NF3 TIPUS DE DADES COMPOSTES

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.

La classe String

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 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".

Inicialitzacio 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 depenent 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 assignables 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:

```
String holaMon = "Hola, món!";

String cadenaBuida = "";
```

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 * unAltreString
- unString && unAltreString
- unString < unAltreString
- unString == unAltreString
- 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 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.

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

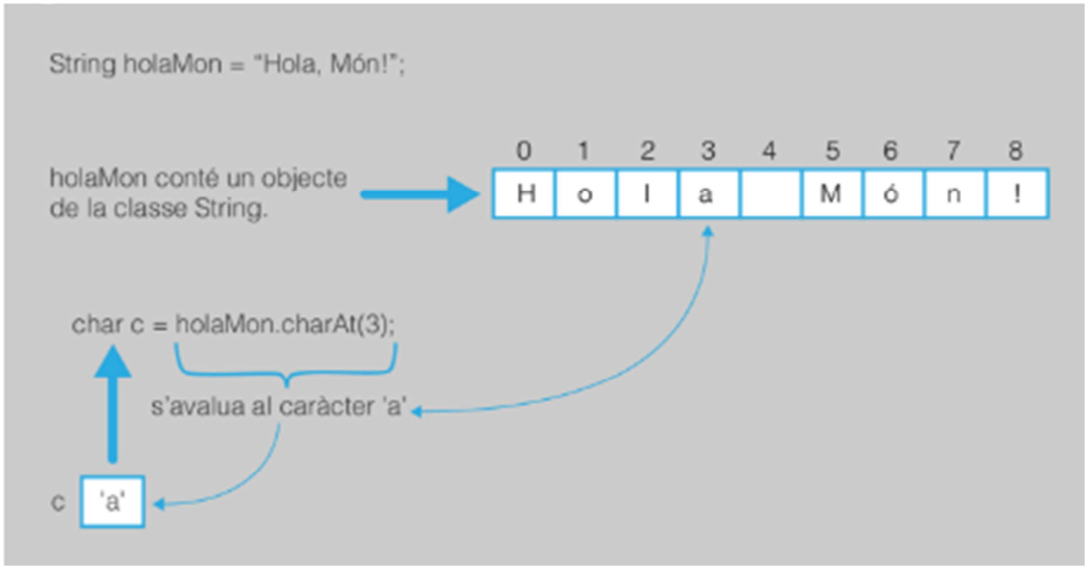
Accés als caràcters d'una cadena de text

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 i fins a la seva llargària − 1.

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 avalua 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:

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



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

```
public class MostraCaracters {
	public static void main(String[] args) {
		String holaMon = "Hola, món!";
		// Es recorren les posicions de la cadena de text una per una.
		for (int i = 0; i < holaMon.length(); i++) {
			System.out.println(holaMon.charAt(i));
		}
	}
}
```

Concatenació

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:

```
public class Dividir {
	public static void main(String[] args) {
		double dividend = 18954.74;
		double divisor = 549.12;
		double resultatDivisio = dividend / divisor;
		// S'assigna el resultat de la concatenació a una variable de la
		// classe string.
		String missatge = "El resultat obtingut és " + resultatDivisio + ".";
		System.out.println(missatge);
		}
}
```

També disposem del mètode concat() de la classe string que ens fa el mateix. Observa però que no modifica les cadenes sino que ens retorna una nova cadena amb el resultat de concatenar les dues cadenes.

```
String cad1="Hola ";
String cad2="mon";
cad1.concat(cad2);
System.out.println(cad1.concat(cad2));
System.out.print("Observació:" + cad1);
```

"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:

