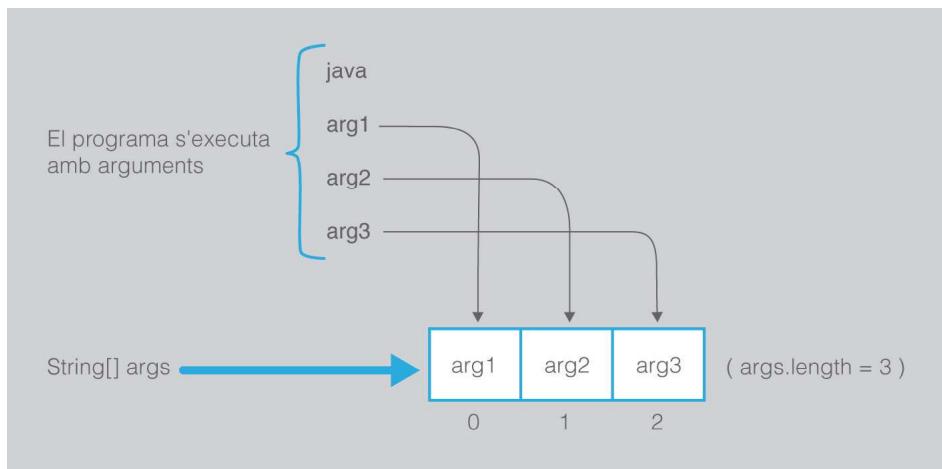
Exemples d'eines que permeten l'execució diferida de programes són les tasques programades en el Windows o la instrucció cron de Unix.

Normalment, la llista d'arguments d'un programa s'introdueix com un conjunt de paraules a continuació del nom del fitxer executable, separades per espais.

La llista de paraules que conformen els arguments d'un programa en Java es pot obtenir directament a partir de la variable especial que es defineix en la declaració del mètode principal: `String[] args`. Concretament, pren la forma d'un array de cadenes de text, en què aquestes ocupen les posicions en el mateix ordre que s'han escrit els arguments en executar el programa, tal com mostra la figura 2.4.

Disposeu d'un annex en què us expliquem com cal incloure arguments en executar un programa mitjançant l'IDE que usareu al llarg del curs.

FIGURA 2.4. Assignació d'arguments d'entrada en l'execució del programa “java” a la variable “args”



Executar un programa amb arguments

Normalment, els exemples bàsics de programes executats amb arguments són els vinculats a la línia d'ordres. Entre aquests, es pot trobar l'intèpret del Java mateix, l'executable java (en sistemes Unix) o java.exe (en sistemes Windows). Si obriu una finestra de línia d'ordres i hi escriviu java -help, apareix una llista dels arguments possibles per executar-lo sense la necessitat d'un IDE: els fitxers amb *bytecode* Java que ha d'executar, si cal mostrar-ne la versió, si ha de mostrar informació de depuració addicional, etc. De fet, "-help" és un argument en si mateix.

El vostre IDE automatitza l'execució i el pas d'arguments a aquest programa sense la necessitat d'haver-vos de preguntar res. Partint dels fitxers que heu generat, ell sol genera la llista d'arguments que cal usar.

Per saber amb quants arguments s'ha executat el programa cal usar l'atribut length associat als *arrays*. Si no s'ha passat cap argument, el seu valor serà 0. Aneu amb compte amb aquesta situació, ja que llavors qualsevol accés a una posició de l'*array* serà sempre incorrecte.

A mode d'exemple, proveu el programa següent, que enumera els arguments amb els quals s'ha executat. A efectes pràctics, es tracta d'un recorregut sobre un *array*, si bé els elements emmagatzemats a les seves posicions ara són cadenes de text en lloc de valor numèrics.

```
1 //Programa que escriu per pantalla els arguments d'un programa.  
2 public class LecturaArguments {  
3     //Els arguments estan a la variable "args".  
4     public static void main(String[] args) {  
5         //Primer cal mirar si n'hi ha algun.  
6         if (args.length > 0) {  
7             //N'hi ha. Es mostren per pantalla ordenadament.  
8             for (int i = 0; i < args.length; i++) {  
9                 System.out.println("Argument " + i + ": " + args[i]);  
10            }  
11        } else {  
12            //No n'hi ha cap.  
13            System.out.println("No hi ha cap argument.");  
14        }  
15    }  
16}
```

