**Proprietati**

Clasa java.util.Properties este utilizata pentru a incarca si salva perechi cheie-valoare, care sunt stocate in fisiere text si au de obicei extensia .properties. Proprietatile sunt folosite pentru a seta valorile predefinite ale aplicatiei si sunt in mod obisnuit citite la pornirea acesteia. O alta aplicabilitate este aceea a internationalizarii unei aplicatii.

**public** **class** PropertiesExample {

**public** **static** **void** main(String[] args) {

Properties myProps = **new** Properties();

**try** {

FileInputStream fis = **new** FileInputStream("ServerInfo.properties");

myProps.load(fis);

} **catch** (IOException e) {

System.*out*.println("Error: " + e.getMessage());

}

// Print Values

System.*out*.println("Server: " + myProps.getProperty("hostName"));

System.*out*.println("User: " + myProps.getProperty("userName"));

System.*out*.println("Password: " + myProps.getProperty("password"));

}

}

Iar fisierul de proprietati va avea urmatorul continut:

# ServerInfo Properties

hostName = www.example.com

userName = user

password = pass

**Clasa System**

Pune la dispozitie urmatoarele facilitati:

* Gestioneaza fluxurile de intrare, de iesire si de eroare
* Returneaza consola asociata JVM curenta
* Incarca fisiere si librarii
* Acceseaza proprietati definite extern
* Copiaza portiuni din siruri

Toti membrii sunt statici. Constructorul este privat.

Putem folosi urmatoarele metode ale clasei pentru a accesa si modifica proprietatile sistem:

* public static String getProperty(String), returneza valoarea unei proprietati sistem, identificata prin parametru. Exemplu, afisarea class path-ul Javei:

**public** **static** **void** main(String[] args){

System.*out*.println(System.*getProperty*("java.class.path"));

}

Metoda arunca NullPointerException daca parametrul are valoarea null. Putem folosi varianta supraincarcata, cu doi parametri, pentru a defini valoarea returnata daca proprietatea sistem este invalida, ca al doilea argument

* public static Properties getProperties(), returneza toate proprietatile sistem simultan. Obiectul Properties returnat contine o lista a proprietatilor si a valorilor lor corespunzatoare. Acest obiect trebuie transformat apoi intr-o multime de proprietati sistem ale mediului curent de rulare. Daca sistemul nu gaseste aceasta multime de proprietati el initializeaza o multime predefinita. Exemplu:

**public** **static** **void** main(String[] args){

System.*getProperties*().list(System.*out*);

}

Argumentul lui list() este System.out ceea ce determina listarea continutului obiectului Properties prin fluxul predefinit de iesire. Metoda poate arunca o SecurityException daca metoda checkPropertiesAccess() din SystemManager nu permite accesul la proprietatile sistemului.

* public static String setProperty(String, String), modifica valoarea unei proprietati. Primul argument este numele proprietatii iar al doilea este noua valoarea. Metoda poate arunca o SecurityException daca metoda checkPropertiesAccess() din SecurityManager nu permite accesul la proprietatile sistemului sau o NullPointerException daca numele proprietatii nu este precizat.
* public static void setProperties(Properties), modifica proprietati aplicatiei curente prin obiectul argument. Exemplu, setarea proprietatilor sistemului:

public static void main(String[] args){  
    Properties p = new Properties(System.getProperties());  
    p.load("UserProperties.txt");  
    System.setProperties(p);  
}

In exemplul anterior am creat un obiect caruia i-am atribuit proprietatile sistemului apeland metoda load() din clasa Properties, ce are ca argument numele unui fisier text ce contine noile proprietati, UserProperties.txt. Metoda poate arunca o SecurityException daca metoda checkPropertiesAccess() din SystemManager nu permite accesul la proprietatile sistemului

* public static String getenv(String) si

public static Map<String, String> getenv() pentru a gestiona proprietatile de mediu. Proprietatile de mediu definesc procesele globale in timp ce proprietatile sistem definesc subprocesele Java. Prima versiune este folosita atunci cand se specifica numele proprietatii, ca parametru, iar a doua pentru a returna o mapare nemodificabila a intregului mediu incluzand valorile si cheile nule. Perechile returnate sunt de forma nume=valoare. Exemplu:

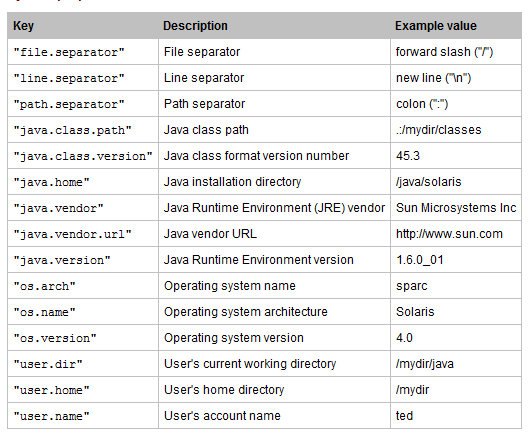
**public** **static** **void** main(String[] args){

System.*out*.println("proprietatile de mediu sunt:");

System.*out*.println(java.lang.System.*getenv*());

}

Cand pornim platforma Java clasa System intializeaza proprietatile ce contin date despre mediul de dezvoltare. Dam in continuare un tabel cu unele dintre proprietatile sistem:



Clasa java.lang.System gestioneaza fluxurile de intrare iesire ce cuprind:

* intrarile standard (System.in). Metoda System.in.read() citeste intrarile de la consola, ca intrare, sub forma de text. System.in este referinta pentru un obiect InputStream
* Iesirile standard (System.out) ce afiseaza datele de iesire, in general la monitor sau sunt expediate catre imprimanta. System.out este referinta pentru un obiect PrintStream
* Erorile standard (System.err) ce afiseaza mesajele de eroare. System.err este referinta pentru un obiect PrintStream

Toate aceste fluxuri se afla in pachetul java.io. Este de evitat reasignarea fluxurilor standard de iesire, dar putem redirectiona erorile catre un fisier de iesire folosind functia setErr(), ce are ca parametru un obiect de tip PrintStream.

Unele dispozitive nu au iesire standard si de aceea trebuie sa trimitem obligatoriu iesirea catre un fisier. Aceasta se face folosind functia setOut(), ce are ca parametru un obiect de tip PrintStream.

Daca dorim sa testam aplicatia folosind diferite multimi de intrare, memorate intr-un fisier va trebui sa cream un nou flux de intrare care sa accepte intrari din fisier in loc de cele de la tastatura. Aceasta se face folosind functia setIn() ce are ca parametru un obiect InputStream. Daca security manager este instalat acesta poate verifica daca avem autorizarea de a reasigna fluxurile standard de intrare/iesire.

Clasa System contine urmatoarele metode folositoare:

* public static void arraycopy(Object src, int src\_pos, Object dst, int dst\_pos, int length)copiaza o portiune dintr-un array intr-un altul
* public static long currentTimeMillis(), returneaza timpul scurs in milisecunde de la 1 ianuarie 1970, ora 00.00.00.
* public static void exit(int status), folosita pentru a termina programat toate firele si incheia executia interpretorului
* public static void gc(), folosita pentru a cere JVM sa faca o operatie de garbage collector
* public static long nanoTime(), folosita pentru a returna valoarea timpului exprimat in nanosecunde a cronometrului sistem. Este folosit in special pentru a calcula timpul scurs intre doua puncte ale unui program, puncte identificate prin doua apeluri ale metodei. Exemplu:

**public** **static** **void** main(String[] args){

**long** startTime = System.*nanoTime* () ;

**long** interval = System.*nanoTime* () - startTime ;

System.*out*.println (interval) ;

}

* public static Console console(), introdusa in 6.0 pentru a returna obiectul consola si a accesa dispozitivul asociat JVM-ii curente, in mod caracter. De obicei, cand un program Java este lansat, in mod automat nu avem disponibila o consola si de aceea metoda va returna NULL. Obiectul Console detine majoritatea functionalitatilor fluxului standard, dar si functionalitati noi cum ar fi introducerea securizata a parolei. Exemplu:

**public** **static** **void** main(String[] args){

Console thisConsole = System.*console*();

**if** (thisConsole != **null**) {

thisConsole.writer().print("introduceti parola");

thisConsole.flush();

**char**[] password = thisConsole.readPassword();

}

}

In exemplul anterior metoda readPassword() citeste parola fara ecou si o returneaza ca pe un sir de caractere. Exemplul va returna o consola nenula doar in linia de comanda.

