4. Exemple de clase de biblioteca Java

4.1. Clase Java pentru lucrul cu (siruri de) caractere

4.1.1. Incapsularea caracterelor si a sirurilor de caractere

Platforma Java contine trei **clase care pot fi folosite pentru lucrul cu date de tip (sir de) caractere**:
- **Character** - clasa care incapsuleaza o singura **valoare caracter**
- **String** - clasa care incapsuleaza un **sir de caractere ne**modificabil (*immutable*)
- **StringBuffer** - clasa care incapsuleaza un **sir de caractere modificabil** (*mutable*)

4.1.2. Clasa care incapsuleaza caractere Unicode (Character) – interfata publica

Clasa **`Character`** incapsuleaza o singura **valoare caracter**, **nemodificabila** (*immutable*). Caracterul incapsulat este astfel protejat, poate fi pasat in interiorul unui obiect (accesat prin referinta), comparat, convertit. De asemenea, caracterului ii poate fi determinat tipul (litera, numar, etc.).

4.1.2.1. Declaratia clasei Character

Declaratiile de pachet si clasa ale clasei **`Character`** sunt urmatoarele:

```
package java.lang;

public final class Character extends Object implements java.io.Serializable,
                                                       Comparable {
  // corpul clasei Character
}
```

4.1.2.2. Constructorii clasei Character

Constructorul clasei `Character`:

`Character``(char value)` Construieste un obiect `Character` nou care incapsuleaza valoarea `char` specificata ca argument.

4.1.2.3. Metodele clasei Character

Folosind urmatorul format de prezentare:

`[modif.] tipReturnat`	**`numeMetoda`**`([tipParametru numeParametru [, tipParametru numeParametru]])` Descrierea metodei

in continuare sunt detaliate cateva dintre **metodele** declarate in clasa **`Character`**:

`char`	**`charValue`**`()` Returneaza valoarea primitiva incapsulata in acest obiect `Character`.
`int`	**`compareTo`**`(Character anotherCharacter)` Compara numeric doua obiecte `Character`. Rezultatul este diferenta intre codificarea caracterului incapsulat obiectul curent si codificarea incapsulat in caracterul primit ca parametru.
`int`	**`compareTo`**`(Object o)` Compara acest `Character` (`this`) cu un obiect al clasei `Object`. Daca obiectul argument este un `Character` functia se comporta ca `compareTo(Character)`. Altfel, metoda arunca o exceptie de tipul `ClassCastException` (obiectele `Character` fiind comparabile doar intre ele).
`boolean`	**`equals`**`(Object obj)` Compara continutul obiectului curent cu continutul obiectului primit ca parametru.
`static int`	**`getNumericValue`**`(char ch)` Returneaza valoare `int` care reprezinta codificarea Unicode a caracterului primit ca parametru.
`static int`	**`getType`**`(char ch)` Returneaza o value indicand categoria generala a caracterului specificat ca parametru.
`static boolean`	**`isDefined`**`(char ch)` Determina daca un caracter es if a character is defined in Unicode.
`static boolean`	**`isDigit`**`(char ch)` Determina daca un caracter specificat este digit (caz in care `Character.getType(ch)` este `DECIMAL_DIGIT_NUMBER`.).
`static boolean`	**`isLetter`**`(char ch)` Determina daca un caracter specificat este litera.
`static boolean`	**`isLetterOrDigit`**`(char ch)` Determina daca un caracter specificat este litera sau digit.
`static boolean`	**`isLowerCase`**`(char ch)` Determina daca un caracter specificat este caracter litera mica.
`static boolean`	**`isSpaceChar`**`(char ch)` Determina daca un caracter specificat este caracter Unicode spatiu.
`static boolean`	**`isUpperCase`**`(char ch)` Determina daca un caracter specificat este caracter litera mare.
`static boolean`	**`isWhitespace`**`(char ch)` Determina daca un caracter specificat este *white space* conform Java (vezi documentatia Java).
`static char`	**`toLowerCase`**`(char ch)` Converteste caracterul primit ca argument la litera mica.
`String`	**`toString`**`()` Returneaza un obiect `String` care incapsuleaza caracterul curent.
`static String`	**`toString`**`(char c)` Returneaza un obiect `String` care incapsuleaza caracterul specificat ca argument (`char`).
`static char`	**`toUpperCase`**`(char ch)` Converteste caracterul primit ca argument la litera mare.

Asadar, **clasa `Character` permite**:
- **comparatii** intre caractere (**`compareTo()`**, **`equals()`**, etc.),
- **determinarea tipului** de caracter (**`isLetter()`**, **`isDigit()`**, **`isLowerCase()`**, **`isUpperCase()`**, etc.),
- **conversii** (**`toLowerCase()`**,**`toUpperCase()`**, **`toString()`**, etc.).

Exemplu de utilizare a clasei **`Character`**:

```
public class CharacterDemo {
   public static void main(String args[]) {
      Character a = new Character('a');
      Character a2 = new Character('a');
      Character b = new Character('b');

      int difference = a.compareTo(b);

      if (difference == 0) {
         System.out.println("a este egal cu b.");
      } else if (difference < 0) {
         System.out.println("a este mai mic decat b.");
      } else if (difference > 0) {
         System.out.println("a este mai mare decat b.");
      }
      System.out.println("a is " + ((a.equals(a2)) ? "equal" : "not equal")
         + " to a2.");
      System.out.println("Caracterul" + a.toString() + " este "
         + (Character.isUpperCase(a.charValue()) ? "upper" : "lower") + "case.");
   }
}
```

Rezultatul executiei programului este urmatorul:

```
a este mai mic decat b.
a este egal cu a2.
Caracterul a este lowercase.
```

4.1.3. Clasa care incapsuleaza siruri de caractere nemodificabile (String) – interfata publica

Clasa **`String`** incapsuleaza **un sir de caractere**, **nemodificabil** (*immutable*). Sirul de caractere incapsulat este astfel protejat, poate fi pasat in interiorul unui obiect (accesat prin referinta), comparat, convertit la valori numerice sau la tablouri de octeti sau caractere. De asemenea, caracterele pe care le contine pot fi accesate individual.

4.1.3.1. Declaratia clasei String

Declaratiile de pachet si clasa ale clasei **`String`** sunt urmatoarele:

```
package java.lang;

public final class String implements java.io.Serializable, Comparable,
CharSequence {
  // corpul clasei String
}
```

4.1.3.2. Constructorii clasei String

Principalii constructori ai clasei `String`:

`String``()` Initializeaza un obiect `String` nou creat astfel incat sa reprezinte un sir de caractere vid.
`String``(byte[] bytes)` Construieste un nou obiect `String` prin decodarea unui tablou de octeti specificat utilizand setul de caractere implicit al platformei pe care se lucreaza. `bytes` reprezinta tabloul de octeti ce va fi decodat in caractere.
`String``(byte[] bytes, int offset, int length)` Construieste un nou obiect `String` prin decodarea unui sub-tablou de octeti specificat (`bytes` reprezinta tabloul de octeti ce va fi decodat in caractere, `offset` indexul primului octet din tablou ce va fi decodat iar `length` numarul de octeti ce va fi decodat) utilizand setul de caractere implicit al platformei pe care se lucreaza..
`String``(char[] value)` Construieste un nou obiect `String` astfel incat sa reprezinte sirul de caractere continut in tabloul de caractere primit ca argument (`value`). Continutul tabloului de caractere este copiat, astfel incat modificarile ulterioare ale tabloului de caractere nu afecteaza nou creatul sir de caractere.
`String``(char[] value, int offset, int count)` Construieste un nou obiect `String` astfel incat sa reprezinte subsirul de caractere continut in tabloul de caractere primit ca argument (`value`).
`String``(String original)` Initializeaza un obiect `String` nou creat astfel incat sa reprezinte acelasi sir de caractere ca argumentul (sirul nou creat e o copie a celui primit ca argument).
`String``(StringBuffer buffer)` Construieste un nou obiect `String` care contine sirul de caractere curent continut in argumentul de tip sir de caractere modificabil (*string buffer*).

4.1.3.3. Metodele clasei String

Declaratiile si descrierea catorva metode ale clasei **`String`**:

`char`	**`charAt`**`(int index)` Returneaza caracterul aflat la indexul specificat.
`int`	**`compareTo`**`(Object o)` Compara acest `String` (`this`) cu un obiect al clasei `Object`. Daca obiectul argument este un `String` functia se comporta ca `compareTo(String)`. Altfel, ea arunca (declanseaza) o exceptie `ClassCastException` (obiectele `String` sunt comparabile doar cu alte obiecte `String`).
`int`	**`compareTo`**`(String anotherString)` Compara doua siruri de caractere din punct de vedere lexicografic. Comparatia este bazata pe valoarea Unicode a fiecarui caracter al sirului. Rezultatul este un intreg negativ daca obiectul curent (`this`) precede argumentul si pozitiv daca urmeaza dupa el. Rezultatul e nul daca sirurile sunt egale. Daca sirurile au caractere diferite la unul sau mai multe pozitii ale indexului, fie *indexMin* cel mai mic astfel de index, atunci sirul al carui caracter de index *indexMin* are valoarea cea mai mica