2.3 Manipulació de cadenes de text

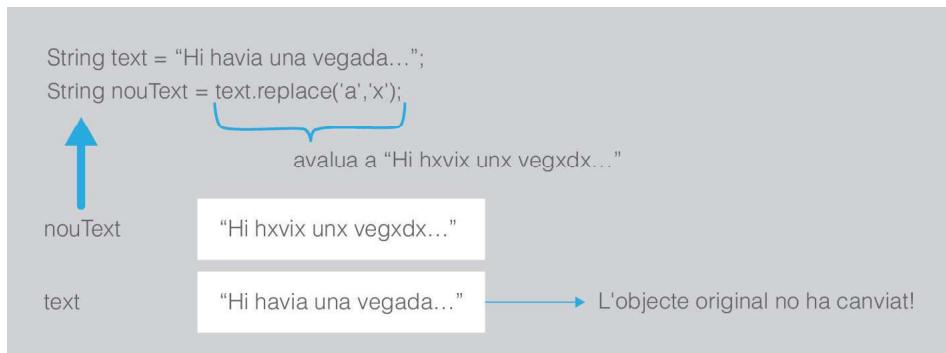
Tot i que una cadena de text organitza els caràcters que emmagatzema d'una manera molt semblant a com ho faria un *array*, en forma de seqüència ordenada en què cada element ocupa una posició identificada per un índex, hi ha dues diferències destacades en el seu comportament i que tenen gran incidència a l'hora de manipular-les.

D'una banda, la classe String ofereix un ventall molt ampli de mètodes amb els quals es poden manipular, de manera que en la immensa majoria de casos no caldrà fer un tractament individual de cada posició (per exemple, mitjançant estructures de repetició). Normalment, tots els esquemes d'ús i manipulació de cadenes de text es basen simplement en la invocació del mètode adient.

D'una altra banda, una altra particularitat que les diferencia dels *arrays* és que són immutables. La invocació d'un mètode pot consultar les dades dins d'una cadena de text, o pot generar una nova cadena de text amb certes modificacions respecte de l'original. Però mai no es veurà modificat el valor inicial de la cadena de text original. Si volem, aquesta nova cadena pot ser assignada a la variable original, de manera que es reemplaça el valor inicial pel nou.

La figura 2.5 mostra aquest comportament per la invocació del mètode `replace(caracterNou, caracterVell)`, que reemplaça totes les aparicions d'un caràcter (`caràcterVell`) per un altre de diferent (`caràcterNou`).

FIGURA 2.5. Els mètodes invocats sobre cadenes de text no en modifiquen el valor, en creen de noves modificades



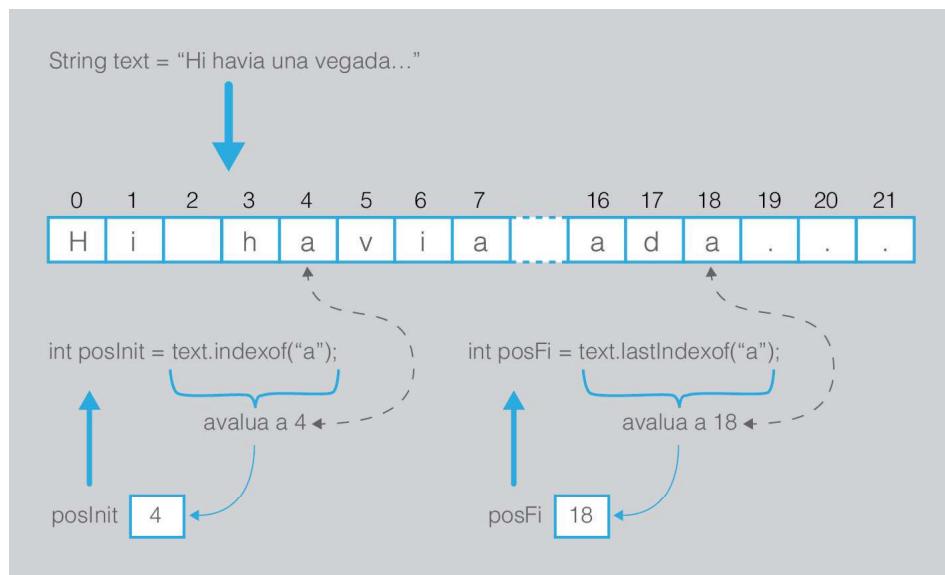
2.3.1 Cerca

Normalment, el procés de cerca d'un valor concret dins d'un *array* s'hauria de fer amb una estructura de repetició que anés comprovant si aquest valor existeix en alguna de les posicions, fins a trobar-lo o arribar al final de l'*array*. Això es podria replicar amb les cadenes de text usant el mètode `charAt`. Per sort, la classe `String` ofereix dos mètodes que estalvien aquesta tasca, ja que avaluen directament la posició on es troba un caràcter o una subcadena de text concreta.

- `indexOf(textACercar)` avalia la primera posició dins de la cadena de text on es troba el text per cercar.
- `lastIndexOf(textACercar)` avalia la darrera posició dins de la cadena de text on es troba el text per cercar.