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

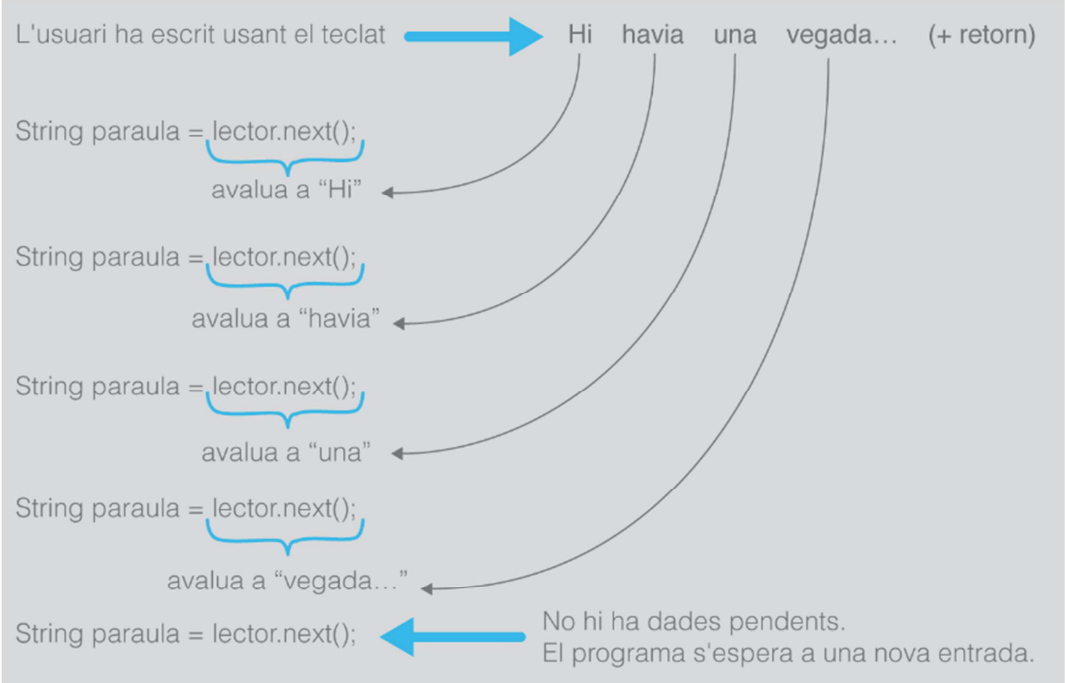
Per accedir a una posició d'un array d'objectes, la sintaxi és igual que per als tipus primitius, usant un índex entre claus ([i]). Tot i així, els objectes tenen la particularitat de permetre la invocació de mètodes. En aquest aspecte, una posició es comporta com una variable qualsevol. Això es pot 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:

```
public class ArrayString {
	public static void main(String[] args) {
		String[] text = { "Hi", "havia", "una", "vegada..." };
		System.out.println("El text de l'array és:");
		for (int i = 0; i < text.length; i++) {
			System.out.println("Posició " + i + ": " + text[i]);
		}
		System.out.println("Les seves mides són:");
		for (int i = 0; i < text.length; i++) {
			System.out.println("Posició " + i + ": " + text[i].length());
		}
	}
}
```

Lectura de paraules individuals

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.



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.

```
import java.util.Scanner;

//Llegeix un cert nombre de paraules (en aquest cas, 10).
public class LecturaParaules {
        public static final int NUM_PARAULES = 10;

        public static void main(String[] args) {
                Scanner lector = new Scanner(System.in);
                System.out.println("Escriu " + NUM_PARAULES + " paraules separades per espais.");
                System.out.println("Les pots escriure en línies de text diferent, si vols.");
                for (int i = 0; i < NUM_PARAULES; i++) {
                        // Es van llegint paraules una per una.
                        // Recordar el comportament lectura de seqüències de dades pel
                        // teclat.
                        String paraula = lector.next();
                        System.out.println("Paraula " + i + ": Has escrit \"" + paraula + "\".");
                }
                // Es llegeix la resta de la línia i s'ignora el contingut.
                lector.nextLine();
        }
}
```

