

# Lucrarea 1

## Concepte de bază ale programării orientate pe obiecte

### Cuprins

Concepte fundamentale în programarea orientată pe obiecte .....	1
Primul program Java.....	2
Pregătirea instrumentelor software necesare.....	3
Crearea unui proiect nou, compilarea și rularea lui.....	3
Operații de intrare/ieșire la nivel de linii de caractere.....	7
Utilizarea mediului de dezvoltare Eclipse .....	9
Temă.....	9

Un program dezvoltat cu ajutorul tehnologiei obiectuale are drept unitate de construcție nu subprogramul, ci obiectul. Un obiect înglobează date și operații și reprezintă o abstracțiune a unei entități din lumea reală. Obiectele componente interacționează, determinând transformarea datelor de intrare în date de ieșire, adică rezolvarea problemei. Într-un program dezvoltat în manieră obiectuală NU mai există date globale (sau, în orice caz, foarte puține), datele fiind repartizate și înglobate în obiecte. Metoda obiectuală de dezvoltare respectă principii ale ingineriei software precum:

- *localizarea și modularizarea*: codul sursă corespunzător unui obiect poate fi scris și actualizat independent de alte obiecte;
- *ascunderea informației*: un obiect are o interfață publică pe care celelalte obiecte o pot utiliza pentru comunicare. Pe lângă interfața publică, un obiect poate include date și operații private, „ascunse” față de alte obiecte, și care pot fi modificate oricând, fără a afecta restul obiectelor;
- *reutilizarea codului*: clasele de obiecte pot fi definite pe baza claselor existente, preluând automat (prin moștenire) conținutul acestora din urmă.

### Concepte fundamentale în programarea orientată pe obiecte

Principalele concepte care stau la baza programării orientate pe obiecte sunt:

- 1) **Abstractizarea** este una dintre căile fundamentale prin care oamenii ajung să înțeleagă și să cuprindă complexitatea. Ea scoate în evidență toate detaliile semnificative pentru perspectiva din care este analizat un obiect, suprimând sau estompând toate celelalte caracteristici ale obiectului. În contextul programării orientate pe obiecte, toate acestea se transpun astfel:
  - obiectele și nu algoritmi sunt blocurile logice fundamentale;
  - fiecare obiect este o instanță a unei clase. Clasa este o descriere a unei mulțimi de obiecte caracterizate prin structură și comportament similare;
  - clasele sunt legate între ele prin relații de moștenire.
- 2) **Încapsularea** este conceptul complementar abstractizării. Încapsularea este procesul de compartimentare a elementelor care formează structura și comportamentul unei abstracțiuni; încapsularea servește la separarea interfeței „contractuale” de implementarea acesteia. Din definiția de mai sus rezultă că un obiect este format din două părți distincte:
  - interfața (protocolul);
  - implementarea interfeței.

Abstractizarea definește interfața obiectului, iar încapsularea definește reprezentarea (structura) obiectului împreună cu implementarea interfeței. Separarea interfeței unui obiect de reprezentarea lui permite modificarea reprezentării fără a afecta în vreun fel pe nici unul dintre clienți, deoarece aceștia depind de interfață și nu de reprezentarea obiectului-server.

3) **Modularizarea** constă în divizarea programului într-un număr de subunități care pot fi compilate separat, dar care sunt cuplate (conectate) între ele. Gradul de cuplaj trebuie să fie mic, pentru ca modificările aduse unui modul să afecteze cât mai puține module. Pe de altă parte, clasele care compun un modul trebuie să aibă legături strânse între ele pentru a justifica gruparea (modul coeziv). Limbajele care suportă conceptul de modul fac distincția între:

- interfața modulului, formată din elementele (tipuri, variabile, funcții etc.) „vizibile” în celelalte module;
- implementarea sa, adică elemente vizibile doar în interiorul modulului respectiv.

4) **Ierarhizarea** reprezintă o ordonare a abstracțiunilor. Cele mai importante ierarhii în paradigma obiectuală sunt:

- ierarhia de clase (relație de tip *is a*);
- ierarhia de obiecte (relație de tip *part of*).