En tots dos casos, `caràcterACercar` sempre ha de ser una cadena de text (ja sigui mitjançant un literal o una variable) i si el text cercat no existeix, avaluen a -1. Per tant, no solament és possible saber si existeix el valor cercat, sinó també establir directament posicions on es pot trobar.

La figura 2.6 mostra el comportament dels dos mètodes si se cerca el caràcter 'a' a la frase "Hi havia una vegada...".

FIGURA 2.6. Comportament dels mètodes “indexOf” i “lastIndexOf”

El codi següent mostra un exemple de cerca d'un caràcter dins d'un text qualsevol (compost per diverses paraules). Compileu-lo, proveu-lo i estudieu-ne el comportament.

```
1 import java.util.Scanner;
2
3 //Cerca un caràcter concret dins d'una cadena de text qualsevol.
4 public class CercaCaracter {
5
6     public static void main(String[] args) {
7
8         Scanner lector = new Scanner(System.in);
9
10        System.out.println("Escriu una línia de text qualsevol i pitja retorn:");
11        String text = lector.nextLine();
12
13        System.out.println("Quin caràcter vols cercar? ");
14        String charText = lector.next();
15        lector.nextLine();
16
17        char charCerca = charText.charAt(0);
18
19        int posInit = text.indexOf(charCerca);
20        int posFi = text.lastIndexOf(charCerca);
21        if (posInit > -1){
22            System.out.println("Les aparicions del caràcter '" + charCerca + "' són
23 :");
24            System.out.println("Primer cop - " + posInit);
25            System.out.println("Darrer cop - " + posFi);
26        } else {
27            System.out.println("Aquest caràcter no es troba al text.");
28        }
29    }
30
31 }
```

Repte 3: feu un programa que, en un seguit de cadenes de text passades com a arguments al mètode principal, compti quantes contenen el caràcter ‘a’.

2.3.2 Comparació

Una tasca que es fa molt sovint amb cadenes de text és veure si un text entrat per l'usuari correspon a una opció concreta entre les disponibles. Això és especialment freqüent en el cas d'entrada de dades via arguments del mètode principal, ja que en molts programes aquests arguments indiquen opcions per a l'execució.

Malauradament, els objectes no accepten cap mena d'operació entre ells, i això inclou aquelles vinculades als operadors relacionals. Per tant, no es poden comparar directament tampoc, tal com es faria entre valors de tipus primitius. També cal usar algun mètode disponible a la classe a la qual pertanyen. Si la seva classe no ofereix cap mètode que us ajudi a comparar, llavors és impossible comparar-los.

La classe `String` ofereix diferent mètodes per comparar cadenes de text.

- `equals(textPerComparar)` avalua un valor booleà (`true/false`) segons si la cadena de text continguda a la variable en què s'invoca i el text usat com a paràmetre són idèntics o no. El text usat com a paràmetre pot ser tant un literal, `"..."`, com una altra variable en què hi hagi emmagatzemada una altra cadena de text.

`equalsIgnoreCase(text)`

Aquest altre mètode fa la mateixa funció, però ignorant majúscules i minúscules.

Com a exemple, es pot comprovar si els arguments passats a un programa corresponen a alguna de les opcions possibles. En el programa següent, és possible detectar si els arguments corresponen a algun dels acceptats: `-version`, `-help` i `-server`. Per a cada cas, es fa una opció diferent. Si hi ha algun argument que no és cap d'aquests tres, el programa informa que hi ha un error en els arguments. Proveu que és així.

```
1 //Comprova si els arguments es corresponen amb un conjunt preestablert.
2 public class ComprovaArguments {
3     public static void main (String args[]) {
4         //No oblideu mai comprovar si l'array conté arguments!
5         if (args.length > 0) {
6             //Es va recorrent l'array i es mira quin valor hi ha.
7             for (int i = 0; i < args.length; i++) {
8                 if (args[i].equals("-version")) {
9                     System.out.println("S'ha usat l'argument \"-version\"");
10                } else if (args[i].equals("-help")) {
11                    System.out.println("S'ha usat l'argument \"-help\"");
12                } else if (args[i].equals("-server")) {
13                    System.out.println("S'ha usat l'argument \"-server\"");
14                } else {
15                    System.out.println("L'argument \"" + args[i] + "\" no és vàlid!");
16                }
17            }
18        } else {
19            System.out.println("No hi havia cap argument.");
20        }
21    }
22 }
```

De vegades pot ser útil comparar dues cadenes de text des del punt de vista de l'ordre alfabètic, és a dir, de quina seria la seva posició relativa en un diccionari.

