11. Iznimke i tokovi (exceptions and streams)

Ovo poglavlje bavi se iznimkama tj. upravljanjem greškama te kako pisati i čitati s tipkovnice, datoteke, itd. U zadnjem dijelu dan je prikaz tehnike snimanja sadržaja objekata.

SADRŽAJ

---

1. Iznimke (Exceptions)

Java ima poseban mehanizam za upravljanje run-time pogreškama. Pretpostavite da pišete neki kod koji može uzrokovati pogrešku u tijeku izvršavanja programa. Npr. neka varijabla je trebala referirati na neki objekt, ali je u njoj vrijednost **`null`**. Ako preko takve reference pozovemo metodu objekta javit će se greška. Isto vrijedi i kad npr. pokušamo dijeljenje s nulom (cjelobrojne vrijednosti) ili pokušamo pristupiti elementu van granica niza. Bilo bi prekomplicirano svaki put provjeravati sadržaj varijabli. Stoga se upotrebljava druga tehnika. Program puštamo da se izvršava, a sustav u trenutku pogreške ***baca iznimku* (*throws an exception)*** koju trebamo obraditi. Iznimku možemo i sami generirati. Što se događa kad je iznimka bačena:

1. Kreira se objekt koji opisuje pogrešku. Takav objekt se obično naziva objekt iznimke exception object). (U stvari ovaj objekt pripada klasi Throwable, i može biti u subklasi Error ako se radi o ozbiljnoj sistemskoj grešci ili u subklasi Exception ako se radi o normalnoj run-time grešci.)

2. Interpreter zaustavlja izvršavanje tekuće naredbe i počinje tražiti catch blok koji je napisan da odgovori na točno taj tip greške. Ako interpreter ne može naći odgovarajući catch blok, program će se zaustaviti i bit će ispisana poruka o grešci u prozoru (DOS) konzole. U ispisu će biti naziv pogreške i popis svih metoda koji se trenutno izvršavaju. To ima nekog smisla za programera, ali korisnik programa ne zna što će s tim podacima. Program je pao !

`catch` blok je dio `try-catch` izraza koji ima slijedeću formu.

```
try                                 )
{  NAREDBE                          |    try blok
}                                   )
catch (EXCEPTION1 e1)               )
{  NAREDBE                          |
}                                   |
catch (EXCEPTION2 e2)               |    bilo koji broj catch blokova
{  NAREDBE                          |
}                                   |
:                                   )
finally                             )
{  NAREDBE                          |    opcionalni finally blok
}                                   )
```

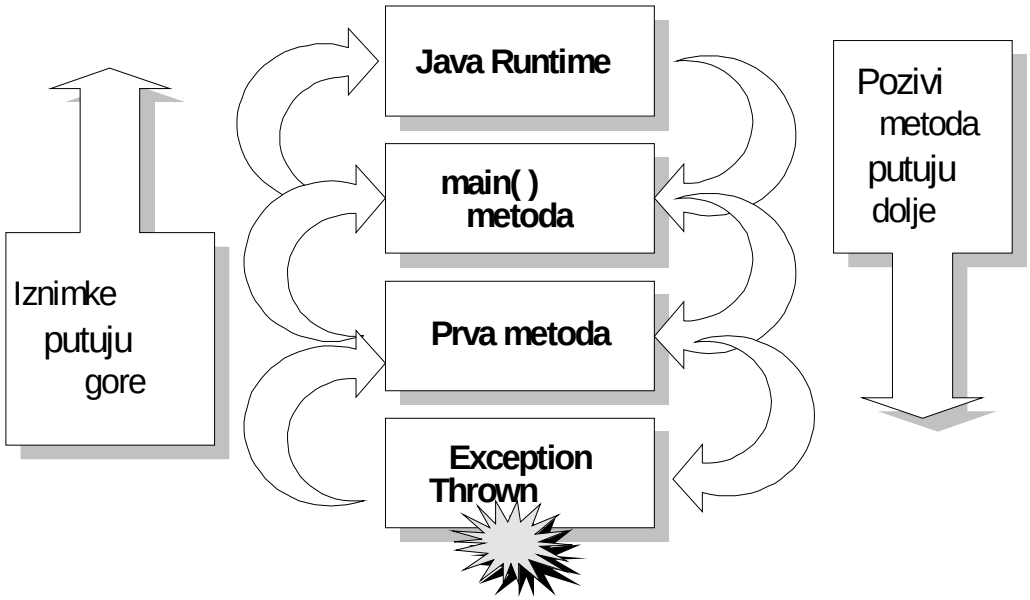
Zadnji dio tj. `finally` izraz se često izostavlja, a bitno za njega je da se uvijek izvršava i to nakon izvršavanja try-catch blokova.