Moștenirea definește o relație între clase în care o clasă folosește structuri și comportamente definite în una sau mai multe clase (după caz vorbim de moștenire simplă sau multiplă). Semantic, moștenirea indică o relație de tip *is a* („este un/o”). Ea implică o ierarhie de tip generalizare/specializare, în care clasa derivată specializează structura și comportamentul mai general al clasei din care a fost derivată. Agregarea este relația între două obiecte în care unul dintre obiecte aparține celui alt obiect. Semantic, agregarea indică o relație de tip *part of* („parte din”).

## Primul program Java

```
class PrimulProgram{
    public static void main(String[] args) {
        System.out.print("Hello world!");
    }
}
```

Referindu-ne strict la structura acestui program, trebuie spus că programele Java sunt constituite ca seturi de clase (în cazul nostru avem o singură clasă). Pentru descrierea unei clase se folosește o construcție sintactică de forma:

```
class nume_clasa {
    //continut clasa
}
```

Aproape orice program, indiferent că este scris într-un limbaj procedural sau obiectual, are o rădăcină sau un punct de plecare. Astfel, în programele Pascal avem ceea ce se numește program principal, iar în C avem funcția *main()*. În programele *Java* vom avea o clasă rădăcină care se caracterizează prin faptul că include o funcție al cărei prototip este:

```
public static void main(String[] args)
```

Elementele desemnate prin cuvintele public și static vor fi lămurite mai târziu. Numele clasei rădăcină este un identificator dat de programator. Parametrul funcției *main()* este de tip tablou de elemente *String* (șiruri de caractere). Prin intermediul acestui parametru putem referi și utiliza în program eventualele argumente specificate la momentul lansării în execuție a programului. Acțiunile executate de programul de mai sus presupun două operații de afișare: *print()* și *println()*. Prefixul *System.out* care însoțește numele celor două operații reprezintă numele unui obiect: este vorba despre un obiect predefinit care se utilizează atunci când destinația unei operații de scriere este ecranul monitorului. Deoarece metodele sunt incluse în clase și, majoritatea, și în instanțele claselor, apelul presupune precizarea numelui clasei (dacă e vorba de metode statice) sau al obiectului „posesor”. Cu alte cuvinte, în Java, apelul unei operații se realizează folosind una din notațiile:

```
nume_obiect.nume_metoda(parametri_actuali)
```

sau:

```
nume_clasa.nume_metoda(parametri_actuali)
```

Într-un program Java vom lucra cu 2 tipuri de clase:

- clase definite de programator;
- clase predefinite, furnizate împreună cu mediul de dezvoltare Java, care formează așa numita *API* (*Application Programming Interface*), adică interfața pentru programarea aplicațiilor.

Revenind la primul nostru program Java, vom spune că metodele *print()* / *println()* pot primi ca parametru un șir de caractere dat, fie sub formă de constantă, așa ca în exemplul prezentat, fie sub formă de expresii al căror rezultat este de tip *String*. Metodele *print()* / *println()* mai pot primi ca parametru și valori ale tipurilor primitive sau referințe de clase, dar acestea sunt convertite tot la tipul *String* înainte de afișare. *Println()* poate fi apelată și fără parametri, caz în care execută doar un salt la linie nouă: *System.out.println()* ;

## Pregătirea instrumentelor software necesare

Înainte de editarea codului și de rularea primei aplicații este necesară instalarea instrumentelor software necesare. Acestea sunt:

- *JDK (Java Development Kit) JRE (Java Runtime Environment) for Windows:*  
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

- *Eclipse IDE for Java EE Developers:*  
[https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-12/R/eclipse-jee-2020-12-R-win32-x86\\_64.zip&mirror\\_id=1186](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-12/R/eclipse-jee-2020-12-R-win32-x86_64.zip&mirror_id=1186)

## Crearea unui proiect nou, compilarea și rularea lui

La prima rulare a aplicației Eclipse trebuie selectat spațiul de lucru (workspace), loc în care vor fi salvate toate proiectele aflate în lucru. După pornire va apărea fereastra „Welcome” care poate fi închisă. De asemenea, la prima rulare se alege ca aspectul mediului Eclipse să fie adecvat pentru realizarea aplicațiilor *Java SE* (*Java Standard Edition*) (vezi figura 1.1).

