Laborator 9: Fluxuri de input/output și serializare

1 Responsabil

Gabriel Guțu-Robu, gabriel.gutu@upb.ro
Publicat: 12 decembrie 2022, 05:57

2 Clasa String

Un string este o *secvență* de caractere. În multe limbaje, string-urile sunt tratate ca un șir (sau vector) de caractere, însă în Java un string este un obiect (creat din clasa `String`). Clasa `String` are `11` constructori și peste `40` de metode. Pentru a crea un obiect de tipul `String`, se poate folosi oricare dintre următoarele instrucțiuni:

```
String s = new String("Buna ziua");
String s = "Bună ziua";
char[] tablouChar = {'B', 'u', 'n', 'a'};
String s = new String(tablouChar);
```

---

*Un obiect `String` este **immutable**, adică are conținut nemodificabil!.*

Următoarea secvență de cod **NU** modifică valoarea string-ului.

```
String s = "Buna";
s = "ziua";
```

Prima instrucțiune creează un obiect de tipul `String` cu conținutul `"Buna"` și asignează referința la `s`. A doua instrucțiune creează **un nou obiect String** cu conținutul `"ziua"` și asignează referința la `s`. Primul obiect de tipul `String` va exista în continuare, dar conținutul său nu va putea fi accesat deoarece acum variabila `s` trimite către un nou obiect.

```
String s1 = "Buna";
String s2 = new String("ziua");
String s3 = "Buna";
System.out.println((s1==s2));
System.out.println((s1==s3));
```

Va afișa:

```
False
True
```

În exemplul de mai sus, `s1` și `s2` sunt obiecte diferite, pe când `s1` și `s3` sunt referințe către același conținut (și, de fapt, același obiect), și anume `"Buna"`.

Operatorul ==, aplicat string-urilor, verifică dacă cele 2 string-uri se referă **la același obiect**. El nu returnează nicio valoare referitoare la conținutul lor! Pentru verificarea conținutului se pot folosi metodele `equals()` – care returnează `true` sau `false` sau `compareTo()`, care returnează `0` dacă cele două string-uri sunt identice sau o valoare negativă sau pozitivă, calculată în funcție de diferența dintre codurile caracterelor. În cazul în care nu se ține cont de litere mari sau mici, se pot folosi metodele `equalsIgnoreCase()` sau `compareToIgnoreCase()`. Metoda `match()` este asemănătoare metodei `equals()`, cu diferența că poate identifica o **expresie regulată** ca subșir al unui șir. Toate exemplele de mai jos returnează `true`. Expresia `.*` se referă la orice succesiune de `0` sau mai multe caractere ce urmeaza șirului `"Buna"`.

```
"Buna ziua".matches("Buna.*")
"Buna ziua tuturor ".matches("Buna.*")
"Buna".matches("Buna")
```

Pentru a obține lungimea unui string se poate folosi metoda `length()`, iar pentru a returna un caracter de pe o anumită poziție, metoda `charAt(int index)`, unde index trebuie să fie cuprins între `0` și `s.length()-1`. Un obiect din clasa `String` este reprezentat intern (înăuntrul clasei `String`) sub forma unui tablou. Metoda `concat()` concatenează două string-uri:

```
String s3 = s1.concat(s2);
```

Sau, echivalent:

```
String s3 = s1 + s2;
```

Pentru a împărți un șir în mai multe subșiruri separate de delimitatori, se folosește metoda `split()`:

```
String[] s = "-Buna ziua, ce mai faci?".split("[-,?]");
for (int i = 0; i < s.length; i++)
      System.out.println(s[i]);
```

Pentru a găsi un caracter sau un subșir într-un șir, se utilizează metodele `indexOf` sau `lastIndexOf`:

```
"Buna ziua".indexOf('B') va returna 0
"Buna ziua".lastIndexOf('a') va returna 8
```

Pentru a converti un string într-un tablou de caractere se folosește metoda `toCharArray()`.

```
char[] tablou = "Buna".toCharArray();
```

Iar în sens invers:

```
String str = new String(new char[]{'B', 'u', 'n', 'a'});
```

Sau

```
String str = String.valueOf(new char[]{'B', 'u', 'n', 'a'});
```

Metoda `String.valueOf(val_numerica)` creează un string dintr-un tip primitiv[1] – `char, char[], double, float, long, boolean.`