**Clasa PrintWriter**

Scrie caractere in loc de octeti. Clasa implementeaza toate metodele de printare existente in PrintStream. Clasa PrintStream converteste caracterele in octeti folosind codarea de caractere predefinita a platformei. Spre deosebire de PrintStream, daca flush-ul automat este activ, el va fi facut doar cand una dintre metodele println(), printf() sau format() este invocata, in loc de a fi facut ori de cate ori caracterul linie noua este inclus la iesire.

**public** **class** PrintWriterExample {

**public** **static** **void** main(String[] args) {

PrintWriter pw = **new** PrintWriter(System.*out*, **true**);

pw.println("This is some output.");

}

}

In exemplul anterior am creat un obiect ce foloseste optiunea de auto flush. Valoarea true va forta PrintWriter-ul sa faca flush dupa fiecare linie afisata la consola.

Java furnizeaza mai multe metode de a formata string-uri: metodele printf() si String.format().

**public** **class** PrintfExample {

**public** **static** **void** main(String[] args){

PrintWriter pw = **new** PrintWriter(System.*out*, **true**);

**double** price = 24.99; **int** quantity = 2; String color = "Blue";

System.*out*.printf("We have %03d %s Polo shirts that cost $%3.2f.\n", quantity, color, price);

System.*out*.format("We have %03d %s Polo shirts that cost $%3.2f.\n", quantity, color, price);

String out = String.*format*("We have %03d %s Polo shirts that cost $%3.2f.", quantity, color, price);

System.*out*.println(out);

pw.printf("We have %03d %s Polo shirts that cost $%3.2f.\n", quantity, color, price);

}

}

**Clasa String**

Java utilizeaza caractere Unicode pe 16 biti pentru a codifica datele unui string. Aceasta tehnica permite Javei sa codifice pina la 65536 de caractere. In Java stringurile sunt incapsulate prin clasele String si StringBuffer. Clasa String extinde Object si implementeaza interfetele Serializable, Comparable si CharSequence. Obiectele clasei String nu sunt obligatoriu a fi create prin operatorul new, ci doar printr-o simpla atribuire a unei valori literale. Exista mai multi constructori asociati clasei String ce ne permit sa cream instante pentru:

* un sir de bytes, adica un string codificat UTF-8. Intr-un caracter unicode doar un byte reprezinta caracterul, celalalt este un cod de limba. (public String(byte[])). Exemplu:

**byte**[] encodedBytes = **new** **byte**[] {111, 99, 122, 123, 96, 117} ;

String s = **new** String(encodedBytes) ;

System.out.println(s);

* o submultime a unui sir de bytes (public String(byte[],int indexInceput,int lung)). Exemplu:

**byte**[] encodedBytes = **new** **byte**[] {111, 99, 122, 123, 96, 117} ;

String s = **new** String(encodedBytes, 2, 3) ;

System.out.println(s);

* un sir de caractere (public String(char[])). Exemplu:

**char**[] chars = **new** **char**[] {'a', 'r', 'r', 'a', 'y', 's'} ;

String s = **new** String(chars) ;

System.*out*.println(s);

* o submultime a unui sir de caractere (public String(char[],int indexInceput,int lung))
* un alt string (public String(String))
* un string buffer (public String(StringBuffer))