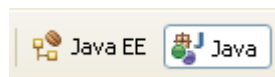


Fig. 1.1 – configurare *Eclipse Juno* pentru aplicații *Java SE*

Următorul pas presupune crearea unui nou proiect din meniul *File* (*File > New > Java Project*). În fereastra care se deschide (New Java Project – vezi figura 1.2) se atribuie un nume proiectului și se apasă comanda „Finish”.

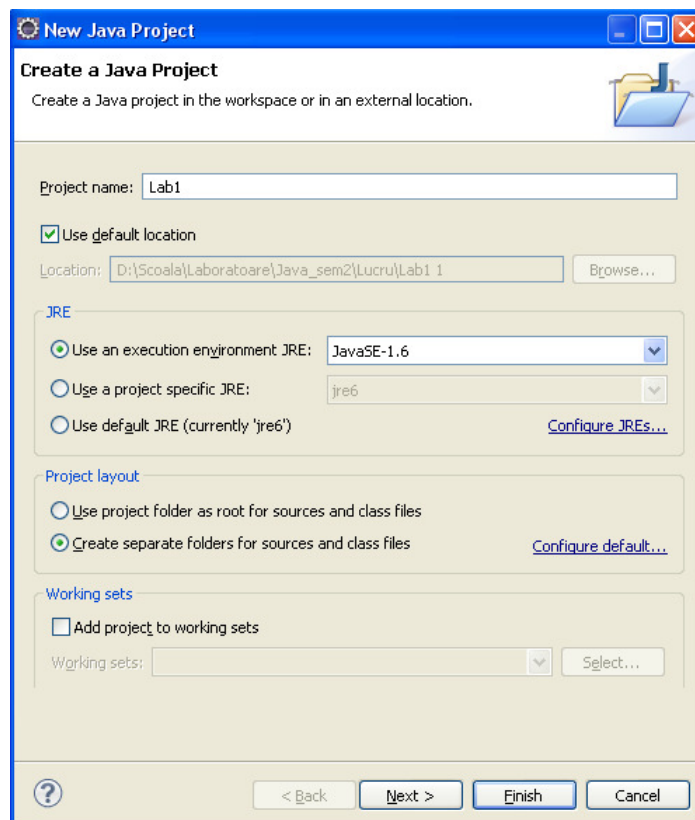


Fig 1.2 - Crearea unui nou proiect Java

În continuare proiectul creat poate fi vizualizat în fereastra „*Package Explorer*” (vezi figura 1.3).

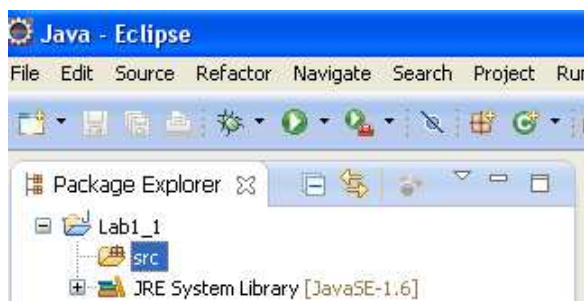


Fig 1.3 - Vizualizarea proiectului nou creat în fereastra *Package Explorer*

În următoarea etapă trebuie adăugat un pachet nou în proiectul creat (vezi figura 1.4).

Un pachet reprezintă un grup de tipuri de date (clase, interfețe, enumerări etc.) care au legătură între ele. Pachetele oferă protecție la acces și gestionarea spațiului de nume. Câteva dintre avantajele utilizării pachetelor sunt următoarele:

- Pot fi grupate anumite tipuri și astfel pot fi determinate cu ușurință care sunt tipurile care au legătură între ele.
- Sunt evitate conflictele dintre numele tipurilor aflate în pachete diferite (deoarece un pachet creează un spațiu de nume).

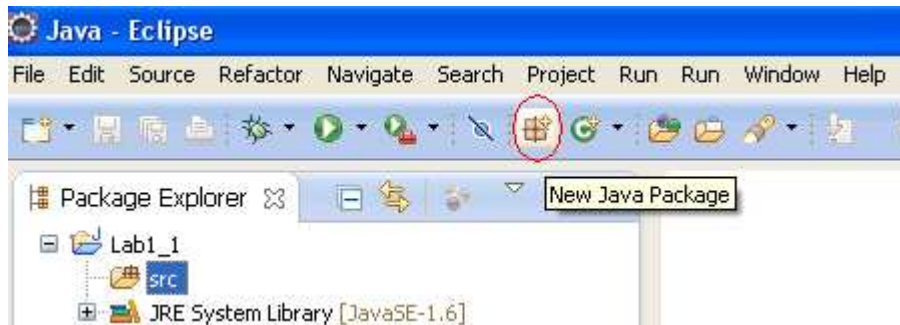


Fig 1.4- Adăugarea unui pachet nou

Se introduce un nume pachetului și se apasă comanda „Finish” (vezi fig 1.5).



