Języki i metody programowania – Java
INF302W
Wykład 4

Autor

Dr inż. Zofia Kruczkiewicz

Struktura wykładu

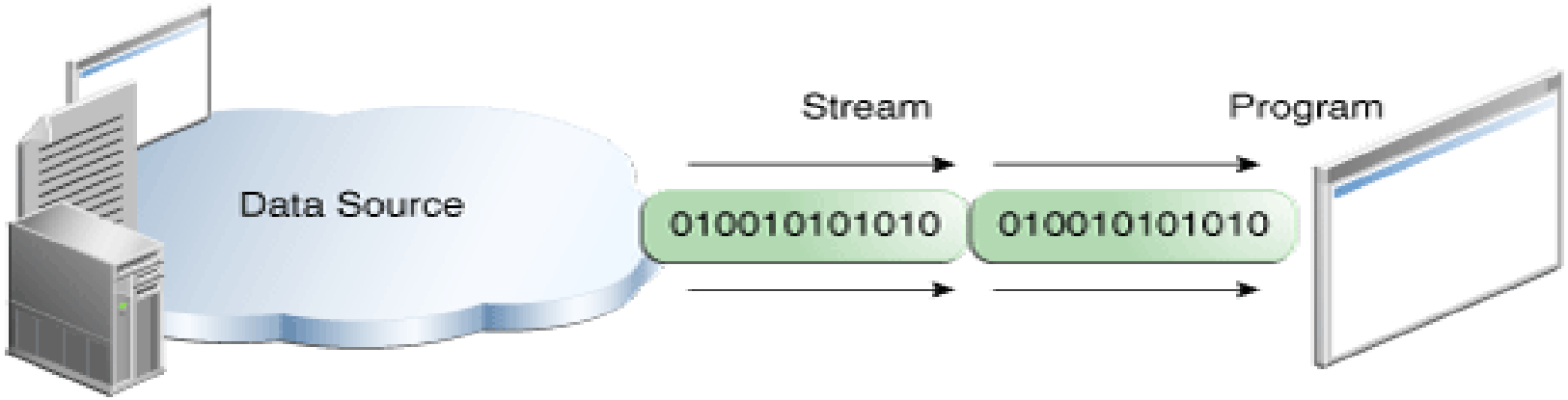
1. Strumienie binarne i tekstowe niebuforowane i buforowane, obsługa plików tekstowych, wprowadzanie danych z klawiatury (R-1, EL-1).
   https://docs.oracle.com/javase/tutorial/essential/io/index.html

2. Pakiet java.nio.file – wspieranie operacji we/wy (I/O)
   https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

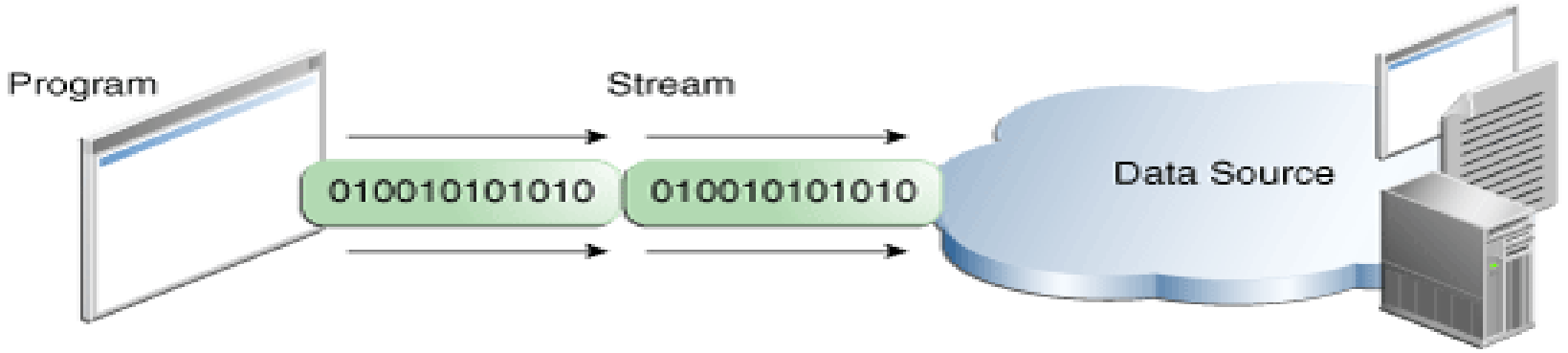
Strumienie wejścia/wyjścia (I/O)

https://docs.oracle.com/javase/tutorial/essential/index.html

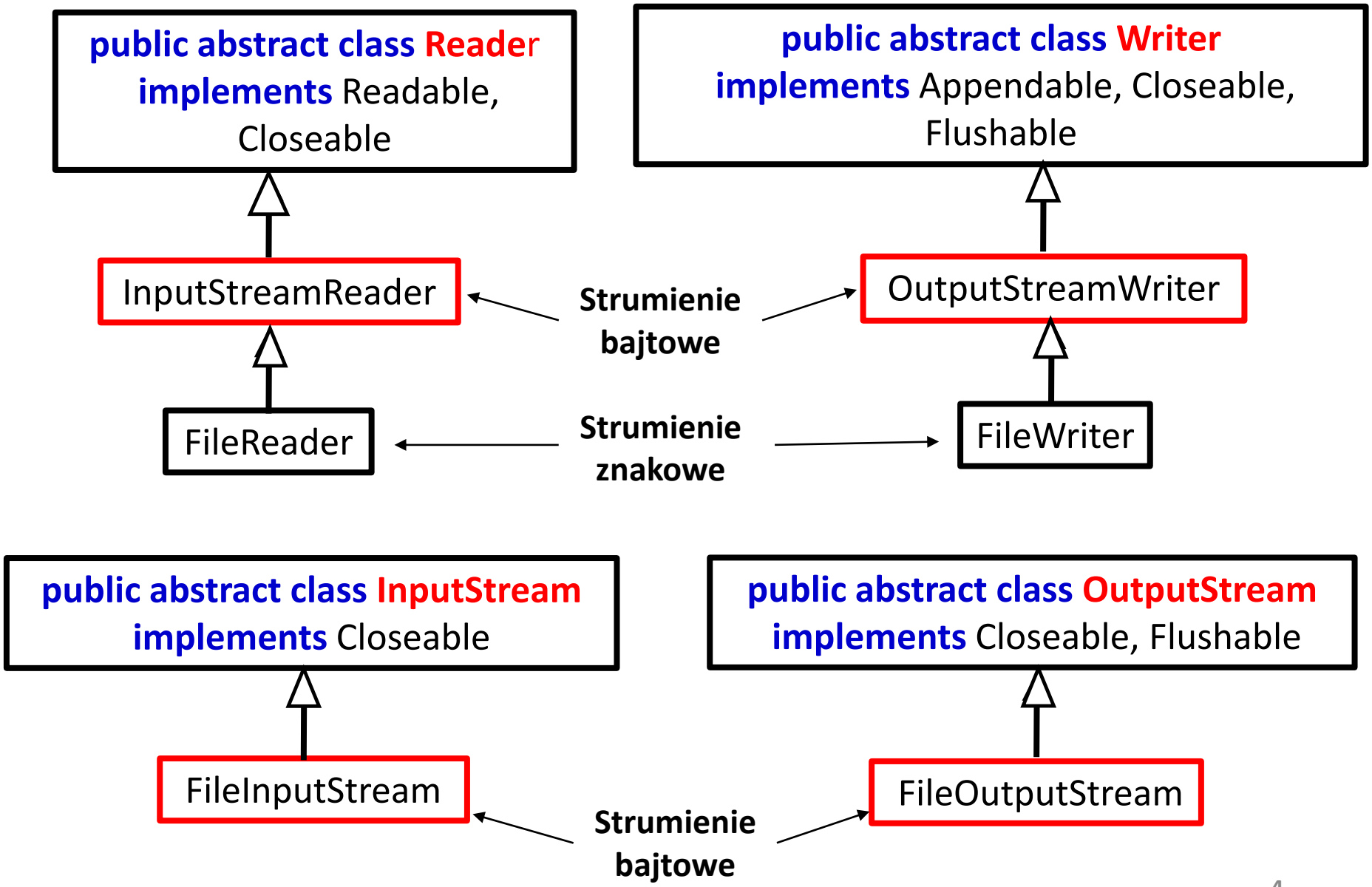
Odczyt danych ze źródła danych przez program