	(determinate utilizand operatorul <) precede lexicographic pe celalalt sir (iar valoarea returnata este in acest caz egala cu `this.charAt(indexMin)-anotherString.charAt(indexMin)`). Daca sirurile sunt de lungime diferita, iar sirul mai scurt nu difera pe toata lungimea lui de caracterele sirului mai lung, atunci el precede sirul mai lung (iar valoarea returnata este in acest caz egala cu `this.length()-anotherString.length()`).
`int`	**`compareToIgnoreCase`**`(String str)` Compara doua siruri de caractere din punct de vedere lexicografic, ignorand diferentele intre literele mari si mici.
`String`	**`concat`**`(String str)` Concateneaza sirul de caractere primit ca argument (`str`) la sfarsitul sirului curent (`this`).
`boolean`	**`contentEquals`**`(StringBuffer sb)` Returneaza `true` daca si numai daca sirul curent (`this`) reprezinta acelasi sir de caractere ca obiectul `StringBuffer` specificat ca argument (`sb`).
`static String`	**`copyValueOf`**`(char[] data)` Returneaza un obiect `String` care reprezinta sirul de caractere din tabloul de caractere the array specificat ca argument (`data`).
`static String`	**`copyValueOf`**`(char[] data, int offset, int count)` Returneaza un obiect `String` care reprezinta subsirul de caractere din tabloul de caractere the array specificat ca argument (`data`).
`boolean`	**`endsWith`**`(String suffix)` Testeaza daca acest `String` se incheie cu sufixul specificat ca argument (`suffix`).
`boolean`	**`equals`**`(Object anObject)` Compara acest sir de caractere cu obiectul specificat ca argument.
`boolean`	**`equalsIgnoreCase`**`(String anotherString)` Compara acest `String` cu alt `String`, ignorand diferentele intre literele mari si cele mici.
`byte[]`	**`getBytes`**`()` Codeaza sirul de caractere curent intr-un sir de octeti, utilizand setul de caractere implicit al platformei pe care se lucreaza, stocand rezultatul intr-un nou tablou de octeti.
`void`	**`getChars`**`(int srcBegin, int srcEnd, char[] dst, int dstBegin)` Copiaza caracterele din sirul curent in tabloul de caractere destinatie. Primul caracter copiat are index `srcBegin`; ultimul caracter copiat are `srcEnd-1` (astfel incat numarul total de caractere copiate este `srcEnd-srcBegin`). Caracterele sunt copiate intr-un subtablou `dst` incepand cu indexul `dstBegin` si terminand la indexul: `dstbegin+(srcEnd-srcBegin)-1`
`int`	**`hashCode`**`()` Returneaza un *hash code* pentru acest sir de caractere, folosind relatia: s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1] utilizand aritmetica `int`, unde `s[i]` este al *i*-lea caracter al sirului, `n` este lungimea sirului, iar `^` indica exponentierea. (Valoarea *hash* a unui sir vid este zero.)
`int`	**`indexOf`**`(int ch)` Returneaza indexul minim in sirul de caractere curent la care apare caracterul specificat ca argument, si valoarea `-1` daca nu exista acel caracter in sirul de caractere curent.

int	**indexOf**(int ch, int fromIndex) Returneaza indexul minim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare caracterul specificat ca argument (ch), si valoarea -1 daca nu exista acel caracter in sirul de caractere curent.
int	**indexOf**(String str) Returneaza indexul minim in sirul de caractere curent la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**indexOf**(String str, int fromIndex) Returneaza indexul minim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**lastIndexOf**(int ch) Returneaza indexul maxim in sirul de caractere curent la care apare caracterul specificat ca argument, si valoarea -1 daca nu exista acel caracter in sirul de caractere curent.
int	**lastIndexOf**(int ch, int fromIndex) Returneaza indexul maxim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare caracterul specificat ca argument (ch), si valoarea -1 daca nu exista acel caracter in sirul de caractere curent.
int	**lastIndexOf**(String str) Returneaza indexul maxim in sirul de caractere curent la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**lastIndexOf**(String str, int fromIndex) Returneaza indexul maxim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**length**() Returneaza lungimea sirului de caractere curent (numarul de caractere pe care le contine).
String	**replace**(char oldChar, char newChar) Returneaza un nou sir obtinut prin inlocuirea tuturor aparitiilor caracterului oldChar in sirul curent cu caracterul newChar.
boolean	**startsWith**(String prefix) Testeaza daca sirul de caractere curent incepe cu prefixul specificat ca argument.
boolean	**startsWith**(String prefix, int toffset) Testeaza daca subsirul de caractere al sirului curent care debuteaza la indexul toffset incepe cu prefixul specificat ca argument.
String	**substring**(int beginIndex) Returneaza ca nou sir de caractere acel subsir al sirului curent care incepe la indexul beginIndex si se termina la indexul endIndex – 1, lungimea noului sir fiind endIndex-beginIndex.
String	**substring**(int beginIndex, int endIndex) Returneaza ca nou sir de caractere acel subsir al sirului curent care incepe la indexul beginIndex si se termina la ultimul caracter al sirului curent.
char[]	**toCharArray**() Converteste sirul curent la un nou tablou de caractere.

`String`	**`toLowerCase`**`()` Converteste toate caracterele din sirul curent la litere mici, utilizand regulile locale implicite.
`String`	**`toLowerCase`**`(Locale locale)` Converteste toate caracterele din sirul curent la litere mici, utilizand regulile locale specificate.
`String`	**`toString`**`()` Returneaza un obiect `String` nou creat care reprezinta acelasi sir de caractere ca sirul curent (sirul nou creat e o copie a celui curent).
`String`	**`toUpperCase`**`()` Converteste toate caracterele din sirul curent la litere mari, utilizand regulile locale implicite.
`String`	**`toUpperCase`**`(Locale locale)` Converteste toate caracterele din sirul curent la litere mari, utilizand regulile locale specificate.
`static String`	**`valueOf`**`(boolean b)` Returneaza reprezentarea ca sir de caractere a argumentului de tip boolean (`boolean`).
`static String`	**`valueOf`**`(char c)` Returneaza reprezentarea ca sir de caractere a argumentului de tip caracter (`char`).
`static String`	**`valueOf`**`(char[] data)` Returneaza reprezentarea ca sir de caractere a argumentului tablou de caractere (`char`).
`static String`	**`valueOf`**`(char[] data, int offset, int count)` Returneaza reprezentarea ca sir de caractere a subtabloului specificat ca argument.
`static String`	**`valueOf`**`(double d)` Returneaza reprezentarea ca sir de caractere a argumentului `double`.
`static String`	**`valueOf`**`(float f)` Returneaza reprezentarea ca sir de caractere a argumentului `float`.
`static String`	**`valueOf`**`(int i)` Returneaza reprezentarea ca sir de caractere a argumentului de tip intreg (`int`).
`static String`	**`valueOf`**`(long l)` Returneaza reprezentarea ca sir de caractere a argumentului `long`.
`static String`	**`valueOf`**`(Object obj)` Returneaza reprezentarea ca sir de caractere a argumentului `Object`.

Se observa urmatoarele **echivalente functionale**:

- constructorul **`String`**`(char[] value)` cu metoda **`static`** `String` **`valueOf`**`(char[] data)`:

```
char[] caractere = {'t', 'e', 's', 't'};
String sir = new String(caractere);
// echivalent cu    String sir = String.valueOf(caractere);
```

- constructorul **`String`**`(char[] value, int offset, int count)` cu metoda **`static`** `String` **`valueOf`**`(char[] value, int offset, int count)`:

```
char[] caractere = {'t', 'e', 's', 't', 'a', 'r', 'e'};
String sir = new String(caractere, 2, 5);
// echivalent cu    String sir = String.valueOf(caractere, 2, 5);
```

- constructorul **String**(String original) cu metoda String **toString**(), dar si cu alte metode:

```
String original = "sir";
String copie = new String(original);
// echivalent cu    String copie = original.tostring();
// echivalent cu    String copie = String.valueOf(original);
// echivalent cu    String copie = original.substring(0);
// etc.
```

Se observa si urmatoarele **complementaritati functionale**:

- constructorul **String**(byte[] bytes) cu metoda byte[] **getBytes**():

```
String sir = "test";
byte[] octeti = sir.getBytes();
String copieSir = new String(octeti);
```

O **parte din codul clasei String**:

```
1   public final class String { // finala - nu poate fi extinsa prin mostenire
2
3     private char[] value; // tabloul de caractere care stocheaza sirul
4     private int offset;  // indexul primei locatii utilizate pentru stocare
5     private int count;   // numarul de caractere al sirului
6
7     // Diferiti constructori String
8     public String() {
9       value = new char[0];
10    }
11    public String(String value) {
12      count = value.length();
13      this.value = new char[count];
14      value.getChars(0, count, this.value, 0);
15    }
16    public String(char[] value) {
17      this.count = value.length;
18      this.value = new char[count];
19      System.arraycopy(value, 0, this.value, 0, count);
20    }
21
22    // Metode publice, de obiect
23    public boolean equals(Object obj) {
24      if ((obj != null) && (obj instanceof String)) {
25        String otherString = (String)obj;            // cast
26        int n = this.count;
27        if (n == otherString.count) {
28          char v1[] = this.value;
29          char v2[] = otherString.value;;
30          int i = this.offset;
31          int j = otherString.offset;
32          while (n-- != 0)
33            if (v1[i++] != v2[j++]) return false;
34          return true;
35        }
36      }
37      return false;
38    }
39    public int length() {
40      return count;
41    }
42    public static String valueOf(int i) {
43      return Integer.toString(i, 10);
44    }
45    // ... multe alte metode
46  }
```