Fig 1.5 – crearea unui pachet și a directorilor corespunzătoare

Următoarea etapă constă în crearea unei noi clase (vezi fig 1.6), în pachetul selectat.

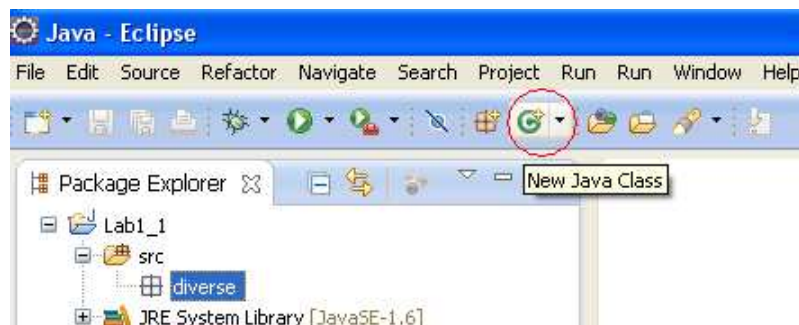


Fig 1.6 – lansarea comenzii de creare a unei noi clase în pachetul selectat

În fereastra pentru crearea unei noi clase se introduce numele acesteia (se recomandă introducerea unei denumiri care începe cu literă mare) și se menționează dacă clasa conține metoda *main* (vezi fig 1.7).

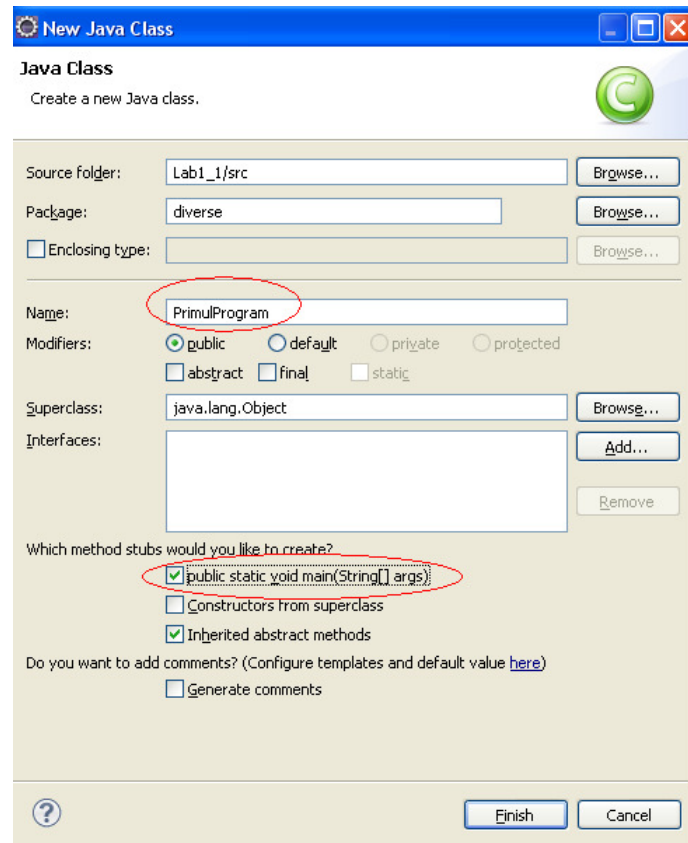


Fig 1.7 – crearea unei noi clase

Odată cu crearea clasei se va crea un fișier care conține numele clasei și extensia java (în cazul nostru *PrimulProgram.java*). Fișierele *java* sunt amplasate în următoarea cale:

director\_lucru > director\_proiect > src > director\_pachet > fisier.java

```

package diverse;

public class PrimulProgram {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.print("Hello World!");
    }

}

```

Fig 1.8 – Completarea codului clasa creată

Compilarea și executarea programului se face apăsând butonul *Run* (vezi figura 1.9) sau apăsând combinația de taste *Ctrl+F11*.