Kada interpreter izvršava `try-catch` izraz, prvo počinje s izvršavanjem naredbi u try bloku. Naredbe se izvršavaju normalnim slijedom i ako se nešto ne baci iznimku neće se izvršiti nijedan od catch blokova. Ako je dodan blok finally bit će izvršen nakon try bloka.

Ako se dogodi bacanje iznimke u try bloku interpreter će potražiti redom po svim catch blokovima da li koji od njih kao argument ima upravo generirani tip objekt iznimke. Ako nađe takav bit će izvršen. Nakon njega opet finally blok. Kako je logika izvršavanja finally bloka jednostavna i jasna odsad ćemo zanemariti mogućnost njegove upotrebe.

U ovome slučaju kažemo da je iznimka ulovljena (has been caught). Ako se iznimka ne ulovi ona se prosljeđuje pozivnoj metodi itd. sve dok se eventualno ne nađemo unutar nekog try bloka.

Slijedeća slika pokazuje smjer prosljeđivanja objekta iznimke:



Slijedi program koji sadržava `try-catch` izraz. To je nova verzija programa koji smo već prije vidjeli. Program učitava niz brojeva u pokretnom zarezu i prestaje s učitavanjem kada naiđe na riječ "kraj". Program zbraja brojeve i u isto vrijeme broji koliko je brojeva uneseno. Na kraju program izračunava prosjek koji se prikazuje na ekranu.

Program provjerava da li se u unesenoj liniji nalazi riječ "kraj". Za čitanje koristi metodu `readLine`. Ako uneseni string nije riječ 'kraj', program pokušava konvertirati string u broj koristeći metodu `Double.parseDouble`. Ako greškom unesete nešto što nije broj (ili riječ "kraj") bit će generirana iznimka NumberFormatException. Ako iznimku ne uhvatimo, program će se zaustaviti i bit će prikazana poruka o grešci.

U ovoj verziji programa umetnut je izraz `try-catch` s ciljem da se iznimka uhvati i izbjegne prekid programa. `try` blok sadržava poziv metode `Double.parseDouble`, i akciju koja slijedi ako je unesen predviđeni string. `catch` blok ispisuje jednostavnu poruku o grešci i program se nastavlja izvršavati.

PRIMJER 1

```
public class Prosjek1

{  /* Učitaj brojeve u pokretnom zarezu
      i ispiši njihov prosjek.
      (Verzija koja koristi ConsoleReader.)
   */
   public static void main(String[] args)
   {  ConsoleReader user =
         new ConsoleReader(System.in);

      /*Čitaj i dodaj vrijednosti,
         te broji ukupan broj vrijednosti. */
      double suma = 0;
      int koliko = 0;
      System.out.println("Unesi podatke.");
      while (true)
      {  String line = user.readLine();
```