Zapis danych w źródle danych przez program

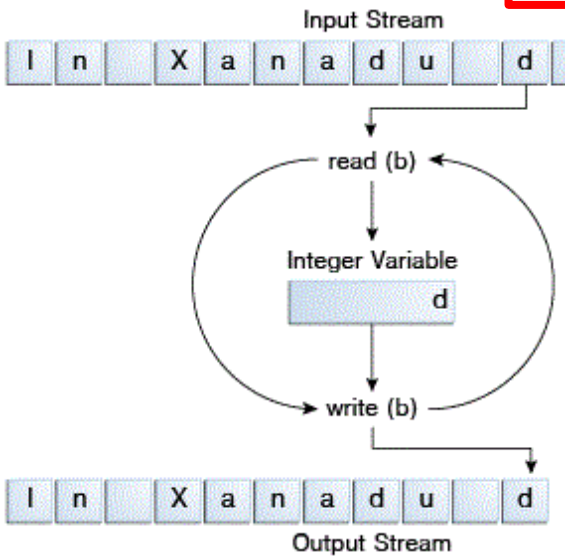


1. Strumienie wejścia/wyjścia (I/O)



1.1. Strumienie bajtowe z linii poleceń

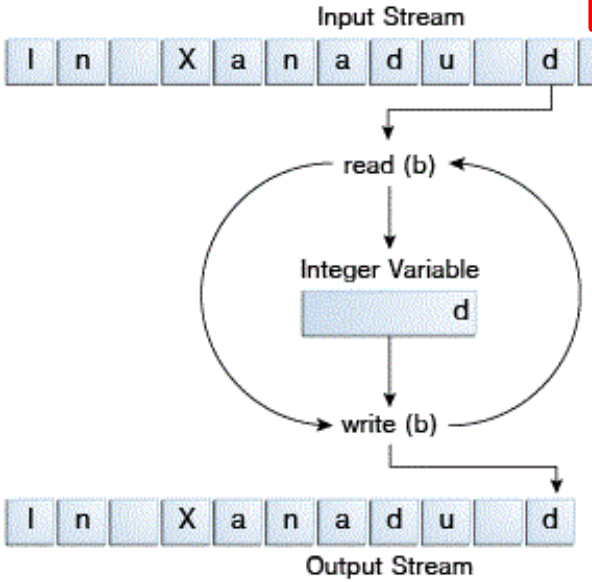
Most pomiędzy stumieniem bajtów a strumieniem znaków – odczytany pierwszy znak i przechowany w strumieniu jego kod




```
package pliki_bajtowe1;

import java.io.*;

public class Pliki_bajtowe1 {

 static int weByte() {

 InputStreamReader wejscie=new InputStreamReader(System.in);
   int dana;
   try {
     System.out.print("Podaj dane: ");
     dana = wejscie.read();    //odczytanie 1 znaku i zwrócenie
     System.out.println(dana);    //kodu ASCII tego znaku
     return dana;
  } catch (IOException e) {
        System.err.println("Blad IO byte1 " + i + " " + e);
        return 0; }
 }
```

1.1. Strumienie bajtowe z linii poleceń (cd)

Most pomiędzy stumieniem znaków a strumieniem bajtów – wyprowadzenie znaku o dostępnej wartości jego kodu




```
static void wyByte(int dana) {
OutputStreamWriter wyjscie =
                    new OutputStreamWriter(System.out);
 try {
          wyjscie.write(dana);          //wyświetlenie 1 znaku
          wyjscie.flush();              //o podanym kodzie ASCII
          wyjscie.write("\n");
          wyjscie.flush();
      } catch (IOException e) {
          System.err.println("Blad IO byte2 " + e);  }
 }
 public static void main(String[] args) {
     int c = weByte();  //odczyt z klawiatury
                wyByte(c); } //zapis na ekran
}
```

```
run:
Podaj dane: 257
50
2
BUILD SUCCESSFUL (total time: 4 seconds)
```

Na wejściu strumienia 3 znaki: 2 5 7
Kod ASCII pierwszego znaku 2: 50
Wyświetlenie znaku o kodzie ASCII równym 50

1.2. Strumienie bajtowe – zapis i odczyt z pliku dyskowego

```
package pliki_bajtowe2;
import java.io.*;


public class Pliki_bajtowe2 {
    static void weByte() {
try (FileInputStream wejscie = new FileInputStream("plikbajtowy.dat"))
        {  System.out.print("Podaj dane z pliku: ");
          while (wejscie.available() != 0)
              System.out.print(wejscie.read());
         System.out.println();
         // wejscie.close();
 } catch (IOException e) {
          System.err.println("Blad IO byte1 " + e);
          }
}
```

Odczyt pliku za pomocą strumienia bajtowego

1.2. Strumienie bajtowe – zapis i odczyt z pliku dyskowego (cd)

Odczyt pliku za pomocą strumienia bajtowego

```
static void wyByte(int dana) {
try (FileOutputStream wyjscie = new FileOutputStream("plikbajtowy.dat"))
 { wyjscie.write(dana);     //zapis tylko jednego bajtu
   wyjscie.write(dana);     //zapis kolejnego tylko jednego bajtu o tej samej  wartości
         // wyjscie.close();
       } catch (IOException e) {
          System.err.println("Blad IO byte2 " + e);  }
   }
   public static void main(String[] args) {
     wyByte(257); //zapis do pliku
     weByte(); //dwukrotny odczyt tylko jednego bajtu o wartości 1, ponieważ 257== 0x101
   }
}
```

```
run:
Podaj dane z pliku: 11
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3. Strumienie znakowe – zalecany, gdy należy przetwarzać strumienie znaków (FileReader i FileWriter)

```
package pliki_znakowe1;
import java.io.*;

public class Pliki_znakowe1 {

   static void weZnaki() {
     int c;
     try (FileReader wejscie = new FileReader("plikznakowy.dat"))
          {  System.out.print("Podaj dane z pliku: ");
             while ((c = wejscie.read()) != -1)   //na końcu pliku odczytany znak -1
               System.out.print(c);
            System.out.println();
           //  wejscie.close();
      } catch (IOException e) {
         System.err.println("Blad IO byte1 " + e); }
   }
```

1.3. Strumienie znakowe – zalecany, gdy należy przetwarzać strumienie znaków (FileReader i FileWriter) (cd)

```
static void wyZnaki(int dana) {
      try (FileWriter wyjscie = new FileWriter("plikznakowy.dat"))
        { wyjscie.write(dana);
          wyjscie.write(dana);
         //  wyjscie.close();
      } catch (IOException e) {
          System.err.println("Blad IO byte2 " + e);  }
   }
  public static void main(String[] args) {
      wyZnaki(257);  //257
      weZnaki(); }
}
```

```
run:
Podaj dane z pliku:  257257
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.4. Liniowo zorientowane pliki znakowe – zastosowanie buforów strumieni znakowych: BufferedReader i BufferWriter

```
package pliki_znakowe2;
import java.io.*;
public class Pliki_znakowe2 {
   static void weZnaki() {
     String linia;
      try (BufferedReader wejscie =
                new BufferedReader (new FileReader("plikznakowylinie.dat")))
           { System.out.print("Podaj dane z pliku: \n");
             while ((linia = wejscie.readLine()) != null)
              System.out.println(linia);
              System.out.println();
            // wejscie.close();   }
    catch (IOException e) {
        System.err.println("Blad IO byte1 " + e);  }
}
```

Buforowanie strumienia znakowego

Buforowany odczyt strumienia

1.4. Liniowo zorientowane pliki znakowe – zastosowanie buforów strumieni znakowych: BufferedReader i BufferWriter (cd)

```
static void wyZnaki1() {
 try (  BufferedWriter wyjscie1 =
            new BufferedWriter(new FileWriter("plikznakowylinie.dat")))
       {  wyjscie1.write(dana+"\n");
          wyjscie1.write(dana+"\n");
          //  wyjscie1.close();
       } catch (IOException e) {
          System.err.println("Blad IO byte2 " + e);
       }
}
```

Buforowanie strumienia znakowego

//zapis danych znakowych znaków
//końca linii: "\r\n", "\r", "\n"

Automatyczne opróżnienie bufora

1.4. Liniowo zorientowane pliki znakowe – zastosowanie buforów strumieni znakowych: BufferedReader i BufferWriter (cd)

```
static void wyZnaki2(int dana) {
  try ( PrintWriter wyjscie =
                new PrintWriter(new FileWriter("plikznakowylinie.dat")))
   {  wyjscie.println(dana);   //zapis danych znakowych i znaków "\r\n", "\r", "\n"
      wyjscie.println(dana);                                    //końca linii:
   //   wyjscie.close();
   } catch (IOException e) {
         System.err.println("Blad IO byte2 " + e);  }
   }
   public static void main(String[] args) {
      wyZnaki1(257); //257
      weZnaki();
      wyZnaki2(257); //257
      weZnaki();    }
}
```