În urma compilării vor rezulta un număr de fișiere egal cu numărul claselor conținute de sursă. Fiecare dintre fișierele rezultate are extensia *.class* și numele identic cu numele

clasei căreia îi corespunde (în cazul nostru *PrimulProgram.class*). Fișierele class vor fi amplasate pe următoarea cale:

```
director_lucru > director_proiect > bin > director_pachet > fisiere.class
```

Un fișier .class conține cod mașină virtual (*Java Byte Code*). Acesta este codul mașină al unui calculator imaginar. Pentru a putea fi executat pe o mașină reală, este necesar un interpretor (sau executiv) care să execute fiecare instrucțiune a codului virtual în termenii operațiilor mașină ai calculatorului real. După compilare, programul în cod virtual obținut poate fi transportat pe orice mașină pentru care există executivul corespunzător, iar scrierea unui executiv este mult mai ușoară decât a unui compilator întreg.

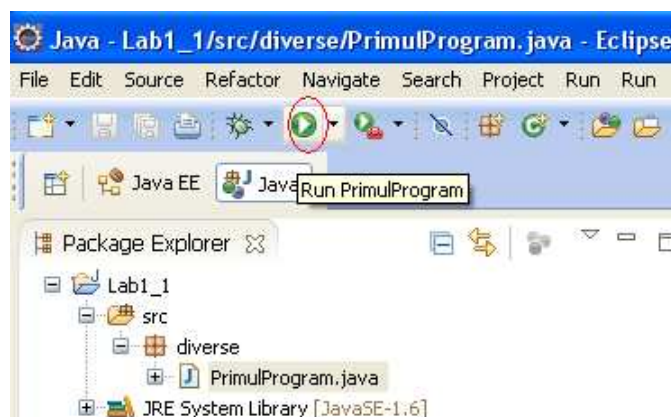


Fig 1.9 – rularea programului

Ieșirea programului este afișată în Consolă.

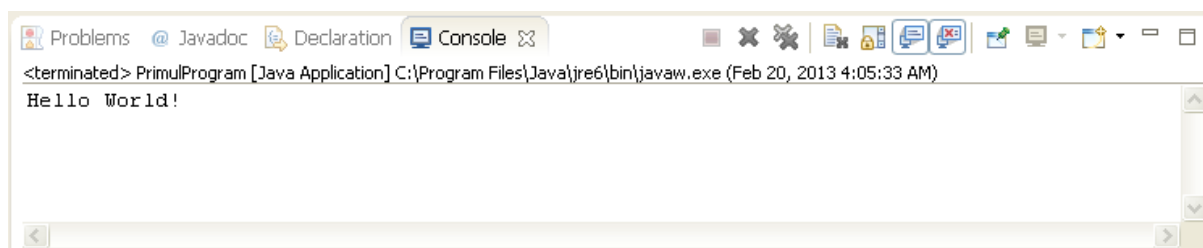


Fig 1.10 – execuția programului

## Operații de intrare/ieșire la nivel de linii de caractere

Pachetul *java.io* este un set de clase cu ajutorul cărora se realizează operațiile de intrare/ieșire într-un program Java. În *Java*, operațiile de intrare / ieșire se bazează pe conceptul de flux de date (*stream*). Un flux de date este o secvență de date care se deplasează dinspre o sursă externă spre memorie (flux de intrare — *input stream*) sau din memorie spre o destinație externă (flux de ieșire — *output stream*). Pentru operațiile de intrare / ieșire cea mai frecventă sursă externă este tastatura, iar destinația este ecranul monitorului. Acestea se mai numesc și suporturi standard de intrare, respectiv ieșire. Corespunzător suporturilor standard, în *Java* exista două obiecte predefinite: *System.in* pentru tastatură și *System.out* pentru monitor. O linie de caractere este o secvență terminată cu un caracter special, numit terminator de linie, care, la tastatură, are drept corespondent tasta *Enter*. Pentru a realiza citiri



la nivel de linie vor fi utilizate două subclase ale clasei *Reader* (*BufferedReader* și *InputStreamReader*). Pentru scrierea de linii, se va folosi clasa *PrintStream*.