Exemple de lucru cu obiecte de tip **`String`**.

```
1   // variabile referinta
2
3   String a;         // referinta la String initializata implicit cu null
4
5   String b = null;  // referinta la String initializata explicit cu null
6
7   // constructie siruri de caractere utilizand constructori String()
8
9   String sirVid = new String();       // sirVid.length = 0, sirVid = ""
10
11  byte[] tabByte = {65, 110, 110, 97};         // coduri ASCII
12  String sirTablouByte = new String(tabByte);  // sirTablouByte = "Anna"
13
14  char[] tabChar = {'T', 'e', 's', 't'};
15  String sirTabChar = new String(tabChar);     // sirTabChar = "Test"
16
17  String s = "Sir de caractere";
18  String sir = new String(s);                  // sir = "Sir de caractere"
19
20  // constructie siruri de caractere utilizand metoda toString()
21
22  String sirCopie = sir.toString();
23
24  // constructie siruri de caractere utilizand metode de clasa
25
26  boolean adevarat = true;
27  String sirBoolean = String.valueOf(adevarat);   // sirBoolean = "true"
28  // echivalent cu   String sirBoolean = String.valueOf(true);
29
30  char caracter = 'x';
31  String sirChar = String.valueOf(caracter);      // sirChar = "x"
32
33  char[] tab2Char = {'A', 'l', 't', ' ', 't', 'e', 's', 't'};
34  String sirTab2Char = String.valueOf(tab2Char);  // sirTabChar2="Alt test"
35
36  int numar = 10000;
37  String sirInt = String.valueOf(numar);          // sirInt = "1000"
38
39  double altNumar = 2.3;
40  String sirDouble = String.valueOf(altNumar);    // sirDouble = "2.3"
41
42  // conversia sirurilor de caractere la alte tipuri
43
44  String sirTest = "ABC abc";
45  byte[] sirTestByte = sirTest.getBytes();         // coduri ASCII
46
47  System.out.print("sirTestByte = ");
48  for (int i=0; i < sirTestByte.length; i++)
49     System.out.print(sirTestByte[i] + " ");
50  System.out.println();
51
52  char[] sirTestChar = sirTest.toCharArray();      // caractere UNICODE
53
54  System.out.print("sirTestChar =  ");
55  for (int i=0; i < sirTestChar.length; i++)
56     System.out.print(sirTestChar[i] + "  ");
57  System.out.println();
58
59  // concatenare
60
61  String f = "Sir " + "de " + "caractere";
62  // echivalent cu:  String f = "Sir de caractere";
```

4.1.4. Clasa care incapsuleaza siruri de caractere modificabile (StringBuffer) – interfata publica

Clasa **`StringBuffer`** incapsuleaza **siruri de caractere modificabile** (*mutable*). Un sir de caractere modificabil este asemanator unui **`String`**, dar continutul lui poate fi modificat.

In orice moment el contine un anumit sir de caractere, dar **lungimea si continutul sirului pot fi modificate prin apelul anumitor metode**.

Sirurile de caractere modificabile (**`StringBuffer`**) sunt sigure din punct de vedere al programelor multifilare (*multithreaded*). Metodele clasei `StringBuffer` sunt implicit sincronizate atunci cand e necesar, astfel incat toate operatiile asupra oricarei instante se desfasoara ca si cum ar aparea intr-o ordine seriala (secventiala) consistenta cu ordinea in care apelurile sunt facute de fiecare fir (*thread*) individual implicat.

4.1.4.1. Declaratia clasei StringBuffer

Declaratiile de pachet si clasa ale clasei **`StringBuffer`** sunt urmatoarele:

```
package java.lang;

public final class StringBuffer implements java.io.Serializable, CharSequence {
  // corpul clasei StringBuffer
}
```

4.1.4.2. Constructorii clasei StringBuffer

Constructorii clasei `StringBuffer`:

`StringBuffer``()` Construieste un obiect sir de caractere modificabil (*string buffer*) fara caractere, a carui capacitate initiala este de 16 caractere.
`StringBuffer``(int length)` Construieste un obiect sir de caractere modificabil (*string buffer*) fara caractere, a carui capacitate initiala este specificata de argumentul `length`.
`StringBuffer``(String str)` Construieste un obiect sir de caractere modificabil (*string buffer*) care incapsuleaza acelasi sir de caractere ca argumentul `str` (continutul initialal *bufferului* de caractere este o copie a argumentului.

4.1.4.3. Metodele clasei StringBuffer

Declaratiile si descrierea catorva metode ale clasei **`StringBuffer`**:

`StringBuffer`	**`append`**`(boolean b)` Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului `boolean b`.
`StringBuffer`	**`append`**`(char c)` Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului `char c`.

StringBuffer	**append**(char[] str) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului tablou de caractere str.
StringBuffer	**append**(char[] str, int offset, int len) Adauga la sirul de caractere modificabil curent subsirul de caractere a specificat de argumente.
StringBuffer	**append**(double d) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului double d.
StringBuffer	**append**(float f) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului float f.
StringBuffer	**append**(int i) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului int i.
StringBuffer	**append**(long l) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului long l.
StringBuffer	**append**(Object obj) Adauga la sirul de caractere modificabil curent reprezentarea ca sir de caractere a argumentului Object obj.
StringBuffer	**append**(String str) Adauga la sirul de caractere modificabil curent sirul de caractere primit ca argument.
StringBuffer	**append**(StringBuffer sb) Adauga la sirul de caractere modificabil curent sirul de caractere modificabil primit ca argument.
int	**capacity**() Returneaza capacitatea curenta a sirului de caractere modificabil curent.
char	**charAt**(int index) Returneaza caracterul indicat de argumentul index din sirul de caractere modificabil curent.
StringBuffer	**delete**(int start, int end) Elimina caracterele aflate intre indexul start si indexul end-1 din sirul de caractere modificabil curent.
StringBuffer	**deleteCharAt**(int index) Elimina caracterul aflat la pozitia specificata de argumentul index din sirul de caractere modificabil curent (reducand cu 1 lungimea sirului).
void	**ensureCapacity**(int minimumCapacity) Asigura o capacitate minima a sirului de caractere modificabil curent cel putin egala cu minimul specificat ca argument.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Caracterele specificate (aflate intre indexul srcBeginsi indexul srcEnd-1) sunt copiate din sirul de caractere modificabil curent in tabloul de caractere destinatie dst (incepand de la index dstBegin si pana la index dstbegin + (srcEnd-srcBegin) - 1).
int	**indexOf**(String str) Returneaza indexul minim in sirul de caractere curent la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.

int	**indexOf**(String str, int fromIndex) Returneaza indexul minim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
StringBuffer	**insert**(int offset, boolean b) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului boolean b.
StringBuffer	**insert**(int offset, char c) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului char c.
StringBuffer	**insert**(int offset, char[] str) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului tablou de caractere str.
StringBuffer	**insert**(int index, char[] str, int offset, int len) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, a subsirului de caractere specificat de argumente.
StringBuffer	**insert**(int offset, double d) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului double d.
StringBuffer	**insert**(int offset, float f) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului float f.
StringBuffer	**insert**(int offset, int i) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului int i.
StringBuffer	**insert**(int offset, long l) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului long l.
StringBuffer	**insert**(int offset, Object obj) Inserts Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, reprezentarea ca sir de caractere a argumentului Object obj.
StringBuffer	**insert**(int offset, String str) Insereaza in sirul de caractere modificabil curent, pe pozitia specificata de argumentul offset, sirul de caractere primit ca argument.
int	**lastIndexOf**(String str) Returneaza indexul maxim in sirul de caractere curent la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**lastIndexOf**(String str, int fromIndex) Returneaza indexul maxim in sirul de caractere curent, incepand cu indexul specificat ca argument (fromIndex), la care apare subsirul de caractere specificat ca argument (str), si valoarea -1 daca nu exista acel subsir in sirul de caractere curent.
int	**length**() Returneaza numarul de caractere al sirului de caractere modificabil curent.
StringBuffer	**replace**(int start, int end, String str) Inlocuieste caracterele specificate (aflate intre indexul start si indexul end-1) din sirul de caractere modificabil curent cu sirul de caractere specificat ca argument (sirul de caractere modificabil fiind redimensionat daca este necesar pentru a cuprinde toate caracterele din sirul str).

`StringBuffer`	**`reverse`**`()` Inlocuieste caracterele din sirul de caractere modificabil curent cu sirul de caractere obtinut prin inversarea ordinii caracterelor.
`void`	**`setCharAt`**`(int index, char ch)` Inlocuieste caracterul specificat (prin indexul `index`) din sirul de caractere modificabil curent cu argumentul `ch`.
`void`	**`setLength`**`(int newLength)` Stabileste lungimea sirului de caractere modificabil curent.
`String`	**`substring`**`(int start)` Returneaza un nou obiect `String` care contine subsirul de caractere din sirul de caractere modificabil curent care incepe la indexul `start` si se incheie la sfarsitul sirului de caractere modificabil curent.
`String`	**`substring`**`(int start, int end)` Returneaza un nou obiect `String` care contine subsirul de caractere din sirul de caractere modificabil curent care incepe la indexul `start` si se incheie la indexul `end-1`.
`String`	**`toString`**`()` Converteste la reprezentare sir de caractere (nemodificabil) datele din sirul de caractere modificabil curent.

Sirurile de caractere modificabile (`StringBuffer`) sunt utilizate de compilator pentru a implementa operatorul + de concatenare binara a sirurilor. De exemplu, **codul**:

```
x = "a" + 4 + "c";
```

este compilat ca:

```
x = new StringBuffer().append("a").append(4).append("c").toString();
```

adica:
- **se creaza un nou sir de caractere modificabil** (`StringBuffer`), initial gol,
- **se adauga la sirul de caractere modificabil** (`StringBuffer`) reprezentarea sir de caractere (`String`) a fiecarui operand, si apoi
- **se converteste continutul sirului de caractere modificabil** (`StringBuffer`) la un sir de caractere (`String`).