Aquest tipus de comparació seria equivalent als operadors relacionals “major que” ($>$) i “menor que” ($<$).

compareToIgnoreCase(text)

Aquest mètode és l'equivalent ignorant majúscules i minúscules.

- `compareTo(textPerComparar)` es comporta de manera semblant a `equals`, però en aquest cas avalua a 0, qualsevol valor positiu o qualsevol valor negatiu depenent de si la cadena emmagatzemada en la variable en què s'invoca el mètode és alfabèticament igual, major o menor (respectivament) que la usada com a paràmetre.

La millor manera de veure-ho és amb un exemple. En el programa següent cal endevinar un text secret partint del fet que, per a cada resposta, el programa us diu com a pista quina és la posició relativa de la paraula secreta segons el seu ordre alfabètic. Reflexioneu sobre quines diferències hi ha entre aquest codi, que tracta cadenes de text, i un programa que fes exactament el mateix amb valors de tipus enter (en aquest cas, per ordre numèric).

```
1 import java.util.Scanner;
2 //Joc d'endevinar una paraula, donant pistes del seu ordre alfabètic.
3 public class EndevinaParaula {
4     //La paraula per endevinar és "objecte".
5     public static final String PARAULA_SECRETA = "java";
6     public static void main (String[] args) {
7         Scanner lector = new Scanner(System.in);
8         System.out.println("Comencem el joc.");
9         System.out.println("Endevina el valor de la paraula del diccionari.");
10        boolean haEncertat = false;
11        while (!haEncertat) {
12            System.out.print("Quina paraula creus que és? ");
13            String paraulaUsuari = lector.next();
14            lector.nextLine();
15            int posicio = paraulaUsuari.compareTo(PARAULA_SECRETA);
16            if (posicio < 0) {
17                //La paraula de l'usuari és anterior a la secreta.
18                System.out.println("Has fallat! La paraula va després...");
19            } else if (posicio > 0) {
20                //La paraula de l'usuari és posterior a la secreta
21                System.out.println("Has fallat! La paraula va abans...");
22            } else {
23                //Si val 0, és que s'ha encertat.
24                haEncertat = true;
25            }
26        }
27        System.out.println("Enhorabona. Has encertat!");
28    }
29 }
```

Repte 4: usant el mètode `compareTo`, ordeneu un *array* de cadenes de text. Per facilitar l'entrada de les dades, aquest *array* pot estar determinat directament a partir d'arguments en l'execució del programa (la variable `args`).

2.3.3 Creació de subcadenes

El mètode `charAt` permet l'extracció de dades d'una cadena de text, però només de caràcters individuals. Tot i així, l'aplicabilitat d'aquest mecanisme és limitada.

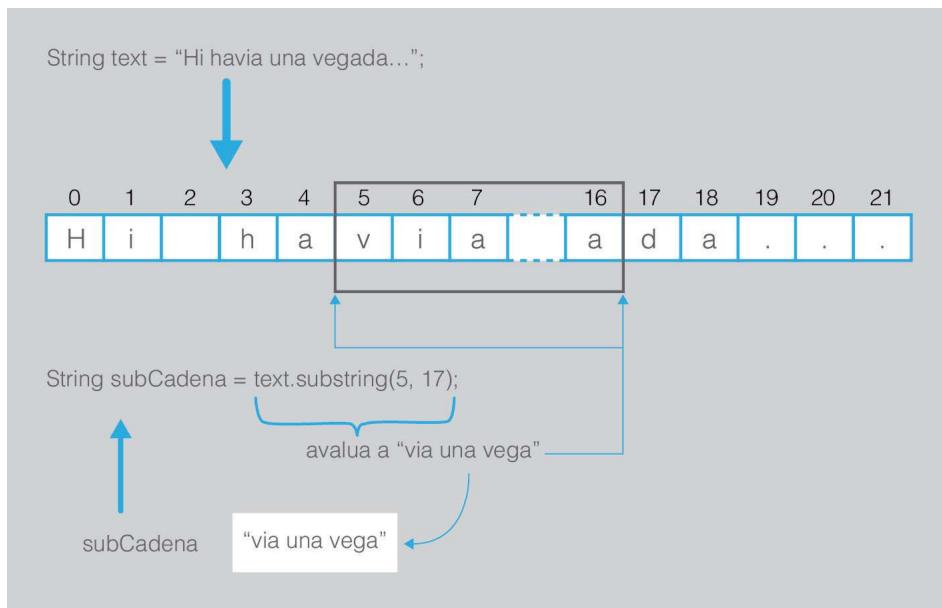
Molt sovint us trobareu que, en realitat, us resultaria més còmode poder extreure cadenes de text completes contingudes dins d'una altra cadena. Per exemple, parts d'una frase o paraules individuals. La classe String ofereix dos mètodes que permeten fer això.