`String.valueOf(1.23)` va construi string-ul `"1.23"`

Pentru a converti un string într-o valoare numerică `double` sau `int`, se utilizează `Double.parseDouble(str)` sau `Integer.parseInt(str).`

Pentru a obține un substring dintr-un string, se folosește metoda `substring(int indexInceput)`, ce returnează un subșir ce începe de la poziția `indexInceput` și se încheie pe ultima poziție a șirului inițial. Metoda `substring(int indexInceput, int indexSfarsit)` returnează subșirul, începând cu poziția `indexInceput` și terminând cu `indexSfarsit-1`. Alte metode des întâlnite în practică sunt:

- `String toLowerCase()` – returnează un nou string în care caracterele sunt convertite la litere mici;
- `String toUpperCase()` - returnează un nou string în care caracterele sunt convertite la majuscule;
- `String trim()` - returnează un nou string în care sunt eliminate spațiile de la ambele capete;
- `String replace (char c1, char c2)` - returnează un nou string în care toate aparițiile caracterului `c1` sunt înlocuite cu `c2`;
- `String replaceFirst (String s1, String s2)` - returnează un nou string în care este înlocuită prima apariție a lui `s1` cu `s2`;
- `String replaceAll (String s1, String s2)` - returnează un nou string în care sunt înlocuite toate aparițiile lui `s1` cu `s2`.

3 Clasa Character

Ca pentru orice tip primitiv, și pentru tipul `char` există clasa corespunzătoare `Character`, aflată în pachetul `java.lang` (importat implicit). Pentru a crea un obiect de tipul `Character` se folosește instrucțiunea:

```
Character ch = new Character('a');
```

Ca și în cazul string-urilor, caracterele pot fi comparate cu ajutorul metodelor `equals` și `compareTo()`. Alte metode de lucru cu caracterele sunt:

- `char charValue()` – returnează valoarea `char` a obiectului;
- `boolean isDigit(char c)` – returnează `true` dacă caracterul este cifră;
- `boolean isLetter(char c)` – returnează `true` dacă caracterul este literă;

[1] https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

- `boolean isLetterOrDigit(char c)` – returnează `true` dacă caracterul este literă sau cifră;
- `boolean isLowerCase(char c)` – returnează `true` dacă caracterul este literă mică;
- `boolean isUpperCase(char c)` – returnează `true` dacă caracterul este literă mare;
- `char toLowerCase(char c)` – transformă caracterul în literă mică;
- `char toUpperCase(char c)` – transformă caracterul în literă mare.

4 Clasele StringBuilder și StringBuffer

Clasele `StringBuilder` și `StringBuffer` sunt o alternativă la clasa `String`. Ele oferă mai multă flexibilitate decât clasa `String`, întrucât conținutul lor poate fi modificat.

Metodele clasei `StringBuffer` sunt *sincronizate*, ceea ce înseamnă că accesul la un astfel de obiect se poate face concurent, din mai multe task-uri. `StringBuilder` este *mai eficient* și poate fi accesat doar de un singur task. Constructorii și metodele ambelor clase sunt identice. Constructorii clasei `StringBuilder` sunt:

- `StringBuilder()` – construiește un `StringBuilder` cu capacitatea de `16`;
- `StringBuilder (int capacitate)` – construiește un `StringBuilder` de capacitate dată;
- `StringBuilder (String s)` – construiește un `StringBuilder` pe baza lui `s`.

Într-un `StringBuilder/StringBuffer` se poate modifica conținutul, adăuga conținut la sfârșit, insera conținut la o anumită poziție, șterge sau înlocui caractere.

```
StringBuilder sb = new StringBuilder();
sb.append("Buna ziua");
sb.delete(2,5); //șterge caracterele de la index 2 la index 5
```

Aceste metode modifică conținutul `StringBuilder`-ului și returnează referința acestuia.

Instrucțiunea `StringBuilder sb1 = sb.reverse();` inversează conținutul lui `sb` și asignează referința lui `sb1`. Astfel, atât `sb`, cât și `sb1` pointează acum la același obiect.

În situația în care asupra obiectului nu trebuie să faceți modificări, este de preferat folosirea lui `String` în loc de `StringBuilder`, întrucât primul are o reprezentare mai optimizată. Pentru a obține un `String` dintr-un `StringBuilder`, se folosește metoda `toString()`.