Unii constructori ne permit sa folosim un string, ca al doilea parametru, pentru a specifica o codificare specifica a caracterelor. Acest constructor este folosit in special atunci cand mapam o multime de bytes unui multimi speciale de caractere. Avem mai multe codificari. Printre acestea: UTF-8 pentru codificarea Unicode, EUC (Extended Unix Code) codifica caracterele in diverse limbi asiatice, US-ASCII, ISO-8859-1, UTF-16, etc. Sintaxa acestui constructor este: String (byte[], String charset); Exemplu:

**byte**[] encodedBytes = **new** **byte**[] {15, 22, 22, 53, 96, 17} ;

String s;

**try** {

s = **new** String(encodedBytes, "UTF-8");

System.*out*.println(s);

} **catch** (UnsupportedEncodingException e) {

e.printStackTrace();

}

Odata creat obiectul String acesta nu mai poate fi modificat. Cand asignam o noua valoare obiectului, practic, cream un nou string.

Orice aplicatie are un pool de stringuri. In momentul compilarii compilatorul compara toate stringurile intalnite cu cele deja existente in pool si le adauga pe cele inexistente. Daca stringul exista, compilatorul economiseste memoria si utilizeaza o copie din pool.

Clasa StringBuffer permite modificarea stringului valoare.

Clasa String contine mai multe metode pentru compararea stringurilor: equals() (compara doua stringuri caracter cu caracter) , equalsIgnoreCase() si contentEquals() (pentru compararea obiectului curent String cu un obiect StringBuffer). Putem folosi in loc de equals si operatorul ==, care poate da rezultate bune din cauza poolului de stringuri, dar acest lucru nu este garantat.

Alte metode ale clasei String:

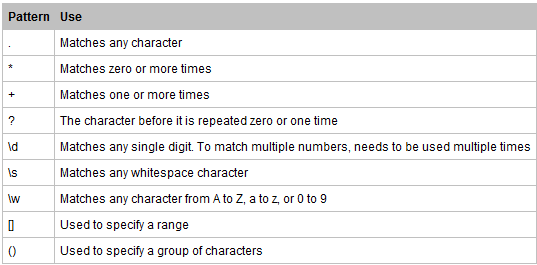
* int length(): returneaza numarul de caractere ale stringului curent
* boolean startsWith(String): returneaza true daca stringul curent incepe cu stringul parametru
* boolean endsWith(String): returneaza true daca stringul curent se termina cu stringul parametru
* String trim(): returneaza un string fara spatiile de inceput si sfarsit ale stringului curent si fara caracterele de control ASCII
* String substring(int inceput, int sfarsit): returneaza un substring al stringului cuprins intre indexul inceput si pina la indexul sfarsit
* String toUpperCase(): returneaza un string cu toate literele mari ale stringului curent
* String toLowerCase(): returneaza un string cu toate literele mici ale stringului curent
* boolean isEmpty(): returneaza true daca stringul curent este gol sau are lungime 0

**Utilizarea expresiilor regulate cu stringuri**

Pentru a verifica ca o secventa de caractere se potriveste unui anume sablon folosim expresii regulate. In Java crearea si utilizarea expresiilor regulate sunt gestionate de clasele Pattern si Matcher din pachetul java.util.regex. Pentru a utiliza o expresie regulata trebuie:

* sa o definim
* sa o aplicam unei secvente de caractere

Tabelul de mai jos prezinta o lista a simbolurilor folosite in crearea expresiilor regulate.



In plus, [^abc] este utilizata pentru a specifica orice caracter care nu este a, b sau c, iar | specifica alternarea.

\D este folosita ca o clasa negare si reprezinta [^0-9].

\W este folosita ca o clasa negare si reprezinta [^a-zA-Z0-9\_].

\S este folosita ca o clasa negare si reprezinta [^ \r\t\n\f\0XB].

Caracterele albe sunt: \t – tab, \n – linie noua, \r – retur de car, \f – form feed, \0XB – tab vertical.