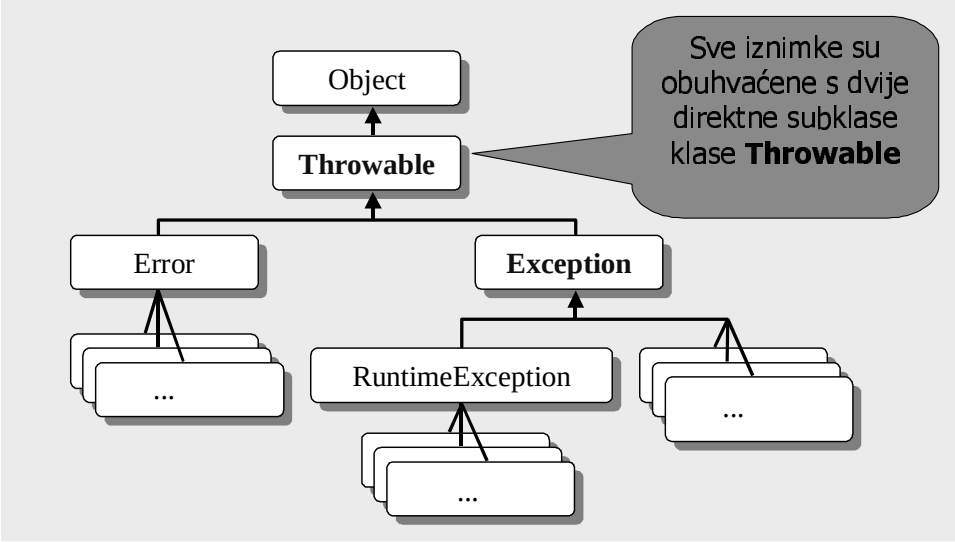
```
            if (line.equals("kraj")) break;
            try
            {  double next = Double.parseDouble(line);
               suma = suma + next;
               koliko++;
            }
            catch (NumberFormatException e)
            {  System.out.println
                  ("Nerazumljiv ulazni podatak.");
            }
         }

         /* Ispiši prosjek. */
         if (koliko > 0)
            System.out.println
               ("Srednja vrijednost = " + suma/koliko);
         else
            System.out.println("Nema unesenih vrijednosti.");
      }
   }
```

Osim iznimki koje generira Java, možete i sami baciti iznimku. Potrebno je upotrijebiti **throw** naredbu. Opći oblik throw naredbe je:

```
throw new NumberFormatException();
        -- referenca na objekt iznimke ----
```

Primijetite da je za kreiranje iznimke upotrijebljen konstruktor. U Java biblioteci postoje različiti tipovi iznimki.:



Error iznimke :

Predstavljaju iznimke koje nisu predviđene da ih hvata programer. Postoje tri direktne subklase Error iznimke.

ThreadDeath: bačena svaki put kad se namjerno zaustavi nit (thread). Ako se ne uhvati nit završava s izvođenjem(ne i program).

LinkageError : ozbiljna greška unutar klasa programa (nekompatibilnost klasa, pokušaj kreiranja objekta nepostojeće klase.
VirtualMachineError – JVM greška

RunTimeException

Subklase:

ArithmeticException: Greška u aritmetici. Npr. cjelobrojno dijeljenje nulom.

IndexOutOfBoundsException : indeks izvan granica objekta koji koristi indekse npr. array, string, i vector

NegativeArraySizeException : Korištenje negativnog broja za veličinu niza.
NullPointerException : pozivanje metode ili pristup polju objekta preko null reference

ArrayStoreException : pokušaj dodjeljivanja objekta neodgovarajućeg tipa elementu niza (Array)

ClassCastException: pokušaj kastiranja objekta u nepravilan tip

SecurityException : prekršaj sigurnosti (Security manager)

---

2. Čitanje s tipkovnice

Kada u Java programu čitamo podatke s nekog ulaznog medija onda koristimo objekt koji upravlja s ulazom. Objekt se spaja na izvor podataka, npr. tipkovnicu ili tekstualnu datoteku. Da bismo čitali podatke koristimo metode tog objekta. Objekti tipa ConsoleReader koji je korišten u primjerima za čitanje podataka s tipkovnice je tipičan primjer takvog objekta.

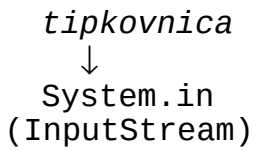
Klase koje upravljaju s ulazom nazivaju se **InputStream** i **Reader**. Razlikuju se u tome što objekti rade kada se čitaju znakovi.

Objekt tipa **InputStream** vraća 8-bitni oktet (byte) svaki put nakon čitanja podatka.

Objekt tipa **Reader** vraća 16-bitne vrijednosti. To je standardan način reprezentacije znakova u Javi, koji se naziva **Unicode.** Unicode skup znakova obuhvaća alfabete većine svjetskih jezika.