- `substring(posicióInicial, posicióFinal)` evalua una nova cadena de text composta pels caràcters que van des de `posicióInicial` fins a `posicióFinal - 1`.

Heu d'anar amb compte en usar aquest mètode, ja que els valors han d'estar dins del rang permès. Si algun supera el valor de les posicions que ocupen els caràcters dins de la cadena de text, es produirà un error.

La figura 2.7 mostra un esquema del comportament del mètode `substring` si s'extreu una subcadena de la frase “Hi havia una vegada...”.

FIGURA 2.7. Comportament del mètode “substring”



Com a exemple del funcionament de la invocació d'aquest mètode, tot seguit trobareu el codi d'un programa que extreu la primera i la darrera paraula d'una frase i les mostra per pantalla. Això és equivalent a mostrar una subcadena de text entre el primer i el darrer espai en blanc. Si hi ha menys de tres paraules, llavors no mostra res.

```
1 import java.util.Scanner;  
2 //Un programa que extreu tot el text d'una frase, excepte la primera i la  
3 //darrera paraula.  
4 public class ExtreureParaules {  
5     public static void main (String[] args) {  
6         Scanner lector = new Scanner(System.in);  
7         System.out.println("Escriu una frase de text i pitja retorn:");  
8         String text = lector.nextLine();  
9         //Cerca el primer i el darrer espai en blanc.  
10        int iniciSubcadena = text.indexOf(' ');  
11        int fiSubcadena = text.lastIndexOf(' ');  
12        System.out.println("El text sense la primera i la darrera paraula és:");  
13        if (iniciSubcadena == fiSubcadena) {  
            //0 bé no hi ha espais (els dos mètode avaluen a -1).
```

```
14     //O bé només hi ha dues paraules (els dos mètodes avaluen la mateixa
15     //posició).
16     //No es mostra res.
17     System.out.println("(No hi ha res per escriure...)");
18 } else {
19     //Es mostra la cadena intermèdia.
20     //La segona paraula comença una posició després del primer espai en blanc
21
22     //La darrera paraula comença una posició després del darrer espai en blanc.
23     String textFinal = text.substring(iniciSubcadena + 1, fiSubcadena);
24     System.out.println(textFinal);
25 }
```

Repte 5: feu un programa que mostri per pantalla cadascuna de les paraules individuals d'una frase en línies diferents. Per fer-ho, abans haureu d'anar esbrinant les posicions on hi ha espais en blanc per poder delimitar on comença i acaba cada paraula.

Per obtenir les paraules d'una frase, no cal cercar on hi ha espais i anar extraient el text entre aquests. Hi ha un altre mètode que facilita molt més la feina.

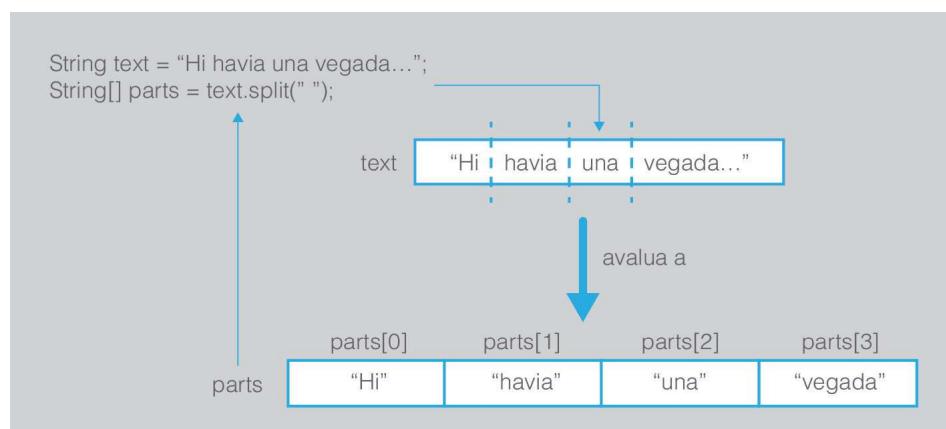
Recordeu que el contingut de la cadena original mai no es modifica.

- `split(textSeparació)` avalia un *array* de cadenes de text, en què en cada posició hi ha el resultat de fer el següent. S'agafa la cadena de text original i se separa en trossos usant com a divisor `textSeparació`. Cada tros, una nova cadena de text, s'ubica en una posició de l'*array*. El valor de `textSeparació` ha de ser una cadena de text.

Per exemple, si com a text de separació d'aquest mètode s'usa un espai en blanc, " ", una frase es dividiria en paraules. El resultat seria un *array* de cadenes de text en què en cada posició hi ha cadascuna de les paraules. Ara bé, el separador pot ser qualsevol text que vulgueu (comes, punts, combinacions de lletres, etc.).