Clasa Pattern este folosita pentru a defini o expresie regulata prin metoda Pattern.compile() ce returneaza un obiect Pattern. Exemplu:

**public** **static** **void** main(String args[]) {

Pattern sablon = Pattern.*compile*("[0-9][0-9][a-z]");

Matcher potrivire = sablon.matcher("lmn89t");

**boolean** potrivireGasita = potrivire.matches();

**if** (potrivireGasita)

System.*out*.println("se potriveste!") ;

**else**

System.*out*.println("nu se potriveste!") ;

}

In exemplul anterior obiectul sablon defineste o expresie regulata formata din doua cifre urmate de o litera. Apoi cream un obiect Matcher pentru stringul in care dorim sa facem cautarea. Cautarea se face prin apelul metodei matcher() a clasei Pattern. Pentru a gasi in stringul de cautat o potrivire cu expresia regulata apelam metoda booleana matches() a clasei Matcher.

Alt exemplu:

**public** **class** CustomCharClassExample {

**public** **static** **void** main(String[] args) {

String t = "It was the best of times";

Pattern p1 = Pattern.*compile*("w.s");

Matcher m1 = p1.matcher(t);

**if** (m1.find()) {

System.*out*.println("Found: " + m1.group());

}

Pattern p2 = Pattern.*compile*("w[abc]s");

Matcher m2 = p2.matcher(t);

**if** (m2.find()) {

System.*out*.println("Found: " + m2.group());

}

Pattern p3 = Pattern.*compile*("t[^eou]mes");

Matcher m3 = p3.matcher(t);

**if** (m3.find()) {

System.*out*.println("Found: " + m3.group());

}

}

}

Am creat un obiect Pattern pentru a stoca expresia regulata dupa care dorim sa facem cautarea, am creat obiectul Matcher prin trimiterea textului in care se face cautarea, metodei din obiectul Pattern anterior creat (aceasta metoda returneaza un Matcher). Matcher.group() returneaza caracterele ce se potrivesc sablonului.

Unele dintre clasele caracter se folosesc repetat.

**public** **class** PredefinedCharClassExample {

**public** **static** **void** main(String[] args){

String t = "Jo told me 20 ways to San Jose in 15 minutes.";

System.*out*.println(t);

Pattern p1 = Pattern.*compile*("\\d\\d");

Matcher m1 = p1.matcher(t);

**while** (m1.find()){

System.*out*.println("Found: " + m1.group());

}

Pattern p2 = Pattern.*compile*("\\sin\\s");

Matcher m2 = p2.matcher(t);

**if** (m2.find()) System.*out*.println("Found: " + m2.group());

Pattern p3 = Pattern.*compile*("\\Sin\\S");

Matcher m3 = p3.matcher(t);

**if** (m3.find()) System.*out*.println("Found: " + m3.group());

}

}

In exemplu: \\d\\d gaseste oricare doua cifre, \\sin\\s gaseste in inclus intre doua spatii, \\Sin\\S gaseste in inclus intre doua caractere non-spatiu.

Cuantificatorii permit selectarea usoara a unui domeniu de caractere:

* {n}, caracterul anterior apare de exact n ori
* {m,n}, caracterul anterior apare de m pina la n ori
* {m,}, caracterul anterior apare de cel putin m ori
* (xx){n}, grupul de caractere apare repetat de n ori

**public** **class** QuantifierExample {

**public** **static** **void** main(String[] args){

String t = "Longlonglong ago, in a galaxy far far away.";

Pattern p1 = Pattern.*compile*("ago.\*");

Matcher m1 = p1.matcher(t);

**if** (m1.find()) System.*out*.println("Found: " + m1.group());

Pattern p2 = Pattern.*compile*("gal.{3}");

Matcher m2 = p2.matcher(t);

**if** (m2.find()) System.*out*.println("Found: " + m2.group());

Pattern p3 = Pattern.*compile*("(long){2}");

Matcher m3 = p3.matcher(t);

**if** (m3.find()) System.*out*.println("Found: " + m3.group());

}

}

In exemplu:

* ago.\*, gaseste ago si 0 sau mai multe caractere ramase pe linie
* gal.{3}, gaseste gal si urmatoarele trei caractere
* (long){2}, gaseste long repetat de doua ori

O expresie regulata incearca de fiecare data sa aduca cat mai multe caractere cu putinta. Acesta este numit principiul „greediness”. Utilizam ? pentru a limita cautarea la cea mai scurta potrivire posibila.