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.

```
import java.util.Scanner;

//Mostra una pregunta tipus test i mira si s'endevina.
public class LecturaCaracter {
	public static final char RESPOSTA_CORRECTA = 'b';

	public static void main(String[] args) {
		Scanner lector = new Scanner(System.in);
		System.out.println("Endevina la pregunta.");
		System.out.println("Quin dels següents no és un tipus primitiu?");
		System.out.println("a) Enter");
		System.out.println("b) Scanner");
		System.out.println("c) Caràcter");
		System.out.println("d) Booleà");
		System.out.print("La teva resposta és l'opció: ");
		// Es llegeix la cadena de text.
		String paraula = lector.next();
		// És una paraula d'un únic caràcter?
		if (paraula.length() == 1) {
			// S'extreu el caràcter de la cadena de text.
			char caracter = paraula.charAt(0);
			// És un caràcter vàlid? (a, b, c o d)
		   if ((caracter >= 'a') && (caracter <= 'd')) {
				// La resposta final és correcta?
			if (caracter == RESPOSTA_CORRECTA) {
			System.out.println("Efectivament, la resposta era '" + RESPOSTA_CORRECTA + "'.");
			} else {
				System.out.println("La resposta '" + caracter + "' és incorrecta.");
			}
		    } else {
				System.out.println("'" + caracter + "' és una opció incorrecta.");
		    }
		} else {
			// No ho era.
			System.out.println("\"" + paraula + "\" no és un caràcter individual.");
		}
		lector.nextLine();
	}
}
```

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.

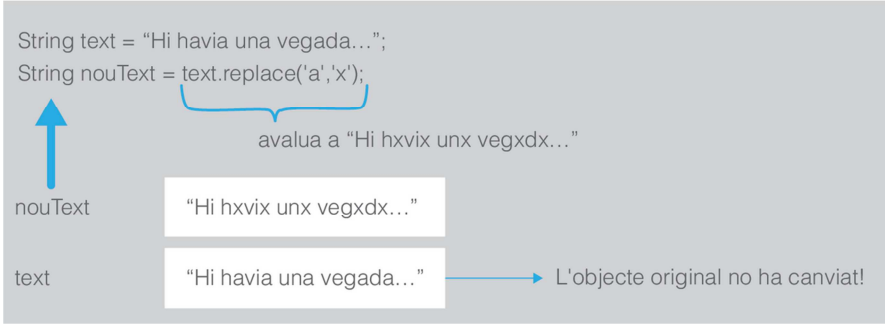
El mètode de la classe Scanner vinculat a la lectura d'una cadena de text en forma de frase en què hi ha diverses paraules separades per espais és nextLine().

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.

```
import java.util.Scanner;

//Llegeix frases escrites pel teclat.
public class LecturaFrase {
	public static final int NUM_FRASES = 4;

	public static void main(String[] args) {
		Scanner lector = new Scanner(System.in);
		System.out.println("Escriu " + NUM_FRASES + " frases.");
		System.out.println("Per acabar una frase, pitja la tecla de retorn.");
		for (int i = 0; i < NUM_FRASES; i++) {
			// Es van llegint frases una per una.
			String frase = lector.nextLine();
			System.out.println("Frase " + i + ": Has escrit \"" + frase + "\".");
		}
		// Ara no cal llegir la resta de cap línia, ja que sempre es llegeixen
		// línies senceres...
	}
}
```

Manipulació de cadenes de text

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.

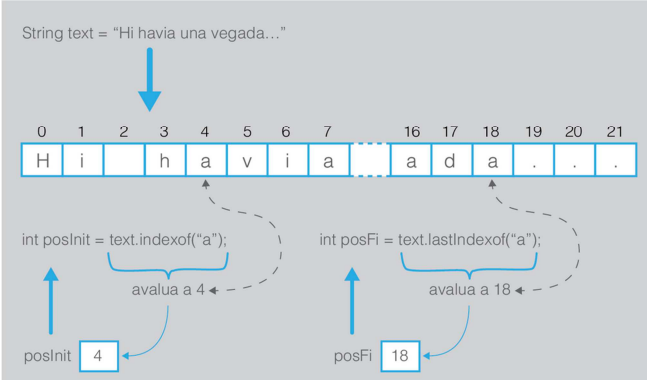


Cerca

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) avalua la primera posició dins de la cadena de text on es troba el text per cercar.
- lastIndexOf(textACercar) avalua 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.