La figura 2.8 mostra un esquema del comportament d'aquest mètode si es divideix la frase "Hi havia una vegada—" usant com a text de separació l'espai en blanc, de manera que queda dividida en paraules.

FIGURA 2.8. Comportament del mètode "split"



Com a exemple, estudieu el codi font següent, que pertany un programa per invertir l'ordre de les paraules d'una frase. Proveu en el vostre entorn de treball que és així.

```
1 import java.util.Scanner;
2 //Programa que inverteix l'ordre de les paraules d'una frase.
3 public class InverteixOrdreParaules {
4     public static void main(String[] args) {
5         Scanner lector = new Scanner(System.in);
6         System.out.println("Escriu una línia de text qualsevol i pitja retorn:");
7         String text = lector.nextLine();
8         //Dividim la frase en paraules, separades per espais.
9         String[] paraules = text.split(" ");
10        System.out.println("El text invertit és:");
11        //Imprimim les paraules en ordre invers.
12        for (int i = paraules.length - 1; i >= 0 ; i--) {
13            System.out.print(paraules[i]);
14            //El mètode split elimina el text de separació (en aquest cas, els espais
15            )
16            System.out.print(" ");
17        }
18        System.out.println();
19    }
}
```

Repte 6: feu un programa que generi automàticament acrònims. Partint d'una frase determinada, ha de mostrar per pantalla el text compost per les inicials de cada paraula individual que forma la frase.

2.3.4 Transformacions entre cadenes de text i tipus primitius

En molts casos, una part important dels programes es basa en les operacions fetes sobre dades de tipus primitius. Ara bé, amb vista a la interacció amb l'usuari, ja sigui tan per mostrar-les com per llegir-les, és necessari operar amb cadenes de text. Per tant, un mecanisme útil és el de la transformació de tipus primitius en cadenes de text i viceversa.

Conversió de tipus primitiu a cadena de text

El cas més senzill és la transformació d'una dada de tipus primitiu a un objecte de la classe `String`. Si els valors que es volen representar formen part d'una frase o una cadena de text més llarga (per exemple: “El resultat és ”, i el valor), tan sols cal usar l'operació suma, com s'ha explicat a l'apartat 1.1.5. Java s'encarrega de fer la conversió de tipus automàticament si en algun dels operands dins d'una concatenació hi ha alguna dada de tipus primitiu. Per al cas de mostrar directament un valor per pantalla, la instrucció `System.out.println` també transforma directament qualsevol tipus primitiu que se li proporcioni en una cadena de text que serà mostrada per pantalla.

Si només volem convertir un valor directament, sense que aquest hagi d'estar inclòs dins d'una frase o amb l'únic objectiu de mostrar-lo per pantalla posteriorment, llavors es pot usar la instrucció `String.valueOf(valor)`. On posa valor es pot usar qualsevol dada corresponent a qualsevol tipus primitiu, ja sigui una

variable, un literal o una expressió. Aquesta instrucció evalua el seu equivalent a cadena de text.

Ateses les capacitats innates de la concatenació i la impressió per pantalla per transformar dades de tipus primitius en cadenes de text automàticament, els casos on resulta aplicable l'ús de la instrucció `String.valueOf(valor)` és limitat. En la immensa majoria dels casos, si voleu convertir un tipus primitiu en cadena de text serà per mostrar-lo per pantalla, ja sigui sol o dins d'una frase. Tot i així, hi pot haver casos en què tractar una dada com a cadena de text en lloc de com a valor numèric pot facilitar la tasca del programador. Per exemple, quan es vol tractar aquest valor com una seqüència de díigits individuals.

Suposeu un programa que mostra els díigits en ordre invers d'un nombre real de certa llargària, amb qualsevol nombre de decimals. Penseu-hi una mica i veureu que, tractant-lo com a valor numèric i usant operacions entre reals, no és un problema la resolució del qual resulti evident. Aquest és un cas en què l'algorisme per resoldre el problema és una mica més senzill si el valor per processar es tracta com una seqüència de caràcters que no pas si es tracta com un nombre. Només caldria fer un recorregut per cadascun dels caràcters individuals (les xifres) usant una estructura de repetició. Per tant, és útil transformar-lo en una cadena de text i processar-lo com a tal.