**public** **class** GreedinessExample {

**public** **static** **void** main(String[] args){

String t = "Longlonglong ago, in a galaxy far far away.";

Pattern p1 = Pattern.*compile*("ago.\*far");

Matcher m1 = p1.matcher(t);

**if** (m1.find()) System.*out*.println("Found: " + m1.group());

// Produces: ago, in a galaxy far far

Pattern p2 = Pattern.*compile*("ago.\*?far");

Matcher m2 = p2.matcher(t);

**if** (m2.find()) System.*out*.println("Found: " + m2.group());

// Produces: ago, in a galaxy far

}

}

In exemplu:

* ago.\*far, ilustreaza principiul greediness
* ago.\*?far, ilustreaza inhibarea principiului greediness

Caracterele boundary pot fi utilizate pentru a potrivi diferite parti ale unei linii:

* ^, potriveste inceputul liniei
* $, potriveste sfarsitul liniei
* \b, potriveste inceputul sau sfarsitul unui cuvant
* \B, nu potriveste inceputul sau sfarsitul unui cuvant

**public** **class** BoundaryCharExample {

**public** **static** **void** main(String[] args){

String t = "it was the best of times or it was the worst of times";

Pattern p1 = Pattern.*compile*("^it.\*?times");

Matcher m1 = p1.matcher(t);

**if** (m1.find()) System.*out*.println("Found: " + m1.group());

Pattern p2 = Pattern.*compile*("\\sit.\*times$");

Matcher m2 = p2.matcher(t);

**if** (m2.find()) System.*out*.println("Found: " + m2.group());

Pattern p3 = Pattern.*compile*("\\bor\\b.{3}");

Matcher m3 = p3.matcher(t);

**if** (m3.find()) System.*out*.println("Found: " + m3.group());

}

}

Cu expresii regulate putem folosi parantezele pentru a identifica parti ale unui string pentru potriviri.

**public** **class** MatchingExample {

**public** **static** **void** main(String[] args){

String email = "george.washington@example.com";

Pattern p1 = Pattern.*compile*("(\\S+?)\\.(\\S+?)\\@(\\S+)");

Matcher m1 = p1.matcher(email);

**if** (m1.find()){

System.*out*.println("First: " + m1.group(1));

System.*out*.println("Last: " + m1.group(2));

System.*out*.println("Domain: " + m1.group(3));

System.*out*.println("Everything Matched: " + m1.group(0));

}

}

}

In exemplu potrivim partile componente ale unei adrese de email. Fiecare grup de paranteze este numerotat. Intr-o expresie regulata group(0) sau group() potriveste tot textul.

In exemplu:

* ^it.\*?times, reprezinta secventa ce incepe o linie cu it urmata de niste caractere si times, greediness fiind dezactivat
* \\sit.\*times$, reprezinta secventa ce incepe cu it urmata de niste caractere si times
* \\bor\\b.{3}, cauta or incercuit decuvinte si urmatoarele trei caractere

Clasa String are cateva expresii ce suporta expresii regulate:

* boolean matches(String): descrisa anterior
* String replaceFirst(String exReg, String inlocuitor): returneaza stringul obtinut prin inlocuirea in stringul curent a primei aparitii a stringului exReg ce se potriveste cu inlocuitor
* String replaceAll(String exReg, String inlocuitor): returneaza stringul obtinut prin inlocuirea in stringul curent a tuturor aparitiilor ale stringului exReg ce se potrivesc cu inlocuitor

**public** **class** ReplacingExample {

**public** **static** **void** main(String[] args){

String header = "<h1>This is an H1</h1>";

Pattern p1 = Pattern.*compile*("h1");

Matcher m1 = p1.matcher(header);

**if** (m1.find()){

header = m1.replaceAll("p");