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.

```
import java.util.Scanner;

//Cerca un caràcter concret dins d'una cadena de text qualsevol.
public class CercaCaracter {

	public static void main(String[] args) {

		Scanner lector = new Scanner(System.in);

		System.out.println("Escriu una línia de text qualsevol i pitja retorn:");
		String text = lector.nextLine();

		System.out.println("Quin caràcter vols cercar? ");
		String charText = lector.next();
		lector.nextLine();

		char charCerca = charText.charAt(0);

		int posInit = text.indexOf(charCerca);
		int posFi = text.lastIndexOf(charCerca);
		if (posInit > -1) {
			System.out.println("Les aparicions del caràcter '" + charCerca + "' són:");
			System.out.println("Primer cop - " + posInit);
			System.out.println("Darrer cop - " + posFi);
		} else {
			System.out.println("Aquest caràcter no es troba al text.");
		}

	}

}
```

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.

```
public class ComprovaArguments {
	public static void main(String args[]) {
		String [] cadenes = new String[10];
		// No oblideu mai comprovar si l'array conté arguments!
		if (args.length > 0) {
			// Es va recorrent l'array i es mira quin valor hi ha.
		 for (int i = 0; i < cadenes.length; i++) {
			if (cadenes[i].equals("-version")) {
				System.out.println("S'ha usat l'argument \"-version\"");
			} else if (cadenes[i].equals("-help")) {
					System.out.println("S'ha usat l'argument \"-help\"");
			} else if (cadenes[i].equals("-server")) {
					System.out.println("S'ha usat l'argument \"-server\"");
			} else {
				System.out.println("L'argument \"" + cadenes[i] + "\" no és vàlid!");
			}
		 }
		} else {
			System.out.println("No hi havia cap argument.");
		}
	}
}
```

- **equalsIgnoreCase(text):** Aquest altre mètode fa la mateixa funció, però ignorant majúscules i minúscules.
- 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).

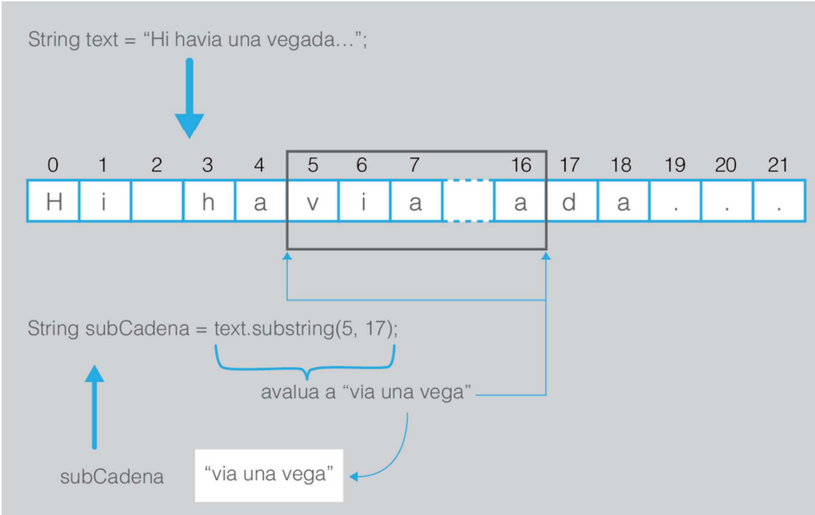
```
import java.util.Scanner;

//Joc d'endevinar una paraula, donant pistes del seu ordre alfabètic.
public class EndevinaParaula {
        // La paraula per endevinar és "objecte".
        public static final String PARAULA_SECRETA = "java";

        public static void main(String[] args) {
                Scanner lector = new Scanner(System.in);
                System.out.println("Comencem el joc.");
                System.out.println("Endevina el valor de la paraula del diccionari.");
                boolean haEncertat = false;
                while (!haEncertat) {
                        System.out.print("Quina paraula creus que és? ");
                        String paraulaUsuari = lector.next();
                        lector.nextLine();
                        int posicio = paraulaUsuari.compareTo(PARAULA_SECRETA);
                        if (posicio < 0) {
                                // La paraula de l'usuari és anterior a la secreta.
                                System.out.println("Has fallat! La paraula va després...");
                        } else if (posicio > 0) {
                                // La paraula de l'usuari és posterior a la secreta
                                System.out.println("Has fallat! La paraula va abans...");
                        } else {
                                // Si val 0, és que s'ha encertat.
                                haEncertat = true;
                        }
                }
                System.out.println("Enhorabona. Has encertat!");
        }
}
```

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) avalua 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.