Buforowanie strumienia znakowgo

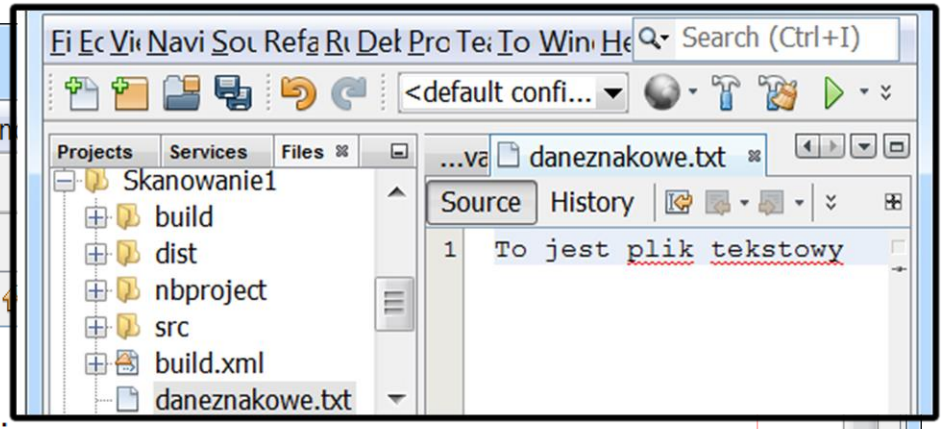
Automatyczne opróżnienie bufora

```
run:
Podaj dane z pliku:
257
257
Podaj dane z pliku:
257
257
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.5. Skanowanie i formatowanie

- **Scaner API -** Interfejs API skanera dzieli dane wejściowe na poszczególne tokeny powiązane z bitami danych.
- **Formatting API -** Formatowanie API składa dane w sformatowaną, czytelną formę dla człowieka .

Skanowanie – program 1



Skanowanie1 - Net

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Win

<default confi...

Projects Services Files

- Skanowanie1
  - Source Packages
    - skanowanie1
      - Skanowanie1.java
  - Libraries
    - JDK 1.8 (Default)
- Sklep_1
- Sklep_7
- Witaj_w3_6

main - Navigator

Members <empty>

- Skanowanie1
  - main(String[] args)

Start Page | Skanowanie1.java

Source History

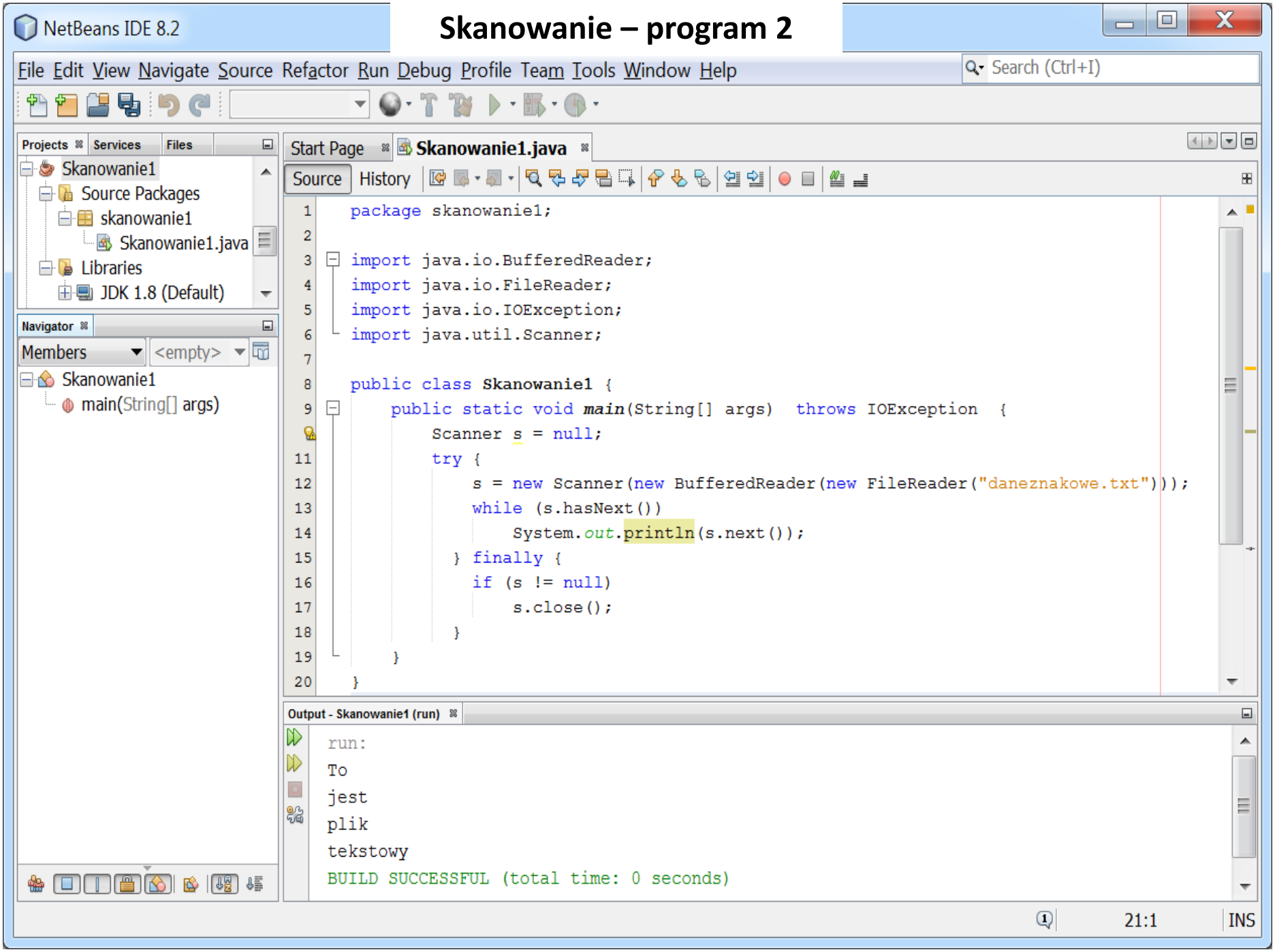
```
1  package skanowanie1;
2
3  import java.io.BufferedReader;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.util.Scanner;
7
8  public class Skanowanie1 {
9
10     public static void main(String[] args)  throws IOException  {
11         Scanner s = null;
12         try {
13             s = new Scanner(new BufferedReader(new FileReader("daneznakowe.txt")));
14             while (s.hasNext())
15                 System.out.print(s.next());
16           } finally {
17             if (s != null)
18                 s.close();
19             }
20     }
21 }
```

Output - Skanowanie1 (run)

```
run:
TojestpliktekstowyBUILD SUCCESSFUL (total time: 0 seconds)
```

15:33 INS

Skanowanie – program 2



Skanowanie – program 3

Skanowanie2 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Hel

<default confi...

Projects Services Files

- Skanowanie2
  - Source Packages
    - skanowanie2
      - Skanowanie2.java
  - Test Packages
  - Libraries
    - JDK 1.8 (Default)
  - Test Libraries

Skanowanie2 - Navigator

Members <empty>

- Skanowanie2
  - main(String[] args)

...ge Skanowanie1.java daneznakowe.txt Sk

Source History

```
1   package skanowanie2;
2
3   import java.io.BufferedReader;
4   import java.io.FileReader;
5   import java.io.IOException;
6   import java.util.Scanner;
7
8   public class Skanowanie2 {
9
10   public static void main(String[] args) throws IOException {
11    Scanner s = null;
12    String pattern="nnn",wynik="brak";
13    try {
14      s = new Scanner(new BufferedReader(new FileReader("liczby.txt")));
15      while (s.hasNext()) {
16       wynik= s.findInLine(pattern);
17       if(wynik!=null)
18          System.out.println(wynik);
19       else
20         s.next();
21      }
22    } finally {
23      s.close(); }
24  }
25  }
```

```
run:
nnn
nnn
BUILD SUCCESSFUL (total time: 0 seconds)
```

25:1 INS

Skanowanie2 - NetBeans IDE ...

F E Vi Nav So Ref R De Pr Te Tc Wir H Search (Ctrl+I

<default confi...

Projects Services Files

- Skanowanie2
  - build
  - dist
  - nbproject
  - src
  - test
  - build.xml
  - liczby.txt
  - manifest.mf

...xt Scanner...

Source History