System.*out*.println(header);

}

}

}

* String[] split(String): returneaza un sir de stringuri, obtinut din obiectul curent prin folosirea parametrului pe post de separator

**public** **class** StringSplit {

**public** **static** **void** main(String[] args){

String shirts = "Blue Shirt, Red Shirt, Black Shirt, Maroon Shirt";

String[] results = shirts.split(", ");

**for**(String shirtStr:results){

System.*out*.println(shirtStr);

}

}

}

Clasa StringTokenizer face acelasi lucru ca split() dar intr-un mod diferit. Trebuie sa iteram tokenii pentru a avea acces la ei.

Clasa Scanner poate imparti un string sau un stream in token-i. In plus un Scanner poate fi utilizat pentru a tokeniza numere sau a le converti la tipuri primitive.

**Clasele StringBuffer si StringBuilder**

Aceste clase sunt preferate a fi folosite atunci cand dorim sa concatenam string-uri. Ele sunt mult mai eficiente decat folosirea operatorului +. Din punct de vedere al concurentei StringBuilder nu este thread safe in timp ce StringBuffer este. Ca indicatie de performanta este recomandarea ca sa setam capacitatea initiala la nevoile pe care le avem. Orice redimensionare conduce la scaderea performantei.

**public** **class** StringBuilding {

**public** **static** **void** main(String[] args){

StringBuilder sb = **new** StringBuilder(500);

sb.append(", the lightning flashed and the thunder rumbled.\n");

sb.insert(0, "It was a dark and stormy night");

sb.append("The lightning struck...\n").append("[ ");

**for**(**int** i = 1; i < 11; i++){

sb.append(i).append(" ");

}

sb.append("] times");

System.*out*.println(sb.toString());

}

}

Folosim StringBuffer-ul cand dorim sa modificam un string, dinamic, prin cod. Clasa are urmatorii patru constructori:

* StringBuffer() construieste un string gol
* StringBuffer(int) construieste un string gol de capacitate data de parametru. Daca bufferul depaseste capacitatea aceasta este automata marita cu valoarea parametrului
* StringBuffer(String) construieste un string initializat cu parametrul
* StringBuffer(CharSequence) construieste un string initializat cu parametrul

Metode uzuale din aceasta clasa:

* void setLength(int): modifica lungimea bufferului. Caracterele ramin aceleasi, cele care sunt eventual in plus vor fi caractere null
* void setCharAt(int,char): modifica caracterul de la pozitia indicata de primul parametru
* StringBuffer reverse(): inverseaza ordinea caracterelor din buffer
* StringBuffer append(String): adauga stringul parametrului, bufferului
* StringBuffer insert(int, String): insereaza in buffer stringul parametru incepand de pe pozitia indicata de primul parametru al functiei

Exemplu de folosire a functiilor clasei StringBuffer:

**public** **class** StringBufferDemo {

**static** **void** printBuffer( StringBuffer buf ) {

System.*out*.println( "buf = " + buf );

System.*out*.println( "length = " + buf.length() );

System.*out*.println( "capacity = " + buf.capacity() );

}

**public** **static** **void** main( String [] args ) {

StringBuffer buf = **new** StringBuffer( "This is a StringBuffer." );

*printBuffer*( buf ); // prints "This is a StringBuffer.",

// length = 23, capacity = 39

buf.setLength ( 7 );

*printBuffer*( buf ); // prints "This is",

// length = 7, capacity = 39

StringBuffer buf2 = **new** StringBuffer( 10 );

*printBuffer*( buf2 ); // prints "",

// length = 0, capacity = 10

buf2.append( "Write once, run anywhere.") ;

*printBuffer*( buf2 ); // prints "Write once, run anywhere."

// length = 25, capacity = 25

buf2.setCharAt( 10, '.' );

buf2.setCharAt( 12, 'R' );

*printBuffer*( buf2 ); // prints "Write once. Run anywhere."

// length = 25, capacity = 25

StringBuffer buf3 = **new** StringBuffer();

buf3 = buf2.insert( 12, "Distribute. " );

*printBuffer*( buf3 ); // prints "Write once. Distribute. Run anywhere."