### Operații de citire a liniilor de text

Clasa care modelează citirea unei linii dintr-un flux de intrare este *BufferedReader*, prin operația *readLine()*. Această operație nu are parametri, iar execuția ei are ca efect citirea din fluxul de intrare a unei secvențe de caractere până la întâlnirea terminatorului de linie. Operația returnează o referință la un obiect *String* care conține caracterele citite, dar fără a include și terminatorul. Cu alte cuvinte, șirul returnat conține doar caracterele utile (semnificative) ale liniei. Dacă s-a ajuns la sfârșitul fluxului de intrare, operația returnează valoarea *null*. Dacă citirea nu se poate desfășura, operația emite o excepție de tip *IOException*. De aceea, semnătura unei funcții care apelează metoda *readLine()*, dar nu tratează eventualele erori de citire, trebuie să conțină clauza *throws IOException*. Una dintre cele mai frecvente erori într-un program *Java* este omiterea clauzelor *throws* din antetul funcțiilor utilizatorului care apelează funcții predefinite dotate cu această clauză. Pentru a crea un obiect al clasei *BufferedReader* este necesar să i se furnizeze constructorului acesteia o referință la un obiect al clasei *InputStreamReader*. Constructorul acesteia din urmă necesită:

- referință la un obiect *FileInputStream*, dacă citirea se face dintr-un fișier de pe disc
- referință *System.in*, dacă citirea se face de la tastatură.

Deci, dacă urmează o citire dintr-un fișier al cărui nume este dat de o variabilă *String* *nume\_fis*, va trebui să se creeze un obiect *BufferedReader* ca în una din cele două secvențe de cod de mai jos:

```
BufferedReader flux_in;  
flux_in = new BufferedReader(new InputStreamReader(new FileInputStream(nume_fis)));  
  
//SAU  
  
flux_in = new BufferedReader(new FileReader(nume_fis));
```

Dacă citirea se va face de la tastatură, obiectul *BufferedReader* se creează astfel:

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

În continuare, citirea se va realiza cu apelul:

```
linie = flux_in.readLine();
```

unde *linie* este o referință *String*. În urma apelului, ea va indica spre un obiect care conține caracterele citite. Când se ajunge la sfârșitul fișierului funcția *readLine()* returnează *null*. În funcție de natura datelor reprezentate de aceste caractere, uneori pot fi necesare conversii de la *String* la alte tipuri, în vederea utilizării datelor respective în diverse calcule. Toate clasele înfășurătoare, asociate tipurilor primitive conțin metode de conversie. Spre exemplu clasa înfășurătoare a tipului primitiv *int* se numește *Integer* și conține metoda statică *parseInt(String)* care realizează conversie de la *String* la tipul primitiv asociat:

```
int x=Integer.parseInt("123");
```



## Operații de citire a tipurilor primitive

Utilizarea obiectelor `BufferedReader` la citirea întregilor, numerelor reale, etc, presupune citirea unui șir de caractere și apoi conversia lui în tipul primitiv dorit. Citirea directă a tipurilor primitive se poate realiza cu ajutorul obiectelor `Scanner` în felul următor:

```
Scanner scanner=new Scanner(System.in);
int x = scanner.nextInt();
```

Pentru citire din fișier instanțierea se face în felul următor:

```
Scanner scanner=new Scanner(new File("in.txt"));
```

Se poate testa dacă mai sunt elemente de citit din fișier cu ajutorul funcției `hasNext()`.

## Operații de scriere a liniilor de text

Afișarea unei linii pe ecran se realizează apelând metodele `print()` / `println()` definite în clasa `PrintStream`, pentru obiectul `System.out`. Pentru a scrie o linie într-un fișier de pe disc, se vor folosi aceleași metode, dar va trebui să se creeze un obiect separat al clasei `PrintStream`. Pentru aceasta, trebuie furnizată constructorului clasei `PrintStream`, ca parametru, un `String` care reprezintă numele fișierului, ca în secvența de mai jos:

```
PrintStream flux_out = new PrintStream (nume_fis);
```

unde `nume_fis` este o referință la un obiect `String` ce conține numele fișierului.