5 Clasa File

Clasa `File` are metode ce permit obținerea proprietăților fișierelor, redenumirea și ștergerea lor. Această clasă însă nu are metode pentru citire și scriere din și în fișiere.

Crearea unei instanțe de tipul `File` se face cu apelul `new File("c:\\data.txt")`, unde `c:\\data.txt` este calea absolută către fișier. Se poate da și calea relativă. Crearea unei instanțe de tipul `File` nu conduce la crearea unui fișier. Pentru a verifica existența unui fișier, se apelează metoda `exists()`.

```
public class Test {

	public static void main(String[] args) {
		java.io.File f = new java.io.File("data.txt");
		System.out.println(f.exists());
		System.out.println(f.length());
		System.out.println(f.canRead());
		System.out.println(f.canWrite());
		System.out.println(f.isDirectory());
		System.out.println(f.isFile());
		System.out.println(f.isAbsolute());
		System.out.println(f.isHidden());
		System.out.println(f.getAbsolutePath());
	}
}
```

Scrierea într-un fișier se poate face folosind metodele din clasa `PrintWriter`, pentru citire se folosesc metodele din clasa `Scanner`. Pentru scriere, se creează un obiect de tipul `PrintWriter` și apoi se apelează metodele `print`, `println` sau `printf`. Dacă fișierul nu există, acesta va fi creat, iar dacă există, va fi suprascris.

```
PrintWriter pw = new PrintWriter(numeFișier);
```

Un exemplu complet de scriere în fișier este următorul:

```
import java.io.*;

public class Test {
	public static void main(String[] args) throws Exception{
		File f = new File("data.txt");
		if (f.exists()) {
			System.out.println("Fișierul există deja");
			System.exit(0);
		}
		PrintWriter pw = new PrintWriter(f);
		pw.print("Bună ziua");
		pw.println(2020);
		pw.close();
	}
}
```

Pentru a citi de la consolă, se creează un obiect `Scanner` cu parametru `System.in`, iar pentru a citi dintr-un fișier, se creează un obiect `Scanner` cu parametru un obiect de tipul `File`.

```
Scanner s = new Scanner(System.in);
```

```
Scanner s = new Scanner(new File(numeFișier));
```

Metodele din clasa `Scanner` sunt:

- `void close ()` – închide scanner-ul;
- `boolean hasNext()` – întoarce true dacă mai există date care trebuie citite;
- `String next()` – întoarce următorul string;
- `String nextLine()` – întoarce o linie citită;
- `byte nextByte()` – întoarce următorul token ca un `byte` de la scanner;
- `short nextShort()` – întoarce următorul token ca un `short` de la scanner;
- `int nextInt()` – întoarce următorul token ca un `int` de la scanner;
- `long nextLong()` – întoarce următorul token ca un `long` de la scanner;
- `float nextFloat()` – întoarce următorul token ca un `float` de la scanner;
- `double nextDouble()` – întoarce următorul token ca un `double` de la scanner.

Acestea sunt metode ce citesc token-uri, separate de delimitatori. În mod implicit, delimitatorii sunt spațiile. Se poate folosi metoda `useDelimiter - (String regex)` pentru a seta noi delimitatori. Metoda `next()` citește până la întâlnirea unui delimitator, iar `nextLine()` citește până la caracterul sfârșit de linie, care este `\r\n` în Windows și `\n` în Unix. Separatorul nu este parte a string-ului returnat de `nextLine()`.

Să presupunem că citim de la tastatură valoarea `10`, introducem Enter, apoi șirul "Ana", introducem Enter:

```
int x = input.nextInt();
String s = input.nextLine();
```

În acest caz, `x` va lua valoarea `10` și se va opri la delimitator, care este Enter. Metoda `nextLine()` se încheie după ce se citește separatorul și returnează valoarea de dinainte, în acest caz, nimic. Variabila s va fi goală.

```
import java.util.Scanner;
import java.io.*;

public class Test {
public static void main(String[] args) throws Exception {
            File f = new File("data.txt");
            Scanner s = new Scanner(f);
            while (s.hasNext()) {
                  String str = s.next();
                  float x= s.nextFloat();
                  int a = s.nextInt();
                  System.out.print(str + " " + x + " " + a);
            }
            s.close();
      }
}
```

6 Citirea fișierelor în Java

Fișierele text pot fi citite de un editor de text, în timp ce **fișierele binare** pot fi citite de către JVM. Avantajul fișierelor binare este că pot fi procesate mult mai eficient decât fișierele text. Informația din fișierele text este dispusă sub forma unei secvențe de caractere, iar în fișierele binare, sub forma unei secvențe de octeți, în hexazecimal. Clasele din Java se împart în clase de procesare **input/output pentru fișiere text** și **input/output pentru fișiere binare**.