```
1  8 nnn 89
2  32,767
3  3 nnn
4  1,000,000.1
```

4:9 INS

Skanowanie – program 4

Skanowanie3 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects | Services | Files

- Skanowanie3
  - Source Packages
    - skanowanie3
      - Skanowanie3.java
  - Test Packages
  - Libraries
  - Test Libraries

Skanowanie3 - Navigator

Members <empty>

- Skanowanie3
  - main(String[] args)

...xt | Skanowanie2.java | liczby.txt | *Scanner.java* | **Skanowanie3.java** | liczby.txt...

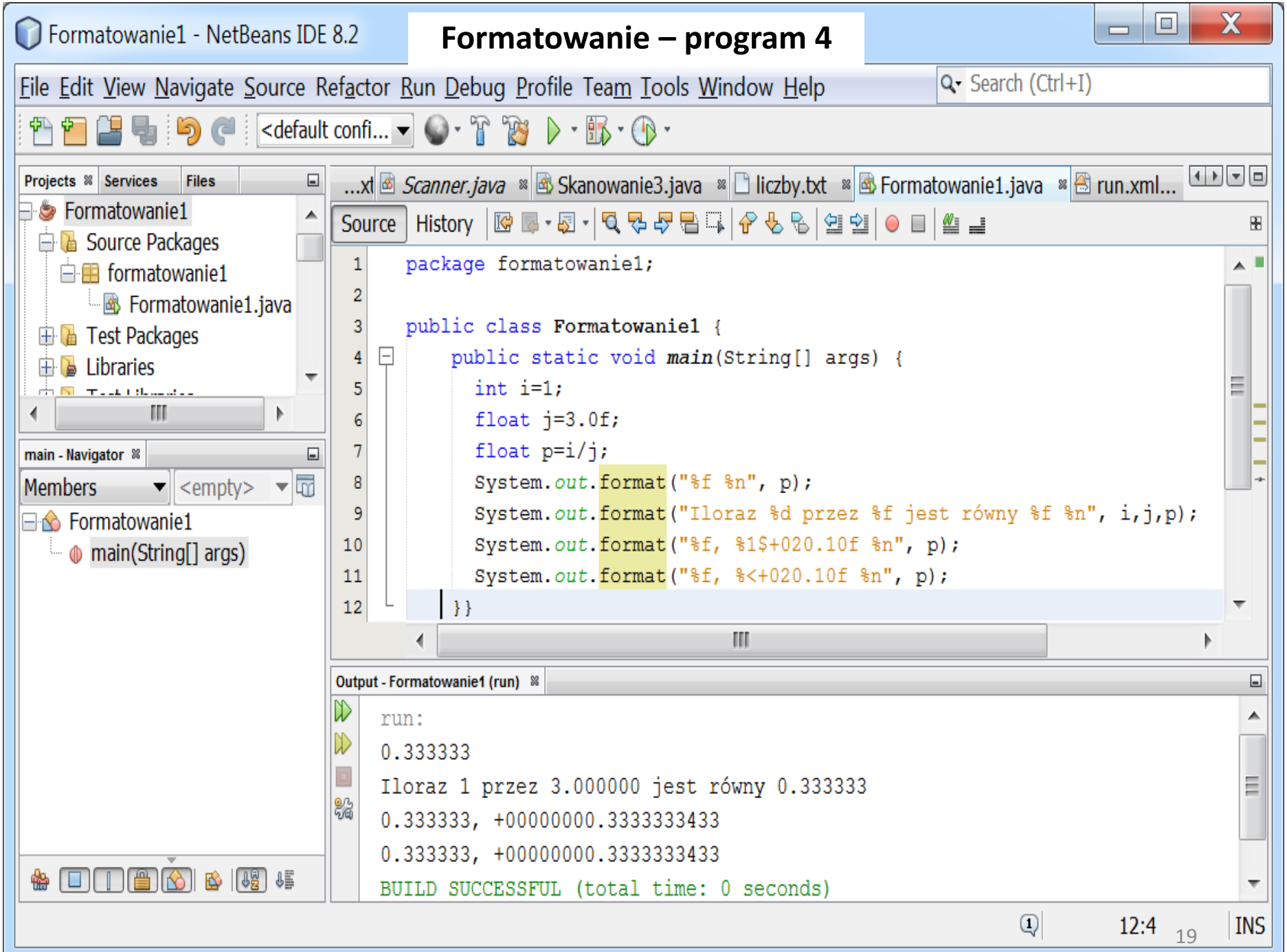
Source History

```
1  package skanowanie3;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  import java.io.IOException;
5  import java.util.Scanner;
6
7  public class Skanowanie3 {
8
9      public static void main(String[] args) throws IOException{
10      Scanner s = null;
11     int suma = 0;
12     try {
13      s = new Scanner(new BufferedReader(new FileReader("liczby.txt")));
14      while (s.hasNext())
15       if (s.hasNextInt())
16          suma += s.nextInt();
17       else
18          s.next();
19     } finally {
          s.close(); }
21        System.out.println(suma);
22     }
23 }
```

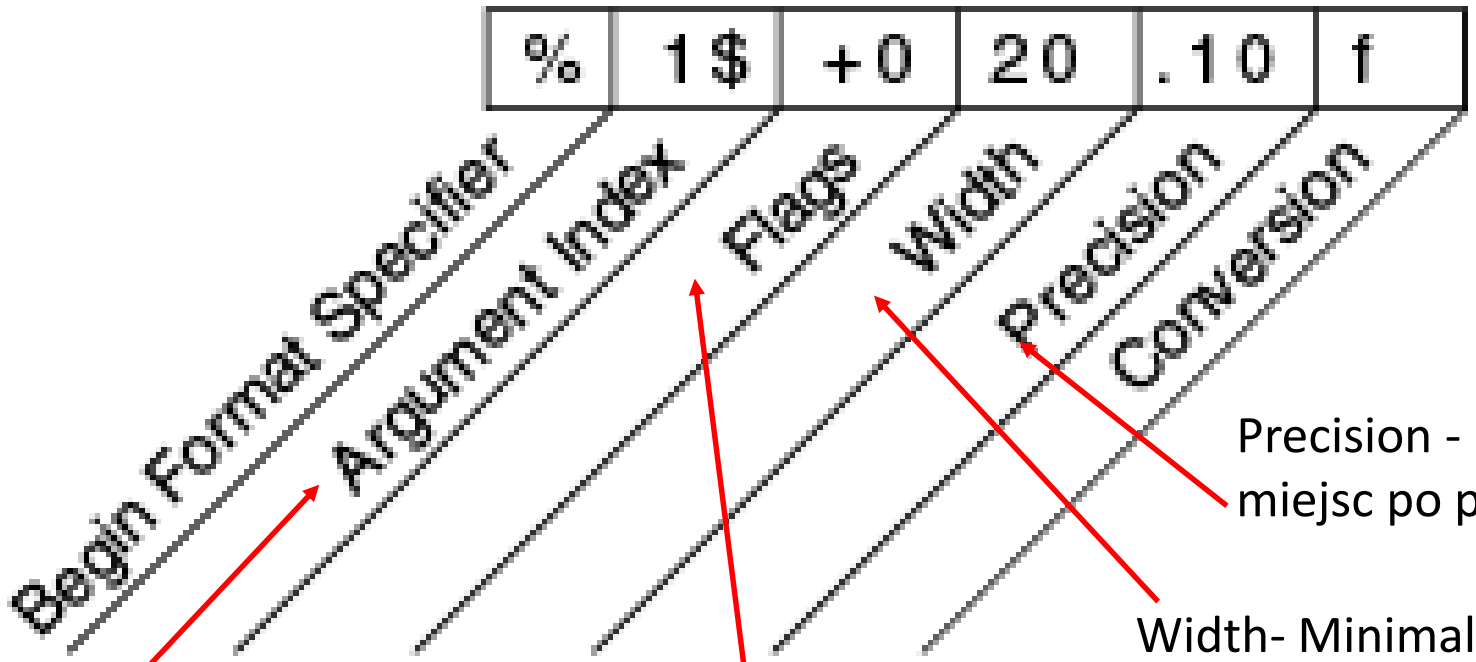
Odczytywanie tylko liczb całkowitych za pomocą metody nextInt

```
run:
100
BUILD SUCCESSFUL (total time: 0 seconds)
```

Formatowanie – program 4



Formatowanie



Dopasowanie kolejnego argumentu do formatowabia

Flags – wartośc z znakiem uzupełniona 0 z możliwym użyciem separatora np dla określenia liczby tysięcy

Precision - Liczba miejsc po przecinku

Width- Minimalny rozmiar przewidziany do przedstawienia wartości

1.6. STRUMIENIE DANYCH

Procedura korzystania ze strumieni danych

Aby utworzyć plik:

1) Należy utworzyć obiekt (np. typu *FileOutputStream*), powiązany ze plikiem danych binarnych (np. "plik3.dat");

FileOutputStream plik = new FileOutputStream(("plik3.dat");

2) W celu buforowania bajtów pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileOutputStream* należy utworzyć obiekt klasy *BufferedOutputStream*

BufferedOutputStream bufor = new BufferedOutputStream(plik);

3) W celu reprezentowania danych typu ***Boolean***, *byte*, ***double***, ***float long***, ***short*** należy utworzyć strumień danych typu *DataOutputStream* powiązanego z obiektem buforującym typu *BufferedOutputStream*

DataOutputStream dana = new DataOutputStream (bufor);

STRUMIENIE DANYCH (cd)

4) Metody (wybrane) strumienia danych do zapisu danych do pliku:

Pojedyncze bajty mogą być zapisywane do pliku za pomocą metody:

void write(int b)

Całe ciągi bajtów mogą być zapisywane do pliku za pomocą metody:

void write(byte[] cbuf, int off, int len) – metoda, która czyta z tablicy *cbuf* od indeksu *off* liczbę *len* bajtów i zapisuje do pliku

void writeBoolean(**boolean** v) – zapisuje do pliku 1-bajtową wartość

void writeByte(**int** v) – zapisuje do pliku 1-bajtową wartość

void writeChar(**int** v) – zapisuje znak jako 2-bajtową wartość

void writeDouble(**double** v) – zapisuje 8-bajtową wartość do pliku

void writeFloat(**float** v) – zapisuje 4-bajtową wartość do pliku

void writeInt(**int** v) – zapisuje 4 bajty do pliku

void writeLong(**long** v) – zapisuje 8 bajtów do pliku

void writeShort(**int** v) – zapisuje 2 bajty do pliku

STRUMIENIE DANYCH (cd)

Aby odczytać plik:

5) Należy utworzyć obiekt (np. typu *FileInputStream*), powiązany ze plikiem danych binarnych (np. "plik3.dat");