In acest fel **se evita crearea temporara a mai multor obiecte `String`**.

Principalele operatii asupra unui `StringBuffer` sunt
- **adaugarea la sfarsitul sirului modificabil curent** (metoda **`append()`**) si
- **inserarea intr-o pozitie specificata in sirul modificabil curent** (metoda **`insert()`**),

al caror nume sunt supraincarcate astfel incat accepta date de orice tip.

In general, daca sb este o instanta `StringBuffer`, atunci `sb.append(x)` are acelasi efect ca `sb.insert(sb.length(), x)`.

Utilizarea metodei `insert()`:

```
StringBuffer sb = new StringBuffer("Drink Java!");
sb.insert(6, "Hot ");
System.out.println(sb.toString());
```

Rezultatul executiei programului este urmatorul:

```
Drink Hot Java!
```

Fiecare sir de caractere modificabil are o **capacitate**. Cat timp lungimea sirului de caractere continut nu depaseste capacitatea, nu este necesara alocarea unui nou tablou de caractere intern. Daca este depasita capacitatea acestui tablou, el este in mod automat largit.

Program de inversare a unui sir folosind `String` **si** `StringBuffer`**:**

```
 1    class ReverseString {
 2      public static String reverseIt(String source) {
 3         int i, len = source.length();
 4         StringBuffer dest = new StringBuffer(len);
 5
 6         for (i = (len - 1); i >= 0; i--)
 7            dest.append(source.charAt(i));
 8         return dest.toString();
 9      }
10    }
11    public class StringsDemo {
12      public static void main(String[] args) {
13         String palindrome = "Dot saw I was Tod";
14         String reversed = ReverseString.reverseIt(palindrome);
15         System.out.println(reversed);
16      }
17    }
```

Rezultatul executiei programului este urmatorul:

```
doT saw I was toD
```

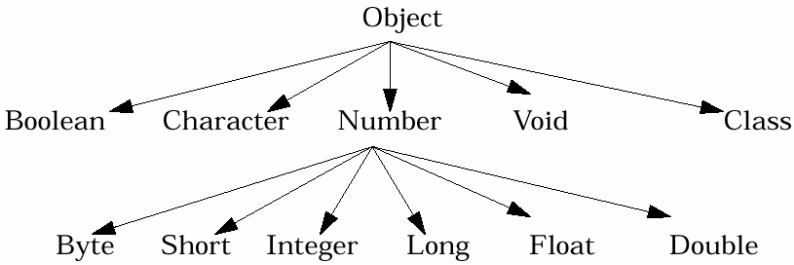
Utilizarea metodei `valueOf()`:

```
String piStr = "3.14159";
Float pi = Float.valueOf(piStr);
```

4.2. Clase predefinite pentru incapsularea tipurilor primitive. Conversii

4.2.1. Incapsularea tipurilor primitive

Ca si caracterele, si tipurile primitive numerice pot fi incapsulate in obiecte al caror continut este nemodificabil (*immutable*) dupa initializare. Iata, mai jos, o parte din ierarhia de clase Java.



Vom analiza clasa **`Integer`**, care incapsuleaza valori intregi primitive de tip **`Integer`**. Celelalte clase care incapsuleaza valori numerice primitive (**`Byte`**, **`Short`**, **`Long`**, **`Float`**, **`Double`**) si clasa care incapsuleaza valori logice primitive (**`Boolean`**) au o interfata si un mod de lucru asemanator.

4.2.2. Clasa care incapsuleaza intregi de tip int (Integer) – interfata publica

Clasa **`Integer`** permite incapsularea valorilor primitive de tip **`int`** in obiecte, dar si conversia intre intregi si alte tipuri primitive. Un obiect de tip **`Integer`** contine un singur atribut (camp) al carui tip este **`int`**.

4.2.2.1. Declaratia clasei Integer

Declaratiile de pachet si clasa ale clasei **`Integer`** sunt urmatoarele:

```
package java.lang;

public final class Integer extends Number implements Comparable {
  // corpul clasei Integer
}
```

4.2.2.2. Constructorii clasei Integer

Constructorii clasei `Integer`:

`Integer``(int value)` Construieste un nou obiect `Integer` care incapsuleaza valoarea de tip `int` specificata ca argument (`value`).
`Integer``(String s)` Construieste un nou obiect `Integer` care reprezinta valoarea de tip `int` indicata de parametrul de tip `String s`.

4.2.3.3. Metodele clasei Integer

Declaratiile si descrierea catorva metode ale clasei **`Integer`**:

`byte`	**`byteValue`**`()` Returneaza sub forma de `byte` valoarea intregului incapsulat in obiectul `Integer` curent.
`int`	**`compareTo`**`(Integer anotherInteger)` Compara numeric obiectul curent cu obiectul primit ca parametru (returnand diferenta dintre intregul incapsulat in obiectul curent si intregul incapsulat in obiectul primit ca parametru). Altfel, metoda arunca o exceptie de tipul `ClassCastException` (obiectele `Integer` fiind comparabile doar intre ele).
`int`	**`compareTo`**`(Object o)` Compara obiectul `Integer` curent cu un alt obiect. Daca obiectul argument este un `Integer` functia se comporta ca `compareTo(Integer)`.
`static Integer`	**`decode`**`(String nm)` Decodeaza un sir de caractere (care contine o valoare literala zecimala, hexazecimala sau octala) intr-un obiect `Integer`.
`double`	**`doubleValue`**`()` Returneaza sub forma de `double` valoarea intregului incapsulat in obiectul `Integer` curent.
`boolean`	**`equals`**`(Object obj)` Compara continutul obiectului curent cu continutul obiectului primit ca parametru.
`float`	**`floatValue`**`()` Returneaza sub forma de `float` valoarea intregului incapsulat in obiectul `Integer` curent.
`int`	**`intValue`**`()` Returneaza sub forma de `int` valoarea intregului incapsulat in obiectul `Integer` curent.
`long`	**`longValue`**`()` Returneaza sub forma de `long` valoarea intregului incapsulat in obiectul `Integer` curent.
`static int`	**`parseInt`**`(String s)` Analizeaza lexical argumentul sir de caractere, returnand intregul zecimal cu semn corespunzator. Metoda arunca exceptia `NumberFormatException` daca argumentul nu respecta formatul unui intreg.
`static int`	**`parseInt`**`(String s, int radix)` Analizeaza lexical argumentul sir de caractere `s`, returnand intregul cu semn corespunzator, in baza indicate de argumentul `radix`. Metoda arunca exceptia `NumberFormatException` daca argumentul nu respecta formatul unui intreg in baza indicata.
`short`	**`shortValue`**`()` Returneaza sub forma de `short` valoarea intregului incapsulat in obiectul `Integer` curent.

`static String`	**`toBinaryString`**`(int i)` Returneaza reprezentarea ca sir de caractere (literala) a argumentului intreg sub forma de intreg fara semn in baza 2.
`static String`	**`toHexString`**`(int i)` Returneaza reprezentarea ca sir de caractere (literala) a argumentului intreg sub forma de intreg fara semn in baza 16.
`static String`	**`toOctalString`**`(int i)` Returneaza reprezentarea ca sir de caractere (literala) a argumentului intreg sub forma de intreg fara semn in baza 8.
`String`	**`toString`**`()` Returneaza reprezentarea ca sir de caractere a valorii incapsulate in obiectul `Integer` curent, convertita la intreg zecimal cu semn.
`static String`	**`toString`**`(int i)` Returneaza reprezentarea ca sir de caractere a valorii intregului de tip `int` primit ca parametru, convertita la intreg zecimal cu semn.
`static String`	**`toString`**`(int i, int radix)` Returneaza reprezentarea ca sir de caractere a intregului de tip `int` primit ca parametru, interpretat in baxa indicata de argumentul `radix`.
`Static Integer`	**`valueOf`**`(String s)` Returneaza un obiect `Integer` care incapsuleaza valoarea specificata de argument, convertita la intreg zecimal cu semn. Metoda este echivalenta cu `new Integer(Integer.parseInt(s))`.
`Static Integer`	**`valueOf`**`(String s, int radix)` Returneaza un obiect `Integer` care incapsuleaza valoarea specificata de argument, interpretat in baxa indicata de argumentul `radix`. Metoda este echivalenta cu `new Integer(Integer.parseInt(s, radix))`.

Metodele clasei `Integer` permit:
- **incapsularea** valorilor intregi primitive (folosind constructori, dar si metode de clasa **`valueOf()`**),
- **comparatii** intre intregi (**`compareTo()`**, **`equals()`**, etc.),
- **conversii** la tipuri numerice primitive (**`byteValue()`**, **`doubleValue()`**, etc.),
- **conversii** la siruri de caractere (**`toString()`**, **`toBinaryString()`**, etc.),
- **conversii** ale sirurilor de caractere la valori intregi primitive (cu metoda de clasa **`parseInt()`**).

Si **celelalte clase care incapsuleaza valori numerice primitive au metode similare metodei `parseInt()`** a clasei `Integer`, cu ajutorul carora pot crea valori numerice primitive din reprezentarile sub forma de siruri de caractere ale valorilor literale. Clasa **`Byte`** are o metoda de clasa **`parseByte()`** care primeste ca parametru un sir de caractere si returneaza valoarea numerica primitiva de tip **`byte`** corespunzatoare, clasa **`Short`** are o metoda de clasa **`parseShort()`**, clasa **`Long`** are o metoda de clasa **`parseLong()`**, clasa **`Float`** are o metoda de clasa **`parseFloat()`**, clasa **`Double`** are o metoda de clasa **`parseDouble()`**, clasa **`Boolean`** are o metoda de clasa **`parseBoolean()`**.