Proveu en el vostre entorn el codi font, mostrat tot seguit:

```
1 //Mostra les xifres d'un nombre real en ordre invers.
2 public class InverteixOrdre {
3     public static void main (String[] args) {
4         float numero = 3.1415926535f;
5         //El convertim en una cadena de text.
6         String numeroText = String.valueOf(numero);
7         //Recorrem els seus díigits un per un, començant pel final.
8         for (int i = numeroText.length() - 1; i >= 0; i--) {
9             //Imprimim cada dígit individual (inclusiu el punt decimal).
10            System.out.print(numeroText.charAt(i));
11        }
12    }
13 }
```

Conversió de cadena de text a tipus primitiu

Tot i que ja sabeu com cal llegir dades dels tipus primitius directament des del teclat, de vegades, partint d'una cadena de text, us interessaria transformar-la a un tipus primitiu (normalment, un valor numèric). Per exemple, aquest pot ser el cas d'un programa no interactiu que obté les dades d'entrada per al pas d'arguments al mètode principal. En aquest cas, sempre es parteix de valors en format cadena de text i, per tant, cal fer una conversió si algun dels arguments és un valor numèric amb el qual cal operar.

Per a aquest cas, Java disposa d'un seguit d'instruccions, una per a cada tipus primitiu, que serveix per fer aquesta conversió. Ara bé, com passa amb les instruccions de lectura de tipus primitius per teclat, cal garantir que la cadena de text per convertir és correcta, ja que en cas contrari el programa donarà un error.

Per exemple, no és possible convertir a un valor de tipus enter les cadenes de text

“2,67334”, “1a88”, “true”, etc. Aquestes instruccions s’enumeren a la taula 2.1. On diu `text` cal posar un objecte de tipus `String`, ja sigui mitjançant una variable, un literal o una concatenació de cadenes.

TAULA 2.1. Instruccions per convertir cadenes de text en tipus primitius

Instrucció	Tipus de dada convertit
<code>Byte.parseByte(text)</code>	<code>byte</code>
<code>Integer.parseInt(text)</code>	<code>int</code>
<code>Double.parseDouble(text)</code>	<code>double</code>
<code>Float.parseFloat(text)</code>	<code>float</code>
<code>Long.parseLong(text)</code>	<code>long</code>
<code>Short.parseShort(text)</code>	<code>short</code>

Per exemple, suposeu un programa que escriu una cadena de text un cert nombre de vegades. Per fer-ho, es basa en dos arguments en el mètode principal. El primer és la cadena de text per escriure, i el segon el nombre de vegades (un valor enter). Observeu-ne el codi, proveu-lo i executeu-lo en el vostre entorn. Observeu també què passa si el segon argument no és un valor de tipus enter.

```
1 //Mostra una cadena de text N vegades, partint d'arguments d'entrada.
2 public class MostraNVegades {
3     public static void main (String[] args) {
4         //Es mira si hi ha els 2 arguments que corresponen.
5         if (args.length != 2) {
6             System.out.println("El nombre d'arguments no és correcte!");
7         } else {
8             //Hi ha 2 arguments, el segon és el nombre de vegades.
9             int numVegades = Integer.parseInt(args[1]);
10            //Fem una estructura de repetició. Cal mostrar el primer argument N
11            //vegades.
12            for (int i = 0 ; i < numVegades; i++) {
13                System.out.println(args[0]);
14            }
15        }
16    }
```

Repte 7: feu un programa que calculi la divisió entre dos valors reals, fent que aquests valors siguin entrats com a arguments en el mètode principal.

2.4 Solucions dels reptes proposats

Repte 1:

```
1 public class InverteixCadena {  
2     public static void main (String[] args) {  
3         String holaMon = "Hola, món!";  
4         //Es recorren les posicions de la cadena de text una per una.  
5         for (int i = holaMon.length() - 1; i >= 0; i--) {  
6             System.out.print(holaMon.charAt(i));  
7         }  
8         System.out.println();  
9     }  
10 }
```