```
FileInputStream plik = new FileInputStream(("plik3.dat");
```

6) W celu buforowania bajtów pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileInputStream* należy utworzyć obiekt klasy *BufferedInputStream*

```
BufferedInputStream bufor = new BufferedInputStream (plik);
```

7) W celu reprezentowania danych typu ***Boolean***, *byte*, ***double***, ***float long***, ***short*** należy utworzyć strumień danych typu *DataInputStream* powiązanego z obiektem buforującym typu *BufferedInputStream*

```
DataInputStream dana=   new DataInputStream (bufor);
```

STRUMIENIE DANYCH (cd)

8) Metody strumienia danych do odczytu danych z pliku:

Ciągi bajtów mogą być odczytywane z pliku za pomocą metod:

int read(byte []b) – metoda, która czyta ze strumienia ciąg bajtów i zapisuje do bufora ***b*** bajtów oraz zwraca liczbę odczytanych bajtów

int read(byte[] cbuf, int off, int len) – metoda, która czyta ze strumienia ciąg bajtów i zapisuje do bufora *cbuf* od indeksu *off* liczbę *len* bajtów oraz zwraca liczbę odczytanych bajtów

boolean readBoolean() – czyta z pliku 1 bajt i wraca wartość true lub false

byte readByte() – czyta z pliku 1 bajt i zwraca wartość typu byte

char readChar() – czyta 1 znak (2 bajty) i zwraca 1 znak

double readDouble() – czyta 8 bajtów z pliku i zwraca wartość **double**

float readFloat() – czyta 4 bajtów z pliku i zwraca wartość **float**

int readInt() – czyta 4 bajty z pliku i zwraca wartość typu **int**

long readLong() – czyta 8 bajtów z pliku i zwraca wartość typu **long**

short readShort() – czyta 2 bajty z pliku i zwraca wartość typu **short**

9) Po zapisie i odczycie strumień danych należy zamknąć metodą ***close()*** *lub skorzystać z mechanizmu* java.lang.AutoCloseable.

STRUMIENIE DANYCH (cd)

```
package plikidanych;
import java.io.*;
public class Plikdanych {
    static char weInt() {
     try {
          InputStreamReader wejscie = new InputStreamReader(System.in);
          BufferedReader bufor = new BufferedReader(wejscie);
          System.out.print("Podaj dane: ");
          String s = bufor.readLine();
          return s.charAt(0);
       } catch (IOException e) {
           System.err.println("Blad IO int " + e);
           return 0;
       } catch (NumberFormatException e) {
           System.err.println("Blad formatu int " + e);
           return 0;    }
     }
```

Metoda do odczytania znaku z klawiatury (danej typu char)

STRUMIENIE DANYCH (cd)

```
static void Zapiszplik3_() {
        char dane = '0';
       try ( FileOutputStream plik = new FileOutputStream("plik2.dat");
            BufferedOutputStream bufor = new BufferedOutputStream(plik);
            DataOutputStream wystrdanych = new DataOutputStream(bufor);)
           {
               while (dane != 'k') {
                  dane = weInt();   //odczyt wartości 4-bajtowej  klawiatury
                   if (dane != 'k')
                   wystrdanych.writeInt(dane);
            }
       } catch (IOException e) {
            System.out.println("Blad zapisu pliku bajtowego" + e);
       }
  }
```

Metoda do zapisu danych typu znakowego (char) do pliku za pomocą strumienia danych

STRUMIENIE DANYCH (cd)

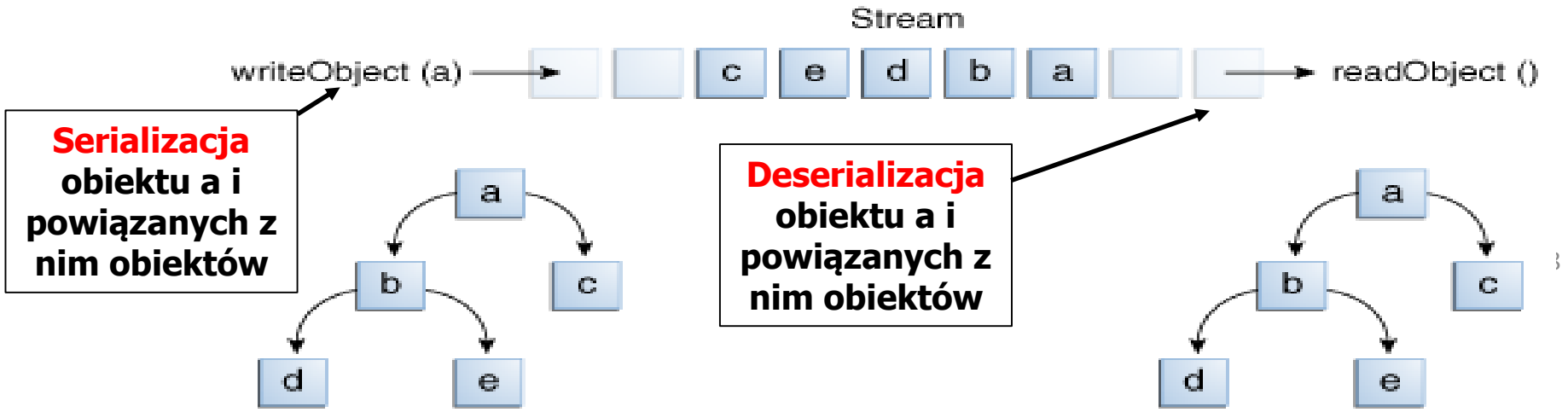
```
static void Odczytajplik3_() {
        int dane = 0;
        try (FileInputStream plik = new FileInputStream("plik2.dat");
             BufferedInputStream bufor = new BufferedInputStream(plik);
             DataInputStream westrdanych = new DataInputStream(bufor);)
          {   while (westrdanych.available()!=0) {
             dane = westrdanych.readInt();
              System.out.print(dane); }
              System.out.println();
           } catch (IOException e) {
          System.out.println("Blad odczytu pliku bajtowe
    }
   public static void main(String[] args) {
        Zapiszplik3_();          //zapis do pliku
        Odczytajplik3_();  } //odczyt z pliku
}
```

Metoda do odczytania danych typu znakowego (char) z pliku za pomocą strumienia danych

```
run:
Podaj dane: 1
Podaj dane: 2
Podaj dane: 3
Podaj dane: 4
Podaj dane: k
49505152
BUILD SUCCESSFUL (total time: 10 seconds)
```

1.7. SERIALIZACJA I DESERIALIZACJA OBIEKTÓW

- Jest to mechanizm szeregowego zapisu do pliku związanego ze strumieniem wyjściowym ciągu bajtów po wykonaniu konwersji obiektu do postaci szeregowej i
- Odczytu szeregowego ciągu bajtów z pliku związanego ze strumieniem wejściowym i dokonanie konwersji do postaci danej (obiektu, typu podstawowego: **int**, **float** itp.)
- Mechanizm ten pozwala zachować całe obiekty w pliku po zakończeniu programu
- **Obiekty zapisywane do pliku muszą implementować pusty interfejs *Serializable***
- **Obiekty z zagnieżdżonymi obiektami są w całości zapisywane do pliku pod warunkiem, że zagnieżdżone obiekty też są serializowane**
- Obiekty zagnieżdżone w serializowanych klasach mogą być pomijane przy zapisie do strumienia, jeśli to konieczne, za pomocą słowa kluczowego **transient** np. **public transient** String s = ”Kowalski”;



STRUMIENIE OBIEKTÓW (cd)

Procedura korzystania ze strumieni obiektowych powiązanych z plikami binarymi

Aby utworzyć plik:

1) Należy utworzyć obiekt (np. typu *FileOutputStream*), powiązany ze plikiem danych binarnych (np. "Wiadomosc.obj");

```
FileOutputStream plikobiektow =
                    new FileOutputStream(("Wiadomosc.obj");
```

2) W celu utworzenia wyjściowego strumienia obiektowego powiązanego z obiektem związanym ze źródłem danych np. typu *FileOutputStream* należy utworzyć obiekt klasy *ObjectOutputStream*

```
ObjectOutputStream strumienobiektow=
                    new BufferedOutputStream (plikobiektow);
```

3) Obiekty dziedziczące po *Object* i implementujace interfejs *Serializable* są zapisywane do pliku w postaci szeregowej za pomocą metody