Apelul metodei **`parseInt()`** poate genera exceptie de tip **`NumberFormatException`** in cazul in care argumentul nu are format intreg. In acest caz, **trebuie tratata exceptia** de tip **`NumberFormatException`**, definita in clasa cu acelasi nume din pachetul **`java.lang`**, cu ajutorul unui bloc de tip:

```
try { /* secventa care poate genera exceptia */
}
catch (NumberFormatException ex) {/* secventa care trateaza exceptia */
}
```

Ca in cazul programului urmator:

```
1   public class AfisareArgumenteProgramIntregi2 {
2     public static void main(String[] args) {
3       int i;
4       for ( i=0; i < args.length; i++ ) {
5         try {
6           System.out.println(Integer.parseInt(args[i]));
7         }
8         catch (NumberFormatException ex) {
9           System.out.println("Argumentul '" + args[i] +
10                                        "' nu are format numeric intreg");
11        }
12      }
13    }
14  }
```

Alte **exemple de lucru cu obiecte** de tip **`Integer`**.

```
1   int     i, j, k;      // intregi ca variabile de tip primitiv
2
3   Integer m, n, o;      // intregi incapsulati in obiecte Integer
4
5   String  s, r, t;      // siruri de caractere (incapsulate in obiecte)
6
7   // constructia intregilor incapsulati
8   // utilizand constructori ai clasei Integer - 2 variante
9
10  i = 1000;
11  m = new Integer(i);        // echivalent cu   m = new Integer(1000);
12
13  r = new String("30");
14  n = new Integer(r);        // echivalent cu   n = new Integer("30");
15
16  // constructia intregilor incapsulati
17  // utilizand metode de clasa ale clasei Integer
18
19  t = "40";
20  o = Integer.valueOf(t);    // echivalent cu   o = new Integer("40");
21
22  // conversia intregilor incapsulati la valori numerice primitive
23
24  byte  iByte = m.byteValue();         // diferit de 1000! (trunchiat)
25  short iShort = m.shortValue();       // = 1000
26  int   iInt = m.intValue();           // = 1000
27  long  iLong = m.longValue();         // = 1000L
28
29  float iFloat = m.floatValue();       // = 1000.0F
30  double iDouble = m.doubleValue();    // = 1000.0
31
32  // conversia intregilor incapsulati la obiecte sir de caractere
33
34  String iString = m.toString();       // metoda de obiect (non-statica)
35
36  // conversia valorilor intregi primitive la siruri de caractere
37
38  String douaSute = Integer.toString(200);  // metoda de clasa (statica)
39
40  String oMieBinary = Integer.toBinaryString(1000); // metoda de clasa
41  String oMieOctal = Integer.toOctalString(1000);   // metoda de clasa
42  String oMieHex = Integer.toHexString(1000);       // metoda de clasa
43
44  // conversia sirurilor de caractere la valori intregi primitive
45
46  int oSuta = Integer.parseInt("100");       // metoda de clasa (statica)
```

4.3. Clase Java pentru operatii de intrare-iesire (IO)

Programele pot avea nevoie de a:

- **prelua** informatii **de la surse** externe, sau
- **trimite** informatii **catre destinatii** externe.

Sursa/destinatia poate fi: ***fisier pe disc, retea, memorie (alt program), dispozitive IO standard (ecran, tastatura)***.

Tipurile informatiilor sunt diverse: ***caractere, obiecte, imagini, sunete.***

Pentru **preluarea** informatiilor programul **deschide un flux de la o sursa** de informatii si **citeste serial** (secvential) informatiile, astfel:

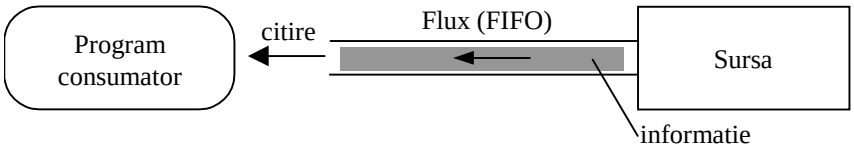


Figura 1. Fluxuri de intrare (citire) Java

Pentru **trimiterea** informatiei programul **deschide un flux catre o destinatie** de informatie si **scrie serial** (secvential) informatiile, astfel:

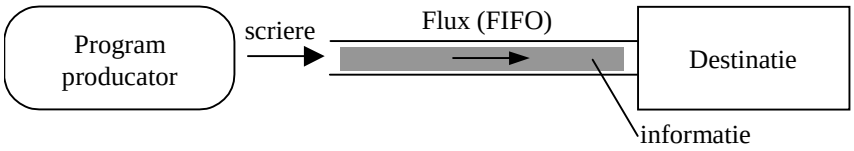


Figura 2. Fluxuri de iesire (scriere) Java

4.3.1. Clasificarea fluxurilor IO in functie de tipul de data transferate

In functie de **tipul de date** transferate, clasele din pachetul **`java.io`** se impart in**:**

- fluxuri **de caractere** (date reprezentate in UNICODE pe **16b**), avand ca radacini ale arborilor de clase derivate superclasele abstracte **`Reader`** (de intrare) si **`Writer`** (de iesire)
- fluxuri **de octeti** (date reprezentate pe **8b**), avand ca radacini ale arborilor de clase derivate superclasele abstracte **`InputStream`** (de intrare) si **`OutputStream`** (de iesire)

4.3.1.1. Constructorii claselor Reader si Writer

Constructorii clasei `Reader`:

`protected`	**`Reader`**`()` Creaza un nou flux de citire a caracterelor ale carui sectiuni critice se vor sincroniza pe fluxul insusi.
`protected`	**`Reader`**`(Object lock)` Creaza un nou flux de citire a caracterelor ale carui sectiuni critice se vor sincroniza pe obiectul specificat ca parametru.

Constructorii clasei `Writer`:

`protected`	**`Writer`**`()` Creaza un nou flux de scriere a caracterelor ale carui sectiuni critice se vor sincroniza pe fluxul insusi.
`protected`	**`Writer`**`(Object lock)` Creaza un nou flux de scriere a caracterelor ale carui sectiuni critice se vor sincroniza pe obiectul specificat ca parametru.

4.3.1.2. Metodele claselor Reader si Writer

Declaratiile si descrierea catorva metode ale clasei **`Reader`**:

`int`	**`read`**`()` Citeste din flux un caracter. Metoda se blocheaza pana cand este disponibil un caracter (caz in care returneaza caracterul ca intreg in intervalul de la 0 la 65535, adica de la `0x0000` la `0xffff`), apare o eroare I/O (caz in care arunca o exceptie `IOException`), sau este atins sfarsitul fluxului (caz in care retureaza `-1`).
`int`	**`read`**`(char[] cbuf)` Citeste din flux caractere in tabloul de caractere primit ca parametru. Metoda se blocheaza pana cand sunt disponibile caractere, apare o eroare I/O, sau este atins sfarsitul fluxului.
`abstract int`	**`read`**`(char[] cbuf, int off, int len)` Citeste din flux caractere in subtabloul de caractere specificat de parametrii primiti. Metoda se blocheaza pana cand sunt disponibile caractere, apare o eroare I/O, sau este atins sfarsitul fluxului.
`abstract void`	**`close`**`()` Inchide fluxul curent.
`void`	**`mark`**`(int readAheadLimit)` Marcheaza pozitia curenta in flux.
`boolean`	**`markSupported`**`()` Returneaza o valoare logica indicand existenta suportului pentru operatia `mark()`. Implementarea implicita returneaza intotdeauna `false`, subclasele urmand sa rescrie aceasta metoda daca se doreste sa returneze `true`.
`boolean`	**`ready`**`()` Returneaza o valoare logica indicand daca fluxul curent este pregatit pentru a fi citit (caz in care urmatorul apel `read()` sigur nu se blocheaza).
`void`	**`reset`**`()` Reseteaza fluxul. Daca fluxul a fost marcat, se incearca repozitionarea pe marcaj. Daca fluxul nu a fost marcat, se incearca resetarea lui intr-un mod potrivit fiecarui tip particular de flux, de exemplu repozitionand fluxul pe punctual de start. Nu toate fluxurile de intrare a caracterelor suporta operatia `reset()`, iar unele and suporta `reset()` fara a suporta `mark()`.
`long`	**`skip`**`(long n)` Sare peste (si elimina) numarul specificat de caractere. Metoda se blocheaza pana cand sunt disponibile caractere, apare o eroare I/O, sau este atins sfarsitul fluxului.

Declaratiile si descrierea catorva metode ale clasei **`Writer`**:

`void`	**`write`**`(int c)` Scrie in flux caracterul continut in cei mai putin semnificativi 16 biti ai intregului (de 32 de biti) primit ca parametru (sunt neglijati cei mai semnificativi 16 biti ai intregului).
`void`	**`write`**`(char[] cbuf)` Scrie in flux caracterele din tabloul de caractere primit ca parametru.
`abstract void`	**`write`**`(char[] cbuf, int off, int len)` Scrie in flux caracterele din subtabloul de caractere specificat de parametri.
`void`	**`write`**`(String str)` Scrie in flux caracterele din sirul de caractere primit ca parametru.
`void`	**`write`**`(String str, int off, int len)` Scrie in flux caracterele din subsirul de caractere specificat de parametri.
`abstract void`	**`flush`**`()` Goleste fluxul (forteaza trimiterea datelor lui catre destinatie, in cazul in care erau stocate temporar intr-un *buffer*). Daca destinatia este un alt flux, metoda `flush()` pentru fluxul curent duce la apelul metodei `flush()` pentru fluxul destinatie. Astfel, invocarea metodei `flush()`duce la golirea *bufferelor* tuturor fluxurilor inlantuite pana la destinatia finala propriu-zisa.
`abstract void`	**`close`**`()` Inchide fluxul, fortand mai intai golirea lui (*flushing*).