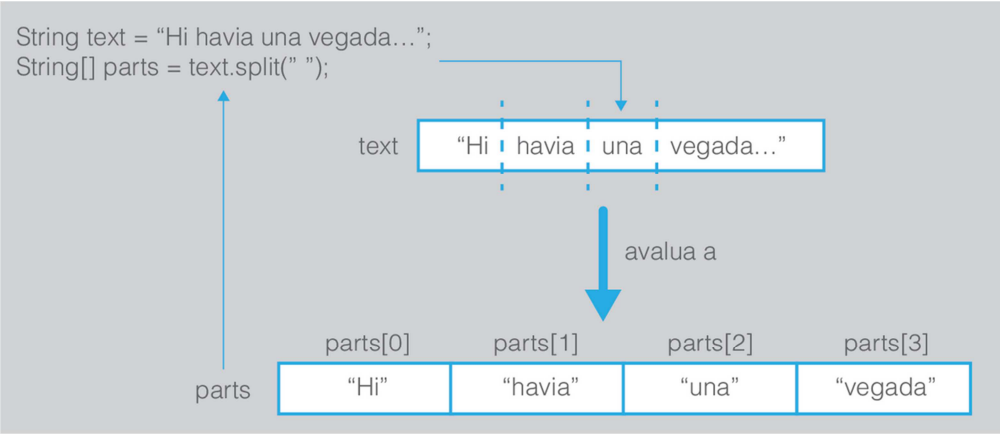


Com a exemple del funcionament de la invocació d'aquest mètode, tot seguit trobareu el codi d'un programa que extreu la primera i la darrera paraula d'una frase i les mostra per pantalla. Això és equivalent a mostrar una subcadena de text entre el primer i el darrer espai en blanc. Si hi ha menys de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

```
import java.util.Scanner;

//Un programa que extreu tot el text d'una frase, excepte la primera i la darrera paraula.
public class ExtreureParaules {
	public static void main(String[] args) {
		Scanner lector = new Scanner(System.in);
		System.out.println("Escriu una frase de text i pitja retorn:");
		String text = lector.nextLine();
		// Cerca el primer i el darrer espai en blanc.
		int iniciSubcadena = text.indexOf(' ');
		int fiSubcadena = text.lastIndexOf(' ');
		System.out.println("El text sense la primera i la darrera paraula és:");
		if (iniciSubcadena == fiSubcadena) {
			// O bé no hi ha espais (els dos mètode avaluen a -1).
			// O bé només hi ha dues paraules (els dos mètodes avaluen la
			// mateixa posició).
			// No es mostra res.
			System.out.println("(No hi ha res per escriure...)");
		} else {
			// Es mostra la cadena intermèdia.
			// La segona paraula comença una posició després del primer espai en
			// blanc.
			// La darrera paraula comença una posició després del darrer espai
			// en blanc.
			String textFinal = text.substring(iniciSubcadena + 1, fiSubcadena);
			System.out.println(textFinal);
		}
	}
}
```

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)** avalua 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.).

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, 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 avalua 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 dígits individuals.

Suposeu un programa que mostra els dígits 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.

```
public class InverteixOrdre {
        public static void main(String[] args) {
                float numero = 3.1415926535f;
                // El convertim en una cadena de text.
                String numeroText = String.valueOf(numero);
                // Recorrem els seus dígits un per un, començant pel final.
                for (int i = numeroText.length() - 1; i >= 0; i--) {
                        // Imprimim cada dígit individual (inclòs el punt decimal).
                        System.out.print(numeroText.charAt(i));
                }
        }
}
```

Conversió de cadena de text a tripus 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.

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