```
void writeObject(Object ob)
```

STRUMIENIE OBIEKTÓW (cd)

4) Metody strumienia obiektów do zapisu różnych danych do pliku:

Pojedyncze bajty mogą być zapisywane do pliku za pomocą metody:

void write(int b)

Całe ciągi bajtów umieszczone w tablicy bajtów mogą być zapisywane do pliku za pomocą metody:

void write(byte[]cbuf)

Całe ciągi bajtów mogą być zapisywane do pliku za pomocą metody:

void write(int[] cbuf, int off, int len) – metoda, która czyta z tablicy *cbuf* od indeksu *off* liczbę *len* bajtów i zapisuje do pliku

void	writeBoolean(**boolean** v)	– zapisuje do pliku 1-bajtową wartość
void	writeByte(**int** v)	– zapisuje do pliku 1-bajtową wartość
void	writeChar(**int** v)	– zapisuje znak jako 2-bajtową wartość
void	writeDouble(**double** v)	– zapisuje 8-bajtową wartość do pliku
void	writeFloat(**float** v)	– zapisuje 4-bajtową wartość do pliku
void	writeInt(**int** v)	– zapisuje 4 bajty do pliku
void	writeLong(**long** v)	– zapisuje 8 bajtów do pliku
void	writeShort(**int** v)	– zapisuje 2 bajty do pliku

STRUMIENIE OBIEKTÓW (cd)

5) Należy utworzyć obiekt (np. typu *FileInputStream*), powiązany ze plikiem danych binarnych (np. "Wiadomosc.obj");

FileInputStream plik = new FileInputStream(("Wiadomosc.obj");

6) W celu odczytu obiektow pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileInputStream* należy utworzyć obiekt klasy *ObjectInputStream*

ObjectInputStream bufor = new ObjectInputStream (plik);

7) Odczytu obiektów z strumienia należy wykonać za pomocą metody

Object readObject()

STRUMIENIE OBIEKTÓW (cd)

8) Metody strumienia obiektów do odczytu danych z pliku:

boolean readBoolean() – czyta z pliku 1 bajt i wraca wartość true lub false

byte readByte() – czyta z pliku 1 bajt i zwraca wartość typu byte

char readChar() – czyta 1 znak (2 bajty) i zwraca 1 znak

double readDouble() – czyta 8 bajtów z pliku i zwraca wartość **double**

float readFloat() – czyta 4 bajtów z pliku i zwraca wartość **float**

int readInt() – czyta 4 bajty z pliku i zwraca wartość typu **int**

long readLong() – czyta 8 bajtów z pliku i zwraca wartość typu **long**

short readShort() – czyta 2 bajty z pliku i zwraca wartość typu **short**

9) Po zapisie i odczycie strumień obiektów należy zamknąć metodą ***close()*** *lub skorzystać z mechanizmu* **java.lang.AutoCloseable.**

STRUMIENIE OBIEKTÓW (cd)

```
package plikiserializowane;
import java.io.*;
import java.util.*;

class Wiadomosc implements Serializable {
    String dane;
    Date data;
    public void wykonajWiadomosc(Date data_, String dane_) {
        data = data_;
        System.out.println(data);
        dane = dane_;
    }
    public void pokazWiadomosc() {
        System.out.println(data);
        System.out.println(dane);
    }
}
```

STRUMIENIE OBIEKTÓW cd

```
public class Plik1 {

    static String weString() {
        InputStreamReader wejscie = new InputStreamReader(System.in);
        BufferedReader bufor = new BufferedReader(wejscie);
        System.out.print("Podaj wiadomosc: ");
        try {
            return bufor.readLine();
        } catch (IOException e) {
            System.err.println("Blad IO String");
            return "";
        }
    }
```

STRUMIENIE OBIEKTÓW cd

```
static void Zapiszobiektydopliku() {
        Date d = new Date();
        Wiadomosc wiadomosc = new Wiadomosc();
        Wiadomosc wiadomosc1 = new Wiadomosc();
        wiadomosc.wykonajWiadomosc(d, weString());
        wiadomosc1.wykonajWiadomosc(d, weString());
        try (
            FileOutputStream plikobiektow = new
                                                FileOutputStream("Wiadomosc.obj");
            ObjectOutputStream strumienobiektow = new
                                                ObjectOutputStream(plikobiektow);)
            {   strumienobiektow.writeObject(wiadomosc);
                strumienobiektow.writeObject(wiadomosc1);
                // strumienobiektow.close();
              System.out.println("Obiekty typu Wiadomosc zostaly zapisane do pliku");
        } catch (IOException e) {
            System.out.println("Blad zapisu pliku obiektowego" + e); }
}
```

STRUMIENIE OBIEKTÓW cd

```
static void Odczytajobiektyzpliku() {
      Wiadomosc wiadomosci[] = new Wiadomosc[2];
      try (  FileInputStream plikobiektow =
                                        new FileInputStream("Wiadomosc.obj");
          ObjectInputStream strumienobiektow =
                                        new ObjectInputStream(plikobiektow);)
        { for (int i = 0; i < wiadomosci.length; i++) {
            wiadomosci[i] = (Wiadomosc) strumienobiektow.readObject();
          System.out.println("Obiekt typu Wiadomosc zostal odczytany z pliku");
           if (wiadomosci[i] != null) {
              wiadomosci[i].pokazWiadomosc(); }
         }
       // strumienobiektow.close();
      } catch (IOException | ClassNotFoundException e) {
         System.out.println("Blad odczytu pliku obiektowego" + e); }
}
```

cd

STRUMIENIE OBIEKTÓW cd

```
public static void main(String[] args) {
    Zapiszobiektydopliku();  //zapis obiektów do pliku za pomocą serializacji
    Odczytajobiektyzpliku(); //odczyt obiektów z pliku za pomocą deserializacji
    }
```

Wprowadzanie danych do obiektu typu Wiadomosc

Wyświetlenie danych obiektu typu Wiadomosc odczytanych z pliku