Toate fișierele sunt stocate în format binar. Operațiile de intrare/ieșire text sunt bazate pe operațiile de intrare/ieșire binare, fiind caracterizate de un nivel înalt de abstractizare pentru codare și decodare. JVM convertește Unicode la o codare specifică fișierelor când scrie într-un fișier text și decodează în sens invers când citește dintr-un fișier text. Operațiile binare de intrare/ieșire nu necesită conversie. Ele sunt mai eficiente decât cele text fiindcă nu necesită codare și decodare. Fișierele binare sunt independente de schema de codare pe mașina gazdă și astfel sunt portabile. Programele Java pe orice mașină pot citi un fișier binar creat de un alt program Java. De aceea clasele Java sunt fișiere binare, astfel putând rula pe orice mașină unde este instalat un JVM.

6.1 Citirea fișierelor text

Tabelul 1 introduce cele mai uzuale clase pentru operații de intrare-ieșire efectuate asupra unor fișiere de tip text. Multiple exemple au fost deja introduse anterior.

Tabelul 1. Clase de intrare-ieșire pentru fișiere text.

Clasa	Extinde	Parametrii pentru constructori	Metode principale
`FileWriter`	`Writer`	`File` `String`	`close(), flush(), write()`
`BufferedWriter`	`Writer`	`Writer`	`close(), flush(), newLine(), write()`

`PrintWriter`	`Writer`	`File`
		`String`
		`OutputStream`
		`Writer`
			`close(), flush(), format(), printf(),`
			`print(), println(), write()`

`FileReader`	`Reader`	`File` `String`	`read()`
`BufferedReader`	`Reader`	`Reader`	`read()` `readLine()`
`Scanner`	`Reader`	`File` `String`	`hasNext(), next(), nextInt(), …`

6.2 Citirea fișierelor binare

`InputStream` și `OutputStream`, derivate din clasa `Object`, se află la baza claselor ce efectuează operații de intrare-ieșire cu fișiere binare. Figura 1 prezintă ierarhia de calse derivate din clasele `InputStream` și `OutputStream`.

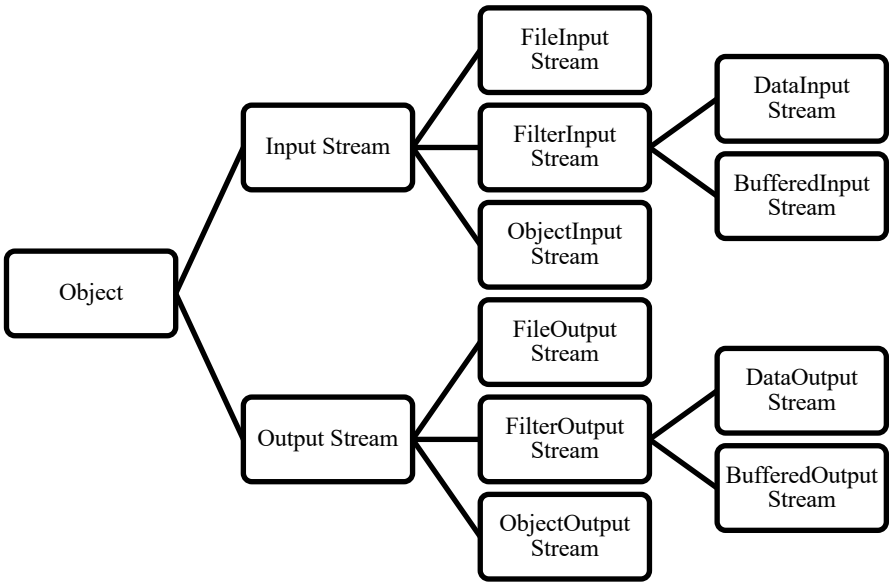

Figura 1. Clasele de intrare-ieșire.

FileInputStream și FileOutputStream

`FileInputStream` și `FileOutputStream` sunt folosite pentru citirea/scrierea octeților din și în fișiere. `FileInputStream` are doi constructori, iar `FileOutputStream` are patru constructori.