Toate metodele au caracter `public` si arunca exceptia `IOException`.

4.3.1.3. Constructorii claselor InputStream si OutputStream

Constructorul clasei `InputStream`:

`InputStream``()` Creaza un nou obiect al clasei `InputStream` fara a face alte initializari.

Constructorul clasei `OutputStream`:

`OutputStream``()` Creaza un nou obiect al clasei `OutputStream` fara a face alte initializari.

4.3.1.4. Metodele claselor InputStream si OutputStream

Declaratiile si descrierea catorva metode ale clasei **`InputStream`**:

`int`	**`available`**`()` Returneaza numarul de octeti care pot fi cititi (sau peste care se poate sari) din fluxul de intrare curent, fara blocare la urmatorul apel al unei metode pentru fluxul curent.

`abstract int`	**`read`**`()` Returneaza urmatorul octet de date din fluxul de intrare (returneaza valori intre `0` si `255`). Daca este detectata atingerea sfarsitului de flux, valoarea returnata este `-1`. Metoda se blocheaza pana cand sunt disponibili octeti, apare o eroare I/O, sau este atins sfarsitul fluxului.
`int`	**`read`**`(byte[] b)` Citeste un numar de octeti de date din fluxul de intrare (returneaza valori intre `0` si `255`) si ii plaseaza in tabloul de octeti primit ca parametru, incepand de la indexul `0`. Daca este detectata atingerea sfarsitului de flux, valoarea returnata este `-1`. Metoda se blocheaza pana cand sunt disponibili octeti, apare o eroare I/O, sau este atins sfarsitul fluxului.
`int`	**`read`**`(byte[] b, int off, int len)` Citeste cel mult `len` octeti de date din fluxul de intrare (returneaza valori intre `0` si `255`) si ii plaseaza in tabloul de octeti primit ca parametru, incepand de la indexul `off`. Daca este detectata atingerea sfarsitului de flux, valoarea returnata este `-1`. Metoda se blocheaza pana cand sunt disponibili octeti, apare o eroare I/O, sau este atins sfarsitul fluxului.
`long`	**`skip`**`(long n)` Sare peste (si elimina) cel mult numarul specificat de octeti (este returnat numarul exact al octetilor eliminati). Metoda se blocheaza pana cand sunt disponibili octeti, apare o eroare I/O, sau este atins sfarsitul fluxului.
`void`	**`reset`**`()` Reseteaza fluxul curent. Daca fluxul a fost marcat, se incearca repozitionarea pe marcaj. Daca fluxul nu a fost marcat, se incearca resetarea lui intr-un mod potrivit fiecarui tip particular de flux, de exemplu repozitionand fluxul pe punctual de start.
`void`	**`mark`**`(int readlimit)` Marcheaza pozitia curenta in fluxul curent.
`boolean`	**`markSupported`**`()` Returneaza o valoare logica indicand existenta suportului pentru operatia `mark()`.
`void`	**`close`**`()` Inchide fluxul curent si elibereaza toate resursele sale.

Declaratiile si descrierea catorva metode ale clasei **`OutputStream`**:

`abstract void`	**`write`**`(int b)` Scrie in flux octetul continut in cei mai putin semnificativi 8 biti ai intregului (de 32 de biti) primit ca parametru (sunt neglijati cei mai semnificativi 24 biti ai intregului).
`void`	**`write`**`(byte[] b)` Scrie in flux octetii din tabloul de octeti primit ca parametru.
`void`	**`write`**`(byte[] b, int off, int len)` Scrie in flux octetii din subtabloul de octeti specificat de parametri.
`void`	**`flush`**`()` Goleste fluxul (forteaza trimiterea datelor lui catre destinatie, in cazul in care erau stocate temporar intr-un *buffer*). Daca destinatia este un alt flux, metoda `flush()` pentru fluxul curent duce la apelul metodei `flush()` pentru fluxul destinatie.
`void`	**`close`**`()` Inchide fluxul curent si elibereaza toate resursele sale.

4.3.2. Clasificarea fluxurilor IO in functie de specializare

In functie de **specializarea pe care o implementeaza**, subclasele claselor abstracte **`Reader`**, **`Writer`**, **`InputStream`**, si **`OutputStream`** se impart in **doua categorii:**

- fluxuri **terminale** (*data sink*), care **nu au ca sursa / destinatie alte fluxuri**, ci:
  - ***fisierele,***
  - ***memoria (tablourile),***
  - ***reteaua (socketurile),***
  - ***sirurile de caractere (String),***
  - ***alte programe (prin conducte - pipes)***

- fluxuri **de prelucrare** (*processing*), care **au ca sursa / destinatie alte fluxuri**, si au ca rol prelucrarea informatiilor:
  - ***buffer-are (stocare temporara),***
  - ***filtrare de diferite tipuri (conversie, contorizare, etc.)***
  - ***tiparire.***

4.3.3. Fluxuri terminale (*data sink*)

Mai jos sunt prezentate **tipurile de fluxuri Java terminale**.

Tip de Terminal	Utilizare	Fluxuri de caractere	Fluxuri de octeti
Memorie	*Accesul secvential la tablouri*	**`CharArray`**`Reader`	**`ByteArray`**`InputStream`
		`CharArray``Writer`	**`ByteArray`**`OutputStream`
	Accesul secvential la siruri de caractere	**`String`**`Reader`	**`StringBuffer`**`InputStream`
		`String``Writer`	**`StringBuffer`**`OutputStream`
Canal / conducta (*pipe*)	*Conducte intre programe*	**`Piped`**`Reader`	**`Piped`**`InputStream`
		`Piped``Writer`	**`Piped`**`OutputStream`
Fisier	*Accesul la fisiere*	**`File`**`Reader`	**`File`**`InputStream`

4.3.3.1. Fluxul de intrare a octetilor din tablou de octeti (`ByteArrayInputStream`)

Un **`ByteArrayInputStream` contine un *buffer* (tablou de octeti) intern care contine octetii ce pot fi cititi din flux** (sursa fluxului este tabloul de octeti intern). Un contor intern este utilizat pentru a determina care este urmatorul octet ce trebuie oferit metodei `read()`.

Inchiderea unui `ByteArrayInputStream` nu are nici un efect vizibil. Metodele acestei clase pot fi apelate si dupa ce fluxul a fost inchis, fara a se genera o exceptie `IOException`.

Constructorii clasei `ByteArrayInputStream`:

`ByteArrayInputStream``(byte[] buf)` Creaza un obiect `ByteArrayInputStream` care utilizeaza `buf` ca tablou de octeti intern. Tabloul nu este copiat, ci se pastreaza o referinta interna catre el.
`ByteArrayInputStream``(byte[] buf, int offset, int length)` Creaza un obiect `ByteArrayInputStream` care utilizeaza `length` octeti din tabloul `buf` ca tablou de octeti intern, incepand de la indexul `offset`.

Declaratiile si descrierea catorva metode ale clasei **`ByteArrayInputStream`**:

`int`	**`available`**`()` Returneaza numarul de octeti care pot fi cititi (sau eliminati) din fluxul de intrare curent, fara blocare la urmatorul apel al unei metode pentru fluxul curent.
`int`	**`read`**`()` Returneaza urmatorul octet de date din fluxul de intrare (valori intre `0` si `255`).
`int`	**`read`**`(byte[] b, int off, int len)` Citeste cel mult `len` octeti de date din fluxul de intrare (returneaza valori intre `0` si `255`) si ii plaseaza in tabloul de octeti primit ca parametru, incepand de la indexul `off`.
`long`	**`skip`**`(long n)` Elimina cel mult `n` octeti (returneaza numarul exact al octetilor eliminati).
`void`	**`reset`**`()` Reseteaza fluxul curent la pozitia marcata.
`void`	**`mark`**`(int readAheadLimit)` Marcheaza pozitia curenta in fluxul curent.
`boolean`	**`markSupported`**`()` Returneaza o valoare logica indicand existenta suportului pentru operatia `mark()`.
`void`	**`close`**`()` Inchiderea unui `ByteArrayInputStream` nu are nici un effect (**!**).

4.3.3.2. Fluxul de iesire a octetilor catre tablou de octeti (`ByteArrayOutputStream`)

Un **`ByteArrayOutputStream` este un flux de iesire al octetilor care scrie datele intr-un tablou de octeti intern** (destinatia fluxului este tabloul de octeti intern). Datele pot fi regasite utilizand metodele **`toByteArray()`** (care returneaza tabloul de octeti) si `toString()`.

Inchiderea unui `ByteArrayOutputStream` nu are nici un efect vizibil. Metodele acestei clase pot fi apelate si dupa ce fluxul a fost inchis, fara a se genera o exceptie `IOException`.