```
run:
Podaj wiadomosc: wiadomosc1
Tue Nov 14 01:10:29 CET 2017
Podaj wiadomosc: wiadomosc2
Tue Nov 14 01:10:29 CET 2017
Obiekty typu Wiadomosc zostaly zapisane do pliku
Obiekt typu Wiadomosc zostal odczytany z pliku
Tue Nov 14 01:10:29 CET 2017
wiadomosc1
Obiekt typu Wiadomosc zostal odczytany z pliku
Tue Nov 14 01:10:29 CET 2017
wiadomosc2
BUILD SUCCESSFUL (total time: 14 seconds)
```

Pakiet java.nio.file – wspieranie operacji we/wy (I/O)

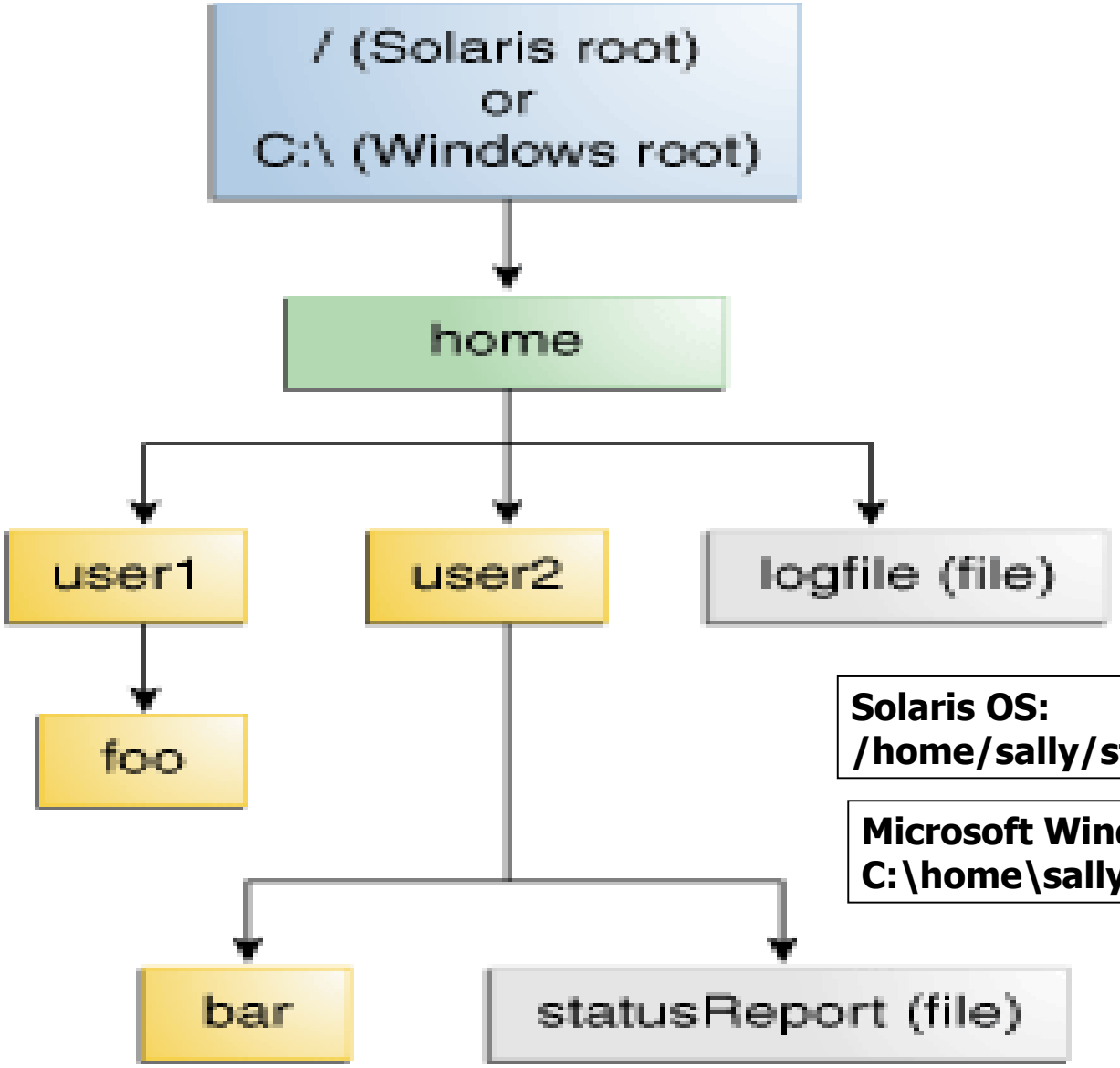
https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

Wspieranie operacji we/wy (I/O)

1. **Ścieżki (path) – tworzenie ścieżek, pobieranie informacji o ścieżce, usuwanie redundancji w oznaczeniu ścieżki, konwersja ścieżki, łączenie ścieżek, tworzenie ścieżki między dwiema ścieżkami, porównanie dwóch ścieżek**
2. **Operacje plikowe z wykorzystaniem scieżek– wyszukiwanie, usuwanie, kopiowanie i przesuwanie plików**
3. *Metadane*
4. *Wyszukiwanie plików za pomocą tzw „wild cards”*

5. *Mapowanie plików do nowych formatów wynikających z wersji Java SE*

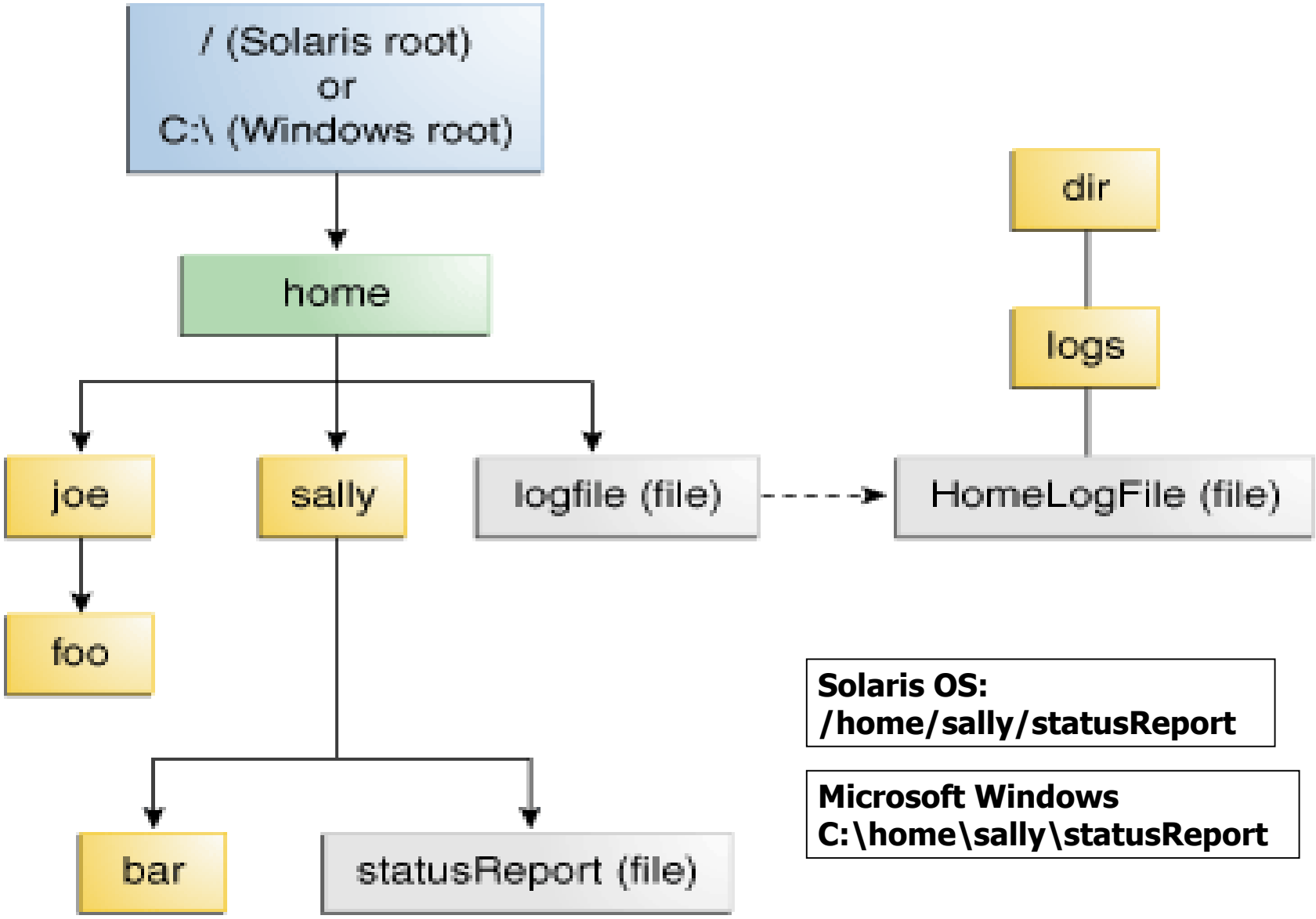
1. Ścieżki (path) – wyszukiwanie, usuwanie, kopiowanie i przesuwanie plików



Solaris OS:
/home/sally/statusReport

Microsoft Windows
C:\home\sally\statusReport

Ścieżki (path) – wyszukiwanie, usuwanie, kopiowanie i przesuwanie plików (cd)



Solaris OS:
/home/sally/statusReport

Microsoft Windows
C:\home\sally\statusReport

ŚCIEŻKI - przykład

```
package sciezki_plikowe;
import java.io.*;
import java.nio.file.NoSuchFileException;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Sciezki_plikowe {
   static void sciezki(String sciezka_) {
      Path sciezka = Paths.get(sciezka_);          //wykonanie obiektu typu Path
       System.out.format("toString: %s%n", sciezka.toString());
       System.out.format("getFileName: %s%n", sciezka.getFileName());
       System.out.format("getName(0): %s%n", sciezka.getName(0));
       System.out.format("getNameCount: %d%n", sciezka.getNameCount());
       System.out.format("subpath(0,2): %s%n", sciezka.subpath(0, 2));
       System.out.format("getParent: %s%n", sciezka.getParent());
       System.out.format("getRoot: %s%n", sciezka.getRoot());
       sciezka = sciezka.normalize();
       System.out.format("Po normalizacji: getParent: %s%n", sciezka.getParent());
       System.out.format("getNameCount: %d%n", sciezka.getNameCount());
   }
```

Parametry ścieżek

ŚCIEŻKI – przykład cd

```
static void konwersja_sciezki(String sciezka_) {
    Path wejsciowa_sciezka, pelna_sciezka1, pelna_sciezka2;
    wejsciowa_sciezka = Paths.get(sciezka_);   //utworzenie obiektu typu Path
    System.out.format("Nazwa pliku bez sciezki: toString: %s%n",
                                  wejsciowa_sciezka.toString());
    pelna_sciezka1 = wejsciowa_sciezka.toAbsolutePath();
    System.out.format("wejsciowa_sciezka.toAbsolutePath(): toString: %s%n",
                                  pelna_sciezka1.toString());
    try {
        pelna_sciezka2 = wejsciowa_sciezka.toRealPath();
        System.out.format("pelna_sciezka.toRealPath(): toString: %s%n",
                                     pelna_sciezka2.toString());
    } catch (NoSuchFileException x) {
        System.err.format("%s: no such" + " file or directory%n",wejsciowa_sciezka);
    } catch (IOException x) {
        System.err.format("%s%n", x); }
}
```

ŚCIEŻKI – przykład cd

```
static void laczenie_sciezek(String s1, String s2) {
    Path sciezka1 = Paths.get(s1);
    Path sciezka2 = Paths.get(s2);
    Path sciezka12 = sciezka1.resolve(sciezka2);
    System.out.format("Sciezka12: toString: %s%n", sciezka12.toString());
    Path sciezka21 = sciezka2.resolve(sciezka1);
    System.out.format("Sciezka21: toString: %s%n", sciezka21.toString());
}
static void sciezka_laczaca_dwie_sciezki(String s1, String s2) {
    Path sciezka1 = Paths.get(s1);
    Path sciezka2 = Paths.get(s2);
    Path sciezka1_2 = sciezka1.relativize(sciezka2);
    System.out.format("Sciezka1_2: toString: %s%n", sciezka1_2.toString());
    Path sciezka2_1 = sciezka2.relativize(sciezka1);
    System.out.format("Sciezka2_1: toString: %s%n", sciezka2_1.toString());
}
```

ŚCIEŻKI – przykład cd