Repte 2:

```
1 import java.util.Scanner;  
2 public class LecturaCaracterTresIntents {  
3     //El nombre d'intents es declara com una constant.  
4     public static final int INTENTS = 3;  
5     public static final char RESPUESTA_CORRECTA = 'b';  
6     public static void main (String[] args) {  
7         Scanner lector = new Scanner(System.in);  
8         System.out.println("Endevina la pregunta.");  
9         System.out.println("Quin dels següents no és un tipus primitiu?");  
10        System.out.println("a) Enter");  
11        System.out.println("b) Scanner");  
12        System.out.println("c) Caràcter");  
13        System.out.println("d) Booleà");  
14        //Cal un comptador d'intents.  
15        int intents = 0;  
16        //Cal saber si s'ha encertat (semàfor).  
17        boolean encertat = false;  
18        //Es pregunta mentre no s'encerti ni s'esgotin els intents.  
19        while ((intents < INTENTS)&&(!encertat)) {  
20            System.out.print("La teva resposta és l'opció: ");  
21            //Es llegeix la cadena de text.  
22            String paraula = lector.next();  
23            //És una paraula d'un únic caràcter?  
24            if (paraula.length() == 1) {  
25                //S'extreu el caràcter de la cadena de text.  
26                char caracter = paraula.charAt(0);  
27                //És un caràcter vàlid? (a, b, c o d).  
28                if ((caracter >= 'a')&&(caracter <= 'd')) {  
29                    //La resposta final és correcta?  
30                    if (caracter == RESPUESTA_CORRECTA) {  
31                        System.out.println("Efectivament, la resposta era '" +  
32                            RESPUESTA_CORRECTA + "'.");  
33                        encertat = true;  
34                    } else {  
35                        System.out.println("La resposta '" + caracter + "' és incorrecta.")  
36                        ;  
37                        intents++;  
38                    }  
39                } else {  
40                    System.out.println("'" + caracter + "' és una opció incorrecta.");  
41                    //Si hi ha aquest error, no s'esgotarà cap intent...  
42                }  
43            } else {  
44                //No ho era.  
45                System.out.println("\'" + paraula + "\' no és un caràcter individual.")  
46                ;  
47                //Si hi ha aquest error, no s'esgotarà cap intent...  
48            }  
49        }
```

```
46     lector.nextLine();
47 }
48 if (intents >= INTENTS ) {
49     System.out.println("Has esgotat tots els teus intents.");
50 }
51 }
52 }
```

Repte 3:

```
1 public class ComptaArgumentsA {
2     public static void main(String[] args) {
3         //Un comptador de paraules amb 'a'.
4         int numAs = 0;
5         //Cal recórrer l'array d'arguments.
6         for (int i = 0; i < args.length; i++) {
7             //Es mira si hi ha alguna lletra 'a' a la cadena de text.
8             if (-1 != args[i].indexOf("a")) {
9                 numAs++;
10            }
11        }
12        System.out.println("El nombre d'arguments amb el caràcter 'a' és " + numAs)
13        ;
14    }
}
```

Repte 4:

```
1 public class OrdenaArguments {
2     public static void main (String[] args) {
3         for (int i = 0; i < args.length - 1; i++) {
4             for(int j = i + 1; j < args.length; j++) {
5                 if (args[i].compareTo(args[j]) > 0) {
6                     String canvi = args[i];
7                     args[i] = args[j];
8                     args[j] = canvi;
9                 }
10            }
11        }
12        //El mostrem per pantalla.
13        System.out.println("Les cadenes ordenades són:");
14        for (int i = 0; i < args.length;i++) {
15            System.out.println(args[i]);
16        }
17    }
18 }
```

Repte 5:

```
1 import java.util.Scanner;
2 public class ObtenirParaules {
3     public static void main(String[] args) {
4         Scanner lector = new Scanner(System.in);
5         System.out.println("Escriu una frase de text i pitja retorn:");
6         String text = lector.nextLine();
7         boolean fi = false;
8         //Mitjançant un bucle, s'anirà escurçant el text paraula per paraula.
9         do {
10             //La primera paraula va de l'inici al primer espai.
11             int primerEspai = text.indexOf(" ");
12             //No hi ha espais. No hi ha cap paraula més per escriure.
13             if (primerEspai == -1) {
14                 System.out.println(text);
15                 fi = true;
16             } else {
17                 //Hi ha un espai. Extreure paraula i mostrar-la.
18                 String paraula = text.substring(0, primerEspai);
19                 System.out.println(peula);
20                 //S'elimina la paraula del text, de manera que s'escurça.
21                 text = text.substring(primerEspai + 1, text.length());
22             }
23         } while (!fi);
24     }
25 }
```

Repte 6:

```
1 import java.util.Scanner;
2 public class GeneradorAcronims {
3     public static void main(String[] args) {
4         Scanner lector = new Scanner(System.in);
5         System.out.println("Escriu una línia de text qualsevol i pitja retorn:");
6         String text = lector.nextLine();
7         //Dividim la frase en paraules, separades per espais.
8         String[] paraules = text.split(" ");
9         //Recorrem les paraules i escrivim cada inicial.
10        for(int i = 0; i < paraules.length; i++) {
11            System.out.print(paraules[i].charAt(0));
12        }
13        System.out.println();
14    }
15 }
```

Repte 7:

```
1 public class DividirArgumentsReals {
2     public static void main(String[] args) {
3         //Està bé comprovar si el nombre d'arguments és correcte.
4         if (args.length != 2) {
5             System.out.println("El nombre d'arguments és incorrecte.");
6         } else {
7             //Conversió de tipus.
8             double dividend = Double.parseDouble(args[0]);
9             double divisor = Double.parseDouble(args[1]);
10            double resultat = dividend/divisor;
11            System.out.println("El resultat és " + resultat);
12        }
13    }
14 }
```