Svaki put kad smo dosad kreirali objekt tipa ConsoleReader posredno smo koristili objekt tipa System.in. To je objekt tipa InputStream spojen direktno na tipkovnicu. Klasa InputStream posjeduje više metoda, ali samo jednu za čitanje i to metodu **`read`** koja čita jedan oktet (byte)

ili unaprijed zadan niz byte-ova. Slijedeći dijagram pokazuje vezu tipkovnice i pripadne klase za čitanje podataka:

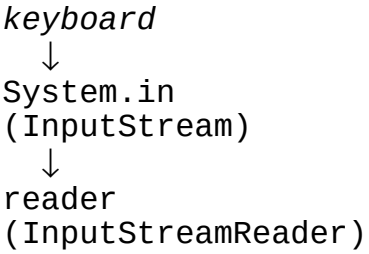


Java posjeduje i klasu **InputStreamReader** koja učitava podatke u Unicode formatu odnosno kao 16-bitne unicode znakove. Postoji konstruktor kojim je moguće pretvoriti InputStream u InputStreamReader. Referenca na objekt tipa InputStream je parametar konstruktora:

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

U konstruktor se može uključiti i drugi parametar za korištenje "nestandardnog" kodiranja 8-bitnog u Unicode prikaz. Inače će se obaviti standardna "default" konverzija u kodnu stranicu koja je trenutno u upotrebi na računalu.
Ovaj način konverzije jedne vrste ulaza u drugu preko konstruktora je tipična za Javu. Isto vrijedi i za klase izlaza.

Slijedi dijagram koji pokazuje povezanost klasa. Na kraju je rezultat 16-bitni karakter.



klasa InputStreamReader posjeduje metodu `read` koja vraća samo jedan znak. Pomoću ove metode moguće je napisati metode koje će čitati brojeve, riječi ,... Međutim bolja polazna točka bila bi klasa koja posjeduje metode za čitanje niza znakova odjednom. Postoje dvije vrste klasa koje to mogu učiniti:

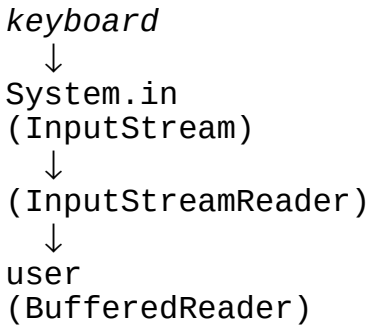
BufferedReader, **LineNumberReader**.

Obje klase posjeduju `readLine` metodu koja vraća uneseni niz znakova.

Moguće je konvertirati InputStreamReader u BufferedReader. Kao i obično to činimo pomoću konstruktora:

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

Ovaj izraz gradi kanal (‘pipeline’) koji izgleda ovako:



Objekt tipa BufferedReader pohranjuje podatke u **buffer.** To ubrzava unos podataka ako izvor podataka dozvoljava učitavanje odjednom cijelog niza znakova. (Konstruktoru se može proslijediti i veličina buffera. Inače će Java kreirati razumno velik buffer).

Slijedi još jedna verzija programa za računanje prosjeka. razlika je u tome da program ne koristi klasu ConsoleReader. Razlike među primjerima su podebljane.

PRIMJER 2

```
import java.io.*;

public class Prosjek2

{  /* Učitaj brojeve u pokretnom zarezu
      i ispiši njihov prosjek.
      (Verzija koja koristi BufferedReader.)
   */
   public static void main(String[] args)
      throws IOException
   {  BufferedReader user =
         new BufferedReader
           (new InputStreamReader(System.in));

      /*Čitaj i dodaj vrijednosti,
         te broji ukupan broj vrijednosti. */
      double suma = 0;
      int koliko = 0;
      System.out.println("Unesi podatke.");
      while (true)
      {  String line = user.readLine();
         if (line.equals("kraj")) break;
         try
         {  double next = Double.parseDouble(line);
            suma = suma + next;
            koliko++;
         }
         catch (NumberFormatException e)
         {  System.out.println
               ("Nerazumljiv ulazni podatak.");
         }
      }

      /* Ispiši prosjek. */
```

```
      if (koliko > 0)
         System.out.println
            ("Srednja vrijednost = " + suma/koliko);
      else
         System.out.println("Nema unesenih vrijednosti.");
   }
}
```

Primijetite promjenu u zaglavlju programa.

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

Ovo pokazuje prevodiocu da **`main`** metoda sadržava metodu , u ovom slučaju **`readLine`**, koja može baciti iznimku tipa **IOException** i koja neće biti uhvaćena jer u metodi **`main`** neće biti odgovarajućeg **`try-catch`** izraza da ga ulovi.