Constructorii clasei `FileInputStream` sunt:

- `void FileInputStream(File f)` – primește ca parametru un obiect de tipul `File`;
- `void FileInputStream(String numeFisier)` – primește ca parametru numele unui fișier.

Dacă se încearcă deschiderea unui fișier care nu există va apărea excepția `FileNotFoundException`.

Constructorii clasei `FileOutputStream` sunt:

```
void FileOutputStream(File f);
void FileOutputStream(String numeFisier);
void FileOutputStream(File file, boolean adauga);
void FileOutputStream(String numeFisier, boolean adauga).
```

Pentru ultimii doi constructori, dacă valoarea lui `adauga` este `true`, atunci se poate adăuga informație la finalul fișierului. Dacă fișierul nu există, atunci el este creat. Dacă există, primii doi constructori vor suprascrie vechiul conținut.

Metodele din clasele de intrare/ieșire aruncă excepția `IOException`. Aceasta trebuie să fie declarată în antetul metodei sau încadrată într-un bloc `try-catch`.

```
import java.io.*;

public class Test {

	public static void main(String[] args) throws IOException {
		FileOutputStream out = new FileOutputStream("test.dat");
		for (int i = 1; i <= 50; i++)
			out.write(i);
		out.close();
		FileInputStream in = new FileInputStream("test.dat");
		int x;
		while ((x = in.read()) != -1) // -1 semnifică finalul de
fișier
			System.out.print(x + " ");
		in.close();
	}
```

7 Interfața Serializable

Implementarea interfeței `java.io.Serializable` permite salvarea conținutului și stării obiectelor. Serializarea obiectelor se realizează folosind clasa `ObjectOutputStream`, iar deserializarea, folosind `ObjectInputStream`. Multe clase din Java au implementată interfața `Serializable`, cum ar fi clasa `java.util.Date`. Dacă se încearcă serializarea unei clase care nu are implementată această interfață, va rezulta excepția `NotSerializableException`.

Când un obiect serializat este stocat, este codată clasa acestuia – numele, semnătura, valorile instanțelor clasei. Valorile variabilelor statice nu sunt stocate. Dacă un obiect este o instanță a unei clase ce implementează `Serializable`, dar conține variabile membru neserializable, atunci acel obiect nu poate fi serializat. Se poate folosi cuvântul cheie *transient* pentru a ignora o variabilă membru la serializare, astfel încât să evităm apariția excepției `java.io.NotSerializableException`. Un tablou este serializabil dacă toate elementele sale sunt serializabile. Un întreg tablou poate fi salvat într-un fișier utilizând metoda `writeObject()` și citit utilizând metoda `readObject()`.

```
import java.io.Serializable;

public class Angajat implements Serializable {
	private String nume;

	public Angajat(String nume) {
		this.nume = nume;
	}
	public String toString() {
		return nume;
	}
}
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Arrays;

public class Companie implements Serializable {
	private Angajat[] angajati = new Angajat[3];

	public Companie() {
		angajati[0] = new Angajat("Mihai Ionescu");
		angajati[1] = new Angajat("Andrei Popescu");
		angajati[2] = new Angajat("Alina Negru");
	}

	public String toString() {
		return Arrays.deepToString(angajati);
	}
}


import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Test {
	public static void main(String[] args) {
		Companie c = new Companie();
		System.out.println("Initial: " + c);
		// Serializare
		ObjectOutputStream os = null;
		try {
			os = new ObjectOutputStream(new
FileOutputStream("out.bin"));
			os.writeObject(c);
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			if (os != null)
				try {
					os.close();
				} catch (IOException e) {
				}
		}
		// Deserializare.
		ObjectInputStream is = null;
		try {
			is = new ObjectInputStream(new
FileInputStream("out.bin"));
			c = (Companie) is.readObject();
			System.out.println("Deserializat: " + c);
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		} finally {
			if (is != null)
```

```
                        try {
                              is.close();
                        } catch (IOException e) {
                        }
            }
      }
}
```

Se va afișa:

```
Initial: [Mihai Ionescu, Andrei Popescu, Alina Negru]

Deserializat: [Mihai Ionescu, Andrei Popescu, Alina Negru]
```

8 Fișiere cu acces aleator

Clasa `RandomAccessFile` permite fișierelor să fie citite și scrise la locații aleatoare, modificate sau să se insereze noi înregistrări în fișiere.

```
RandomAccessFile file = new RandomAccessFile("test.dat", "rw");
```

Fișierul `test.dat` poate fi accesat atât pentru citire, cât și pentru scriere. Dacă fișierul nu există, el este creat.