// length = 37, capacity = 41

buf3 = buf3.append( " Hooray!" );

*printBuffer*( buf3 ); // prints "Write once. Distribute. Run anywhere. Hooray!"

// length = 45, capacity = 57

CharSequence charSeq = "Java" ;

StringBuffer buf4 = **new** StringBuffer( charSeq );

*printBuffer*( buf4 ); // prints "Java"

// length = 4, capacity = 20

buf4.reverse();

*printBuffer*( buf4 ); // prints "avaJ"

// length = 4, capacity = 20

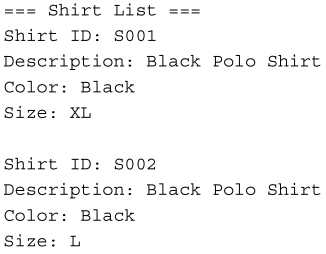
}

}

StringBuffer a fost creata pentru a fi sigura in firele de executie. StringBuilder este desemnata sa inlocuiasca StringBuffer, acolo unde bufferul este utilizat de un singur fir.

TEMA:

1. se da o lista de tricouri, caracteristicile unui tricou fiind separate prin virgula. Sa se afiseze lista de tricouri astfel:



1. fie urmatorul fisier text:

<html>

<title>Gettysburg Address</title>

<style type=*"text/css"*>

**body** {

font-family: *sans-serif*;

}

**h2,h4** {

text-align: *center*;

}

</style>

<body>

<h2>Gettysburg Address</h2>

<h4>Abraham Lincoln</h4>

<h4>Thursday, November 19, 1863</h4>

<div class=*"paragraph"*>

<p class=*"line"*>Four score and seven years ago our fathers brought

forth on this continent a new nation, conceived in liberty, and

dedicated to the proposition that all men are created equal.</p>

<p class=*"line"*>Now we are engaged in a great civil war, testing

whether that nation, or any nation, so conceived and so dedicated,

can long endure.</p>

<p class=*"line"*>We are met on a great battle-field of that war.</p>

<p class=*"line"*>We have come to dedicate a portion of that field,

as a final resting place for those who here gave their lives that

that nation might live.</p>

<p class=*"line"*>It is altogether fitting and proper that we should

do this.</p>

</div>

<div class=*"paragraph"*>

<p class=*"line"*>But, in a larger sense, we can not dedicate, we

can not consecrate, we can not hallow this ground.</p>

<p class=*"line"*>The brave men, living and dead, who struggled

here, have consecrated it, far above our poor power to add or

detract.</p>

<p class=*"line"*>The world will little note, nor long remember what

we say here, but it can never forget what they did here.</p>

<p class=*"line"*>It is for us the living, rather, to be dedicated

here to the unfinished work which they who fought here have thus far

so nobly advanced.</p>

<p class=*"line"*>It is rather for us to be here dedicated to the

great task remaining before uses that from these honored dead we

take increased devotion to that cause for which they gave the last

full measure of devotion that we here highly resolve that these

dead shall not have died in vain that this nation, under God,

shall have a new birth of freedom and that government of the

people, by the people, for the people, shall not perish from the

earth.</p>

</div>

</body>

</html>

Scrieti o aplicatie ce cauta in fisierul anterior un text folosind expresii regulate. Daca textul este gasit sa se afiseze numarul liniei si linia intreaga in care se afla textul cautat. (Spre exemplu: toate liniile ce contin <h4>, toate liniile ce contin cuvantul to, toate liniile ce incep cu 4 spatii, toate liniile ce contin tag-uri de inchidere HTML)

1. in exemplul anterior inlocuim tag-urile <p> prin <span>. Mai mult, valoarea atributului class va fi sentence in loc de line. Folositi expresii regulate pentru a gasi liniile in care dorim sa facem modificarile. Apoi, folositi expresii regulate pentru a transforma tag-urile si atributele vizate. Noul fisier va fi afisat la consola.