Iznimka IOException obuhvaća niz različitih grešaka koje se mogu javiti prilikom čitanja podataka i spada u grupu **checked** iznimki. Znači da je uvijek potrebno ili napisati izraz koji će je uhvatiti ili je potrebno dodati **`throws`** u zaglavlju metode:

```
throws IOException
```

Možemo dodati više tipova iznimki odvojenih zarezom.

Primijetite da kad smo koristili metode **`Double.parseDouble`** ili **`Integer.parseInt`**, nismo uključivali **`throws`** izraz za iznimku **NumberFormatException** koju bacaju navedene metode.

Razlog tome što **NumberFormatException** spada u **unchecked** iznimku. Nema potrebe da se uključuje **`throws`** izraz. Ideja je u tome da postoje iznimke koje se ne bi trebale pojavljivati ako je program dobro napisan.

Kada **`main`** metoda posjeduje **`throws`** izraz to je znak da navedena iznimka može terminirati program ostavljajući korisnika da gleda u ružnu pogrešku. To je normalno ako pišete eksperimentalni program za svoje potrebe, ali ne i za program za krajnjeg korisnika.
U tom slučaju sve iznimke je potrebno uhvatiti i obraditi.

Slijedi popravljeni program iz primjera 2. Dodan je drugi **`catch`**-blok za hvatanje iznimke **IOException** koju može baciti metoda **`readLine`**.

```
try
{  String line = user.readLine();
   if (line.equals("kraj")) break;
   double next = Double.parseDouble(line);
   suma = suma + next;
   koliko++;
}
catch (NumberFormatException e)
```

```
{  System.out.println
      ("Input not recognised.");
}
catch (IOException e)
{  System.out.println("Ulazna greška.");
   return;
}
```

Slijedi kompletna definicija **ConsoleReader** klase. U klasi se kreira **BufferedReader** kao što je to učinjeno u primjeru 2. Metoda `readLine` posjeduje kod za hvatanje bilo koje iznimke tipa IOExceptions. U odgovarajućem catch bloku nalazi se kod za izlaz iz programa.

U klasi nema nikakvog pokušaja hvatanje iznimki tipa **NumberFormatExceptions** koje mogu baciti `readInt` ili `readDouble`. One se šalju nazad u pozivnu metodu gdje ih korisnik može uhvatiti ako to želi (tip unchecked).

klasa ConsoleReader

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;

/**
   A class to read strings and numbers
   from an input stream.
   This class is suitable for
   beginning Java programmers.
   It constructs the necessary buffered
   reader, handles I/O exceptions, and
   converts strings to numbers.
*/

public class ConsoleReader

{  private BufferedReader reader;

   /** Constructs a console reader from
       an input stream such as System.in.
   */
   public ConsoleReader(InputStream inStream)
   {  reader =
       new BufferedReader
        (new InputStreamReader(inStream));
   }

   /** Read a line of input and
       convert it into an integer.
   */
   public int readInt()
   {  String inputString = readLine();
      int n = Integer.parseInt(inputString);
      return n;
   }
```

```
   /** Reads a line of input and convert it
       into a floating-point number.
   */
   public double readDouble()
   {  String inputString = readLine();
      double x =
         Double.parseDouble(inputString);
      return x;
   }

   /** Read a line of input.
       In the (unlikely) event of
       an IOException, the program halts.
   */
   public String readLine()
   {  String inputLine = "";
      try
      {  inputLine = reader.readLine();
      }
      catch(IOException e)
      {  System.out.println(e);
         System.exit(1);
      }
      return inputLine;
   }
}
```

---

3. Čitanje iz tekstualne datoteke

Za čitanje iz datoteke moguće je kreirati posebni tip **Reader** objekta , **`FileReader`** koji je spojen na datoteku.
Za kreiranje veze, u **FileReader** konstruktoru navodimo naziv datoteke kao parametar. Npr. ako je potrebno čitati podatke iz datoteke data.txt koristimo slijedeći izraz:

```
FileReader input = new FileReader(”data.txt”);
```

Ako datoteku nije moguće pronaći, Java baca **`FileNotFoundException`**.
Ova iznimka je **checked** iznimka pa je potrebno koristiti **throws** izraz za svaku metodu gdje se ne hvata.

Klasa **FileReader** poput bilo koje klase tipa **Reader,** posjeduje **read** metodu koje vraća slijedeći unicode znak u obliku **int** vrijednosti, ali ne posjeduje **readLine** metodu.
Da bismo dobili **readLine** metodu, konvertiramo **FileReader** u **BufferedReader** korištenjem iste konstrukcije kao u prethodnom poglavlju.