Un fișier cu acces aleator poate fi văzut ca o secvență de octeți. Un indicator de poziție se găsește acolo unde se realizează citirea și scrierea. La deschiderea fișierului, indicatorul de poziție se află la început, apoi se deplasează, pe măsură ce se fac operații de citire și scriere. Apelul `file.seek(position)` mută indicatorul de poziție. Pentru a-l muta la începutul fișierului, apelăm `file.seek(0)`, iar pentru finalul fișierului, apelăm `file.seek(file.length())`. Pentru a șterge vechiul conținut, se utilizează metoda `file.setLength(0)` cu parametru `0`.

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class Test {
      public static void main(String[] args) throws IOException {
            RandomAccessFile f = new RandomAccessFile("test.dat", "rw");
            f.setLength(0);
            for (int i = 1; i <= 10; i++)
                  f.writeInt(i);
            f.seek(0);
            // se citeste primul int
            System.out.println(f.readInt());
            f.seek(4);
            // se citeste al doilea int
            System.out.println(f.readInt());
            f.seek(20);
            // se citeste al cincilea int
            System.out.println(f.readInt());
            f.seek(f.length());
            f.writeInt(20);
            // acum vom avea 11 elemente
            System.out.println("Noua lungime este "+f.length() );
            f.close();
      }
```

```
}
```

Se va afișa:

```
1
2
6
Noua lungime este 44
```

9 Aplicații

9.1 Parc Auto (text)

Definiți clasa `Auto` ce conține proprietățile:

- `private String model;`
- `private int an;`
- `private int km;`
- `private double pret;`

Și metodele:

- Un constructor cu parametri;
- Gettere și setter pentru cele 4 proprietăți;
- Suprascrierea metodei `public String toString()`, ce returnează un `String` reprezentând o concatenare a informațiilor despre autoturism, pe o singură linie;
- Metoda `public boolean identic(Auto a)`, ce returnează `true` dacă obiectul apelant este identic cu `a` dpdv al conținutului membrilor săi;
- Metoda `public static Auto citeste(BufferedReader br) throws IOException`, ce citește informațiile despre un obiect `Auto` din `BufferedReader br` și îl returnează funcției.

Definiți clasa `ParcAuto`, cu proprietatea:

- `private String numeFisier;`

Și metodele:

- Un constructor cu parametri;
- Metoda `public int numaraMasiniNoi() throws IOException`, ce numără câte mașini noi (cu kilometrajul egal cu 0) se găsesc în fișierul `numeFisier`;
- Metoda `public Auto celMaiScumpAuto() throws IOException`, ce returnează cel mai scump autoturism din fișierul `numeFisier`;
- Metoda `public void adaugaAuto(Auto a) throws IOException`, ce adaugă obiectul `a` de tipul `Auto` în fișierul `numeFisier`;
- Metoda `public boolean cauta(Auto a) throws IOException`, ce caută obiectul `a` de tipul `Auto` în fișierul `numeFisier` și returnează o valoare booleană. Folosiți apeluri ale metodei `identic`;

- Metoda `public void afiseazaParcAuto() throws IOException`, ce afișează pe ecran informațiile despre autoturismele salvate în `numeFisier`. Folosiți apeluri ale metodei statice `Auto.citeste`.

Creați un fișier numit `date.txt` în folder-ul proiectului vostru, în rădăcină, și salvați în el informațiile despre `4` obiecte de tipul `Auto`. Adăugați o linie goală după ultimul șir din fișier. Exemple fișier:

```
BMW
2019
5000
29000.0
Suzuki
2015
0
1900.0
Mercedes
2018
0
25000.0
Renault
2010
8000
7900.0
```

Instanțiați un obiect de tipul `ParcAuto`:

```
ParcAuto p = new ParcAuto("date.txt");
```

Apelați metodele definite anterior.

9.2 Parc Auto (binar)

Salvați starea obiectului de tip `ParcAuto` (și a obiectelor `Auto` "stocate" în el) într-un fișier binar. Realizați operațiile de scriere, iar apoi de citire din fișier. Verificați dacă citirea s-a putut realiza și comparați datele citite prin afișarea conținutului obiectelor (de exemplu, cu metoda `toString`).