Constructorii clasei `ByteArrayOutputStream`:

`ByteArrayOutputStream``()` Creaza un obiect `ByteArrayOutputStream` care utilizeaza un tablou de octeti intern de lungime initiala 32 octeti, dar a carui lungime poate creste daca este necesar.
`ByteArrayOutputStream``(int size)` Creaza un obiect `ByteArrayOutputStream` care utilizeaza un tablou de octeti intern cu lungimea initiala specificata de parametrul `size`, dar a carui lungime poate creste daca este necesar.

Atributele clasei `ByteArrayOutputStream`:

`protected byte[]`	**`buf`** Tabloul de octeti intern (*bufferul*) in care sunt stocate datele.
`protected int`	**`count`** Numarul octetilor valizi din tabloul de octeti intern.

Declaratiile si descrierea catorva metode ale clasei **`ByteArrayOutputStream`**:

`void`	**`write`**`(byte[] b, int off, int len)` Scrie in tabloul de octeti intern `len` octeti din tabloul specificat ca parametru (`b`) incepand de la indexul `off`.
`void`	**`write`**`(int b)` Scrie in tabloul de octeti intern octetul specificat ca parametru (de fapt, octetul cel mai putin semnificativ al intregului `b`).
`void`	**`writeTo`**`(OutputStream out)` Scrie intregul continut al tabloului de octeti intern catre fluxul de octeti de iesire specificat ca parametru, ca sic and s-ar utiliza apelul `out.write(buf, 0, count)`.
`void`	**`reset`**`()` Reseteaza atributul `count` la zero, astfel incat octetii acumulati in tabloul intern sunt eliminati.
`int`	**`size`**`()` Returneaza lungimea curenta a tabloului de octeti intern.
`byte[]`	**`toByteArray`**`()` Returneaza un nou tablou de octeti, copie a celui intern.
`String`	**`toString`**`()` Returneaza un sir de caractere care corespunde octetilor din tabloul de octeti intern, folosind codarea implicita a platformei.
`String`	**`toString`**`(String enc)` Returneaza un sir de caractere care corespunde octetilor din tabloul de octeti intern, folosind codarea specificata ca parametru.
`void`	**`close`**`()` Inchiderea unui `ByteArrayOutputStream` nu are nici un effect (**!**).

4.3.3.3. Lucrul cu fluxuri fisier

In continuare sunt ilustrate:

- **citirea dintr-un fisier** prin intermediul unui **flux de caractere** (Unicode!):

```
1    // Crearea unui obiect referinta la fisier pe baza numelui fisierului
2    File inputFile = new File("nume1.txt");
3
4    // Crearea unui flux de intrare a caracterelor dinspre fisierul dat
5    FileReader in = new FileReader(inputFile);
6
7    // Citirea unui caracter din fisier
8    int c = in.read();         // Input
9
10   // Inchiderea fisierului
11   in.close();
```

- **scrierea intr-un fisier** prin intermediul unui **flux de caractere**:

```
1    // Crearea unui obiect referinta la fisier pe baza numelui fisierului
2    File outputFile = new File("nume2.txt");
3
4    // Crearea unui flux de iesire a caracterelor spre fisierul dat
5    FileWrite out = new FileWriter(outputFile);
6
7    char c = 'x';
8    // Scrierea unui caracter in fisier
9    out.write(c);     // Output
10
11   // Inchiderea fisierului
12   out.close();
```

4.3.4. Fluxuri de prelucrare

Mai jos sunt prezentate **tipurile de fluxuri Java de prelucrare**.

Tip de Prelucrare	**Utilizare**	**Fluxuri de Caractere**	**Fluxuri de octeti**
Buffer-are	*Stocare temporară*	**Buffered**Reader	**Buffered**InputStream
		BufferedWriter	**Buffered**OutputStream
Filtrare	*Prelucrare*	**Filter**Reader	**Filter**InputStream
		FilterWriter	**Filter**OutpuStream
Conversie octet/caracter	*Bridge byte-char*	**InputStream**Reader	
		OutputStreamWriter	
Concatenare	*Prelucrare*		**Sequence**InputStream
Serializarea obiectelor			**Object**InputStream
			ObjectOutputStream
Conversia datelor	*Acces la tip date primitiv Java*		**Data**InputStream
			DataOutputStream
Numararea (contorizarea)	*Numarare linii*	**LineNumber**Reader	**LineNumber**InputStream
Testare	*Buffer de 1 byte/char*	**PushBock**Reader	**Pushback**InputStream
Imprimare	*Tiparire*	**Print**Writer	**Print**Stream

4.3.4.1. Fluxul de intrare a caracterelor cu stocare temporara (`BufferedReader`)

Un **`BufferedReader` citeste caracterele din fluxul din amonte** (primit ca parametru de constructor in momentul initializarii) **si le stocheaza temporar pentru a fi citite eficient** caractere, tablouri sau **linii de text**.

Constructorii clasei **`BufferedReader`**:

`BufferedReader``(Reader in)` Creaza un flux de intrare a caracterelor cu stocare temporara (si posibilitate de citire a caracterelor sub forma de linii) din fluxul de intrare a caracterelor primit ca parametru (`in`), utilizand dimensiunea implicita a tabloului intern.
`BufferedReader``(Reader in, int size)` Creaza un flux de intrare a caracterelor cu stocare temporara (si posibilitate de citire a caracterelor sub forma de linii) din fluxul de intrare a caracterelor primit ca parametru (`in`), utilizand dimensiunea specificata ca parametru pentru tabloul intern (`size`).

Metoda oferita de clasa **`BufferedReader`** pentru **citirea liniilor de text** este:

`String`	**`readLine`**`()` Citeste o linie de text (citeste din bufferul intern pana la intalnirea caracterului care semnaleaza terminarea liniei). Se blocheaza in asteptarea caracterului care semnaleaza terminarea liniei.

4.3.4.2. Fluxul de intrare convertor al octetilor la caractere (`InputStreamReader`)

Un **`InputStreamReader`** este un *bridge* (convertor) de la octeti la caractere. El citeste octetii din fluxul din amonte (primit ca parametru de constructor in momentul initializarii) si ii decodeaza folosind un set de caractere dat (`charset`). Daca nu este specificat un set anume, este folosit in mod implicit cel implicit al platformei pe care se lucreaza.

Constructorul tipic al clasei **`InputStreamReader`** este:

`InputStreamReader``(InputStream in)` Creaza un flux de intrare a caracterelor obtinute prin conversia octetilor primiti de la fluxul de intrare primit ca parametru (`in`) utilizand setul de caractere implicit.

Pentru eficienta maxima, este recomandata inlantuirea (plasarea in cascada) a unui **`InputStreamReader`** si a unui **`BufferedReader`**. De exemplu:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

O utilizare tipica a fluxurilor de prelucrare **`BufferedReader`** si **`InputStreamReader`** plasate **in cascada**, este cea care permite **citirea sirurilor de caractere de la consola standard de intrare** (tastatura, care este **incapsulata in obiectul `System.in`**, al carui tip este **`InputStream`** - flux de intrare a octetilor).

Exemplu de citire a unui nume de la tastatura:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Introduceti numele: ");
String nume = in.readLine();
```

4.3.4.3. Fluxul de iesire a caracterelor cu stocare temporara (`BufferedWriter`)

Un **`BufferedWriter` stocheaza temporar caracterele ce urmeaza a fi scrise in fluxul din aval** (primit ca parametru de constructor in momentul initializarii) **pentru a fi scrise eficient** caractere, tablouri sau **siruri de caractere**.

Constructorii clasei **`BufferedWriter`**:

`BufferedWriter``(Writer out)` Creaza un flux de iesire a caracterelor cu stocare temporara inaintea scrierii lor in fluxul de iesire a caracterelor primit ca parametru (`out`), utilizand dimensiunea implicita a tabloului intern.
`BufferedWriter``(Writer out, int size)` Creaza un flux de iesire a caracterelor cu stocare temporara inaintea scrierii lor in fluxul de iesire a caracterelor primit ca parametru (`out`), utilizand dimensiunea specificata ca parametru pentru tabloul intern (`size`).

Metoda oferita de clasa **`BufferedWriter`** pentru **scrierea separatorilor de linie de text** este:

`void`	**`newLine`**`()` Scrie un separator de linie.

4.3.4.4. Fluxul de iesire convertor al caracterelor la octeti (`OutputStreamWriter`)

Un **`OutputStreamWriter`** este un *bridge* (convertor) de la caractere la octeti. El scrie in fluxul din aval (primit ca parametru de constructor in momentul initializarii) octeti obtinuti prin codarea caracterelor, folosind un set de caractere dat (`charset`). Daca nu este specificat un set anume, este folosit in mod implicit cel implicit al platformei pe care se lucreaza.

Constructorul tipic al clasei **`OutputStreamWriter`** este:

`OutputStreamWriter``(OutputStream out)` Creaza un flux de iesire a caracterelor care vor fi convertite la octeti inaintea scrierii lor in fluxul de iesire primit ca parametru (`out`) utilizand setul de caractere implicit.

Pentru eficienta maxima, este recomandata inlantuirea (plasarea in cascada) a unui **`OutputStreamWriter`** si a unui **`BufferedWriter`**. De exemplu:

```
BufferedWriter out = new BufferedWriter (new OutputStreamWriter(System.out));
```

4.3.4.5. Fluxul de iesire a octetilor pentru afisare (`PrintStream`)

Constructorul tipic al clasei **`PrintStream`** este:

`PrintStream``(OutputStream out)` Creaza un flux de iesire a octetilor pentru afisarea lor de catre fluxul de iesire a octetilor primit ca parametru (`out`).

Metodele oferite de clasa **`PrintStream`** pentru afisarea sirurilor de caractere (scrierea intr-un flux de iesire a octetilor a sirurilor de caractere care cuprind si caractere *escape*) sunt:

`void`	**`print`**`(boolean b)` Afiseaza valoarea booleana primita ca parametru.
	… (Similar pentru celelalte tipuri de date primitive Java: char, double, float, int, …)
`void`	**`print`**`(String s)` Afiseaza sirul de caractere primit ca parametru.
`void`	**`println`**`()` Termina linia curenta, adaugand un separator de linie.
`void`	**`println`**`(boolean x)` Afiseaza valoarea booleana primita ca parametru si apoi termina linia.
	… (Similar pentru celelalte tipuri de date primitive Java: char, double, float, int, …)
`void`	**`println`**`(String x)` Afiseaza sirul de caractere primit ca parametru si apoi termina linia.

O utilizare tipica acestui fluxuri de prelucrare este cea care permite **scrierea sirurilor de caractere la consola standard de iesire** (monitorul, care este **incapsulat intr-un `OutputStream` ce serveste ca destinatie obiectului `System.out`**, al carui tip este **`PrintStream`**).

Exemplu: afisarea argumentelor programului curent:

```
PrintStream ps = System.out;
ps.println("Argumentele programului: ");
for (int i=0; i<args.length; i++) {
  ps.print(args[i] + " ");
}
ps.println();
```

4.3.4.6. Fluxul de intrare a octetilor pentru citirea valorilor primitive (`DataInputStream`)

Clasa **`DataInputStream`** permite **citirea datelor formatate (ca tipuri primitive)** de la fluxul de intrare a octetilor primit ca parametru in momentul constructiei (fluxul din amonte).

Constructorul clasei **`DataInputStream`** este:

`DataInputStream``(InputStream in)` Creaza un flux de intrare a octetilor care permite citirea datelor formatate (ca tipuri primitive) de la fluxul de intrare a octetilor primit ca parametru (`in`).

Metodele oferite de clasa **`DataInputStream`** pentru citirea datelor formatate (ca tipuri primitive) de la fluxul de intrare a octetilor primit ca parametru in momentul constructiei sunt:

`boolean`	**`readBoolean`**`()` Citeste un octet din fluxul de intrare a octetilor primit ca parametru in momentul constructiei (din amonte), si returneaza `true` daca este nenul si `false` daca este nul.
`byte`	**`readByte`**`()` Citeste si returneaza un octet.
`char`	**`readChar`**`()` Citeste si returneaza un `char`.
`double`	**`readDouble`**`()` Citeste 8 octeti si returneaza un `double`.
`float`	**`readFloat`**`()` Citeste 4 octeti si returneaza un `float`.
`void`	**`readFully`**`(byte[] b)` Citeste octetii disponibili si ii stocheaza in tabloul primit ca parametru. Metoda se blocheaza pana cand `b.length` octeti sunt disponibili.
`void`	**`readFully`**`(byte[] b, int off, int len)` Citeste `len` octeti si ii stocheaza in tabloul primit ca parametru incepand de la indexul `off`. Metoda se blocheaza pana cand `len` octeti sunt disponibili.
`int`	**`readInt`**`()` Citeste 4 octeti si returneaza un `int`.

`long`	**`readLong`**`()` Citeste 8 octeti si returneaza un `long`.
`short`	**`readShort`**`()` Citeste 2 octeti si returneaza un `short`.
`int`	**`readUnsignedByte`**`()` Citeste un octet, il extinde la tip `int` adaugand 3 octeti nuli, si returneaza rezultatul, care este in gama `0` la `255`.
`int`	**`readUnsignedShort`**`()` Citeste 2 octeti, ii extinde la tip `int` adaugand 2 octeti nuli, si returneaza rezultatul, care este in gama `0` la `65535`.
`String`	**`readUTF`**`()` Citeste un sir de caractere care a fost codat utilizand un format UTF-8 modificat.
`static` `String`	**`readUTF`**`(DataInput in)` Citeste din fluxul primit ca parametru (`in`) o reprezentare a caracterului Unicode codat in Java folosind formatul UTF-8 modificat; si returneaza ca `String` sirul de caractere rezultat.
`int`	**`skipBytes`**`(int n)` Incearca sa arunce (sa sara peste) numarul de octeti specificat ca parametru din fluxul de intrare primit ca parametru in momentul constructiei. Metoda se blocheaza pana cand `n` octeti sunt disponibili.
`String`	**`readLine()`** **Metoda nerecomandata (*deprecated*) pentru citirea sirurilor de caractere terminate cu separator de linie.**

Exemplu: citirea unui nume de la tastatura:

```
DataInputStream in = new DataInputStream(new BufferedInputStream(System.in));
System.out.println("Introduceti numele: ");
String nume = in.readLine();
```

4.3.4.7. Fluxul de iesire a octetilor pentru scrierea valorilor primitive (`DataOutputStream`)

Clasa **`DataOutputStream`** permite scrierea datelor formatate (ca tipuri primitive) in fluxul de iesire a octetilor primit ca parametru in momentul constructiei (fluxul din aval).

Constructorul clasei **`DataOutputStream`** este:

`DataOutputStream``(OutputStream out)` Creaza un flux de iesire a octetilor care permite scrierea datelor formatate (ca tipuri primitive) catre fluxul de iesire a octetilor primit ca parametru (`out`).

Metodele oferite de clasa **`DataOutputStream`** pentru scrierea datelor formatate (ca tipuri primitive) in fluxul de iesire a octetilor primit ca parametru in momentul constructiei (din aval) sunt:

void	**flush**() Forteaza trimiterea datelor scrise in acest flux de iesire catre fluxul de iesire (din aval) primit ca parametru in momentul constructiei.
void	**writeBoolean**(boolean v) Scrie valoarea booleana primita ca parametru in fluxul de iesire (din aval) primit ca parametru in momentul constructiei, ca octet.
void	**writeByte**(int v) Scrie octetul specificat ca parametru (cei mai putin semnificativi 8 biti ai argumentului v) in fluxul de iesire (din aval) primit ca parametru in momentul constructiei.
void	**writeBytes**(String s) Scrie ca secventa de octeti sirul de caractere specificat ca parametru, in fluxul de iesire din aval.
void	**writeChar**(int v) Scrie cei doi octeti ai caracterului Unicode specificat ca parametru (cei mai putin semnificativi 16 biti ai argumentului v) in fluxul de iesire din aval.
void	**writeChars**(String s) Scrie ca secventa de caractere sirul de caractere specificat ca parametru, in fluxul de iesire din aval.
void	**writeDouble**(double v) Converteste argumentul double la un long utilizand metoda doubleToLongBits() din clasa Double, apoi scrie valoarea rezultata ca 8 octeti in fluxul de iesire din aval, cel mai semnificativ octet primul.
void	**writeFloat**(float v) Converteste argumentul float la un int utilizand metoda floatToIntBits() din clasa Float, apoi scrie valoarea rezultata ca 4 octeti in fluxul de iesire din aval, cel mai semnificativ octet primul.
void	**writeInt**(int v) Scrie valoarea specificata ca parametru int ca 4 octeti in fluxul de iesire din aval, cel mai semnificativ octet primul.
void	**writeLong**(long v) Scrie valoarea specificata ca parametru long ca 8 octeti in fluxul de iesire din aval, cel mai semnificativ octet primul.
void	**writeShort**(int v) Scrie valoarea specificata ca parametru short ca 2 octeti in fluxul de iesire din aval, cel mai semnificativ octet primul.
void	**writeUTF**(String str) Scrie sirul de caractere specificat ca parametru in fluxul de iesire (din aval) primit ca parametru in momentul constructiei utilizand codarea UTF-8 modificata a Java, intr-o forma independenta de masina de calcul.

Exemplu: Afisarea argumentelor programului curent:

```
DataOutputStream dos = new DataOutputStream(System.out);
dos.writeBytes("Argumentele programului: \n");
for (int i=0; i<args.length; i++) {
  dos.writeBytes(args[i] + " ");
}
dos.writeChar('\n');
dos.flush();
```

4.3.4.8. Exemple de lucru cu fluxuri IO pentru formatarea valorilor primitive

Scrierea intr-un fisier cu `DataOutputStream`:

```
DataOutputStream dos = new DataOutputStream(new FileOutputStream("invoice.txt"));

 for ( i=0;i<price.length;i++ ) {
    dos.write Double( price[i] );  //preturi
    dos.write Char( `\t` );         //tab
    dos.write Int( units[i] );      //bucati
    dos.write Char( `\t` );         //tab
    dos.write Chars( desers[i] );  //articole
    dos.write Char( `\u` );     }   //linie noua
 }
```

Continutul lui `invoice.txt` este:

```
10.5  10 ……
20.5  15 ……
…..   …. ……
```

Citirea dintr-un fisier cu `DataInputStream`:

```
DataInputStream dis = new DataInputStream(new FileInputStream ("invoice.txt"));
try { while(true){
       price = dis.readDouble( );  // pret
       dis.readChar( );            // salt peste tab
       unit = dis.readInt( );      // bucati
       dis.readChar( );            // salt peste tab
       desc = dis.readLine( );     // articol
       System.out.println(" Ati comandat" + unit + "bucati de" + desc );
       total = total + unit * price;
      }
}
catch ( EOF Exception e )
{ }
System.out.println( "Pentru un pret total de $" + total );
dis.close( );

while (( input = dis.readline( )) != null ) {
    // ……
}
```

Rezultat

```
Ati comandat 10 bucati de ….   la  $10.5
Ati comandat 15 bucati de ….   la $20.5
……..
Pentru un pret total de: $ …..
```