```
BufferedReader data =
   new BufferedReader
      (new FileReader(”data.txt”));
```

Jednom kad smo kreirali objekt tipa **BufferedReader** koristimo njegovu metodu **readLine** na isti način kao što smo to činili kad su podaci dolazili s tipkovnice.
Jedina razlika je što je potrebno provjeriti da li su pročitane sve linije iz datoteke, odnosno da li smo stigli do kraja datoteke. To je jednostavno jer **readLine** vraća **null** kada stigne do kraja datoteke.
Kada je čitanje iz datoteke završeno potrebno je pozvati metodu **close**.

Slijedi finalna verzija programa za računanje prosjeka. Ova verzija čita podatke iz datoteke numbers.dat.

Primjer 3

```
// Program koji čita floating-point vrijednosti
// iz tekstualne datoteke 'numbers.dat',
// i proračunava njihov prosjek.

import java.io.*;

public class Prosjek3

{  public static void main(String[] args)
   {  BufferedReader data;
      try
      {  data = new BufferedReader
                  (new FileReader("numbers.dat"));
      }
      catch (FileNotFoundException e)
      {  System.out.println
            ("Datoteka numbers.dat nije pronađena.");
         return;
      }

      /*čitaj i dodaj vrijednosti,
         te broji ukupan broj vrijednosti. */

      double suma = 0;
      int koliko = 0;
      try
      {  while (true)
         {  String line = data.readLine();
            if (line == null) break;
            try
            {  double next =
                  Double.parseDouble(line);
               suma = suma + next;
               koliko++;
            }
            catch (NumberFormatException e)
            {  System.out.println
                  ("Nerazumljiv ulazni podatak: " + line);
            }
         }
         data.close();
      }
      catch (IOException e)
      {  System.out.println(e);
```

```
            return;
         }

         /* Ispiši prosjek. */
         if (koliko > 0)
            System.out.println
               ("Srednja vrijednost = " + suma/koliko);
         else
            System.out.println("Nema unesenih vrijednosti.");
      }
   }
```

Primjedbe.

1. Ako ne postoji datoteka bit će uhvaćena iznimka FileNotFoundException i program će biti prekinut uz odgovarajući ispis o pogrešci.
2. Vanjski **`try-catch`** blok, koji sadrži **`while`**-petlju, hvata **IOExceptions** koje mogu baciti metode **`readLine`** i **`close`**.
3. Unutarnji **`try-catch`** blok unutar petlje, hvata iznimke tipa **NumberFormatExceptions** koje može baciti metoda `Double.parseDouble`.
4. Program prestaje s čitanjem je **`line`** postavljen na **null**.
5. Ako bilo koja linija sadržava niz znakova koji ne predstavljaju broj bit će bačena iznimka **NumberFormatException.** Program će u pripadnom catch bloku ispisati sadržaj te linije.

---

4. Pisanje u tekstualnu datoteku

Ako želite pisati u tekstualnu datoteku najbolje je koristiti **`PrintWriter`** klasu.
Ova klasa posjeduje **`println`** i **`print`** metode (poput `System.out` klase).
`PrintWriter` objekt ne može se direktno spojiti na datoteku, već preko `FileWriter` objekta, koji prihvaća unicode karaktere i piše ih u tekstualnu datoteku.

Slijedi izraz koji ilustrira navedeno:

```
PrintWriter out =
   new PrintWriter
     (new FileWriter("data.txt"))
```

U datoteku pišemo korištenjem metoda **`print`** i **`println.`** Nakon što smo završili s ispisom možemo koristiti poziv `out.flush()`. To forsira spremanje sadržaja međuspremnika u datoteku. Na kraju pozivom **`out.close()`** zatvaramo vezu.

Java posjeduje opsežnu biblioteku klasa za čitanje i pisanje. Npr. postoje klase za komprimirano pisanje (zip), klase za rad s XML dokumentima, itd.