## Utilizarea mediului de dezvoltare Eclipse

### Facilitatea de autocompletare

Eclipse dispune de o facilitat de autocompletare a denumirilor de obiecte, clase, clase abstracte, interfețe, enumerări, etc. Pentru a utiliza această facilitat se completează primele litere ale numelui și apoi se apasă `Ctrl+Space`. În situația în care există o singură entitate a cărei denumire începe cu literele completate, Eclipse va completa automat denumirea acesteia. În cazul în care există mai multe entități a căror denumire începe cu literele tastate se va deschide o listă cu acestea din care trebuie aleasă cea dorită. Autocompletarea ajută la editarea rapidă și corectă a codului.

*Exemplu de utilizare:* Să presupunem că doriți să declarați un obiect de tip `BufferedReader`. Scrieți `Buf` apoi apăsați `Ctrl+Space` și din lista selectați clasa dorită.

### Indentarea codului sursă

Pentru a crește lizibilitatea codului, acesta trebuie scris indentat. Liniile de cod ale unei funcții se scriu cu un tab spre interiorul funcției. Dacă codul respectiv conține o instrucțiune repetitivă (`for`, `while`, `do...while`) atunci codul care aparține de acea instrucțiune repetitivă se pune la un tab de începutul ei, etc. Eclipse dispune de o facilitat de

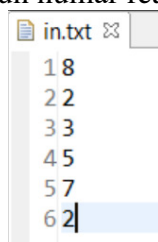
autoindentare a codului sursă, Ctrl+A – selectează codul sursă și Ctrl+I – autoindentează codul sursă.

## Depanarea programelor

Prima acțiune care trebuie realizată atunci când se dorește depanarea unui program este modificarea perspectivei din Eclipse. Se alege opțiunea de meniu *Window > Open perspective > Debug* sau se lansează aceeași comandă prin acționarea butonului *Open Perspective* de pe bara de unelte din colțul dreapta sus. În continuare se pune un *Break Point* pe linia pe care se dorește întreruperea execuției programului. Break pointul se introduce cel mai rapid prin dublu click în partea stângă a cifrei care indică numărul liniei de cod. Pentru ca programul să ruleze până la *break point* trebuie lansată comanda *Debug* (opțiunea de meniu *Run > Debug*) sau și mai rapid se apasă tasta *F11*). Programul va rula până la break point și în continuare poate fi rulat pas cu pas urmărind evoluția variabilelor în memorie. Se poate utiliza comanda *F5 (Step into)* ca să intre în metodele apelate și să permită executarea acestora pas cu pas, sau *F6 (Step over)* ca să execute metodele apelate cu totul fără a intra în ele. Valorile tuturor variabilelor active în zona de cod care se depanează este afișat în secțiune *Variables*, a perspectivei *Debug*. După ce depanarea programului s-a încheiat se revine la perspectiva adecvată dezvoltării de aplicații *Java Standard Edition*.

## Temă

1. Se cere să se scrie un program Java care să calculeze și să afișeze perimetru și aria unui dreptunghi. Valorile pentru lungime și lățime se citesc de la tastatură. Să se adauge un *break point* pe prima linie care citește valoarea unei laturi și din acel punct să se ruleze programul linie cu linie urmărind valorile variabilelor în memorie.
2. Să se scrie un program care citește un set de numerele din fișierul de intrare *in.txt*, care are conținutul din figura 1.11. Programul va determina suma lor, media aritmetică, valoarea minimă, valoarea maximă, va afișa aceste valori pe ecran și le va scrie în fișierul de ieșire *out.txt*. Media aritmetică va fi afișată ca un număr real.



```

18
22
33
45
57
62

```

Fig. 1.11 –fișierul *in.txt*

3. Să se scrie un program care citește un număr *n* natural de la tastatură și afișează toți divizorii acestuia pe ecran. Dacă numărul este prim se va afișa un mesaj corespunzător.
4. Să se determine *cmmdc* a două numere naturale, a căror valoare maximă este 30. Numerele vor fi generate aleatoriu cu ajutorul unui obiect de tip *Random* și metodei *nextInt()*;
5. Să se scrie un program care generează aleatoriu un număr întreg cuprins între 0 și 20. Programul va determina dacă numărul aparține șirului lui *Fibonacci*.