```
static void porownanie_sciezek(String s1, String s2) {
    Path sciezka1 = Paths.get(s1);
    Path sciezka2 = Paths.get(s2);
    Path sciezka12 = sciezka1.resolve(sciezka2);
    Path sciezka21 = sciezka2.resolve(sciezka1);
    if (sciezka2.equals(sciezka12))
          System.out.format("Sciezka1 jest rowna sciezka 2 %s%n",
                                  sciezka1.toString()+" "+  sciezka2.toString());
    if (sciezka12.startsWith(sciezka21))
      System.out.format("Sciezka1 startuje jako sciezka2_1: %s%n",
                                  sciezka12.toString()+" "+ ciezka21.toString());
    if (sciezka2.endsWith(sciezka12))
        System.out.format("Sciezka2 konczy sie jako koniec: %s%n",
                                  sciezka2.toString()+" " + sciezka12.toString());
  }
```

ŚCIEŻKI – przykład cd

```
public static void main(String[] args) {
   sciezki("C:\\Studia\\Szkola\\Wyklady\\Wyklady\\..\\Przyklady_wyklady\\
                    wyklad4_1\\Pliki_znakowe_sciezki");                    //1
   sciezki("C:\\Studia\\Szkola\\Wyklady\\.\\Przyklady_wyklady\\wyklad4_1\\
                    Pliki_znakowe_sciezki");                               //2
   if (args.length > 0) {                                                  //3
       konwersja_sciezki(args[0]);  } //podanie nazwy pliku z linii poleceń
                                                          //plikznakowy.dat

   laczenie_sciezek("C:\\Studia\\", "C:\\Studia\\Szkola\\mmm");            //4
   sciezka_laczaca_dwie_sciezki("C:\\Studia\\", "C:\\Studia\\Szkola\\mmm"); //5
   porownanie_sciezek("C:\\Studia\\", "C:\\Studia\\Szkola\\mmm");          //6
   }
}
```

ŚCIEŻKI – przykład cd

Parametry ścieżek – działanie metody sciezki z dodanym nadmiarowym elementem Wyklady\..\ typu „parent directory” poddanym normalizacji //1

run:
toString: C:\Studia\Szkola\Wyklady\Wyklady\..\Przyklady_wyklady\wyklad4_1\Pliki_znakowe_sciezki
getFileName: Pliki_znakowe_sciezki
getName(0): Studia
getNameCount: 8
subpath(0,2): Studia\Szkola
getParent: C:\Studia\Szkola\Wyklady\Wyklady\..\Przyklady_wyklady\wyklad4_1
getRoot: C:\
Po normalizacji: getParent: C:\Studia\Szkola\Wyklady\Przyklady_wyklady\wyklad4_1
getNameCount: 6

ŚCIEŻKI – przykład cd

Parametry ścieżek – działanie metody sciezki z dodanym nadmiarowym elementem \.\ typu „current directory” poddanym normalizacji //2

toString: C:\Studia\Szkola\Wyklady\.\Przyklady_wyklady\wyklad4_1\Pliki_znakowe_sciezki
getFileName: Pliki_znakowe_sciezki
getName(0): Studia
getNameCount: 7
subpath(0,2): Studia\Szkola
getParent: C:\Studia\Szkola\Wyklady\.\Przyklady_wyklady\wyklad4_1
getRoot: C:\
Po normalizacji: getParent: C:\Studia\Szkola\Wyklady\Przyklady_wyklady\wyklad4_1
getNameCount: 6

ŚCIEŻKI – przykład cd

Nazwa pliku bez sciezki: toString: **plikznakowy.dat**

Wynik działania metody konwersja_sciezki //3

wejsciowa_sciezka.toAbsolutePath(): toString:
C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\wyklad4_1\Pliki_znakowe_sciezki\plikznakowy.dat

pelna_sciezka.toRealPath(): toString:
C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\wyklad4_1\Pliki_znakowe_sciezki\plikznakowy.dat

Sciezka12: toString: C:\Studia\Szkola\mmm

Sciezka21: toString: C:\Studia

Wynik działania metody łaczenie_sciezek //4

Sciezka1_2: toString: Szkola\mmm

Sciezka2_1: toString: ..\..

Wynik działania metody sciezka_laczaca_dwie_sciezki //5

Sciezka1 jest rowna sciezka2 **C:\Studia C:\Studia\Szkola\mmm**

Sciezka1 startuje jako sciezka21: **C:\Studia\Szkola\mmm C:\Studia**

Sciezka2 konczy sie jako sciezka12: **C:\Studia\Szkola\mmm C:\Studia\Szkola\mmm**

BUILD SUCCESSFUL (total time: 0 seconds)

Wynik działania metody porownanie_sciezek //6

2.1. OPERRACJE PLIKOWE – wykorzystanie ścieżek

```
package operacje_plikowe;

import java.io.IOException;
import java.nio.file.DirectoryNotEmptyException;
import java.nio.file.Files;
import java.nio.file.NoSuchFileException;
import java.nio.file.Path;
import java.nio.file.Paths;
import static java.nio.file.StandardCopyOption.REPLACE_EXISTING;

public class Operacje_plikowe {
   static void operacjeplikowe(String sciezka_) {
      Path sciezka = Paths.get(sciezka_);
      boolean isRegularExecutableFile = Files.isRegularFile(sciezka) &
                                        Files.isReadable(sciezka) &
                                        Files.isExecutable(sciezka);
      System.out.format("wynik: %b%n", isRegularExecutableFile);
   }
```

OPERRACJE PLIKOWE – wykorzystanie ścieżek cd

```
static void usuwanie_plikow(String sciezka_) {
        Path sciezka = Paths.get(sciezka_);
        try {
            Files.delete(sciezka);
        } catch (NoSuchFileException x) {
            System.err.format("%s: no such" + " file or directory%n", sciezka);
        } catch (DirectoryNotEmptyException x) {
            System.err.format("%s not empty%n", sciezka);
        } catch (IOException x) {
            System.err.println(x);
        }
    }
```

OPERRACJE PLIKOWE – wykorzystanie ścieżek cd

```
static void kopiowanie_plikow(String sciezka1_, String sciezka2_) {
    Path sciezka1 = Paths.get(sciezka1_);
    Path sciezka2 = Paths.get(sciezka2_);
    try {
        Files.copy(sciezka1, sciezka2, REPLACE_EXISTING);    //COPY_ATTRIBUTES,
                                                             // NOFOLLOW_LINKS
        Files.move(sciezka1, sciezka2, REPLACE_EXISTING);
    } catch (IOException x) {
        System.err.println(x); }
}
```

OPERRACJE PLIKOWE – wykorzystanie ścieżek cd

```
public static void main(String[] args) {
 operacjeplikowe1("C:\\Studia\\Szkola\\CalaJava\\Wyklady\\Przyklady_wyklady\\
            wyklad4_1\\Pliki_znakowe_sciezki\\dist\\Pliki_znakowe_sciezki.jar");
 operacjeplikowe1("C:\\Studia\\Szkola\\CalaJava\\Wyklady\\Przyklady_wyklady\\
                        wyklad4_1\\Pliki_znakowe_sciezki\\
                       src\\sciezki_plikowe\\Sciezki_plikowe.java");
   operacjeplikowe1("C:\\Studia\\Szkola\\CalaJava\\Wyklady\\
                        Przyklady_wyklady\\wyklad4_1\\Pliki_znakowe_sciezki\\
                      build\\classes\\sciezki_plikowe\\Sciezki_plikowe.class");
   operacjeplikowe1("C:\\Studia\\Szkola\\Wyklady\\Przyklady_wyklady\\wyklad4_1\\
                        Pliki_znakowe_sciezki\\plikznakowy.txt");
   usuwanie_plikow("C:\\Studia\\Szkola\\plik.txt");
   kopiowanie_plikow("C:\\Studia\\Szkola\\info.txt","C:\\Studia\\info.txt");    }
}
```

OPERRACJE PLIKOWE – wykorzystanie ścieżek cd

Wynik pierwszego uruchomienia programu

run:

wynik: true
wynik: true
wynik: true
wynik: false

Wynik działania metody operacjeplikowe1:
1. **True dla plików typu class, java i jar**
2. **False dla zwykłego pliku tekstowego plikznakowy.txt**

C:\Studia\Szkola\plik.txt: no such file or directory

Wynik działania metody usuwanie_plikow za pomocą delete w przypadku braku pliku

BUILD SUCCESSFUL (total time: 0 seconds)

Wynik drugiego uruchomienia programu

run:

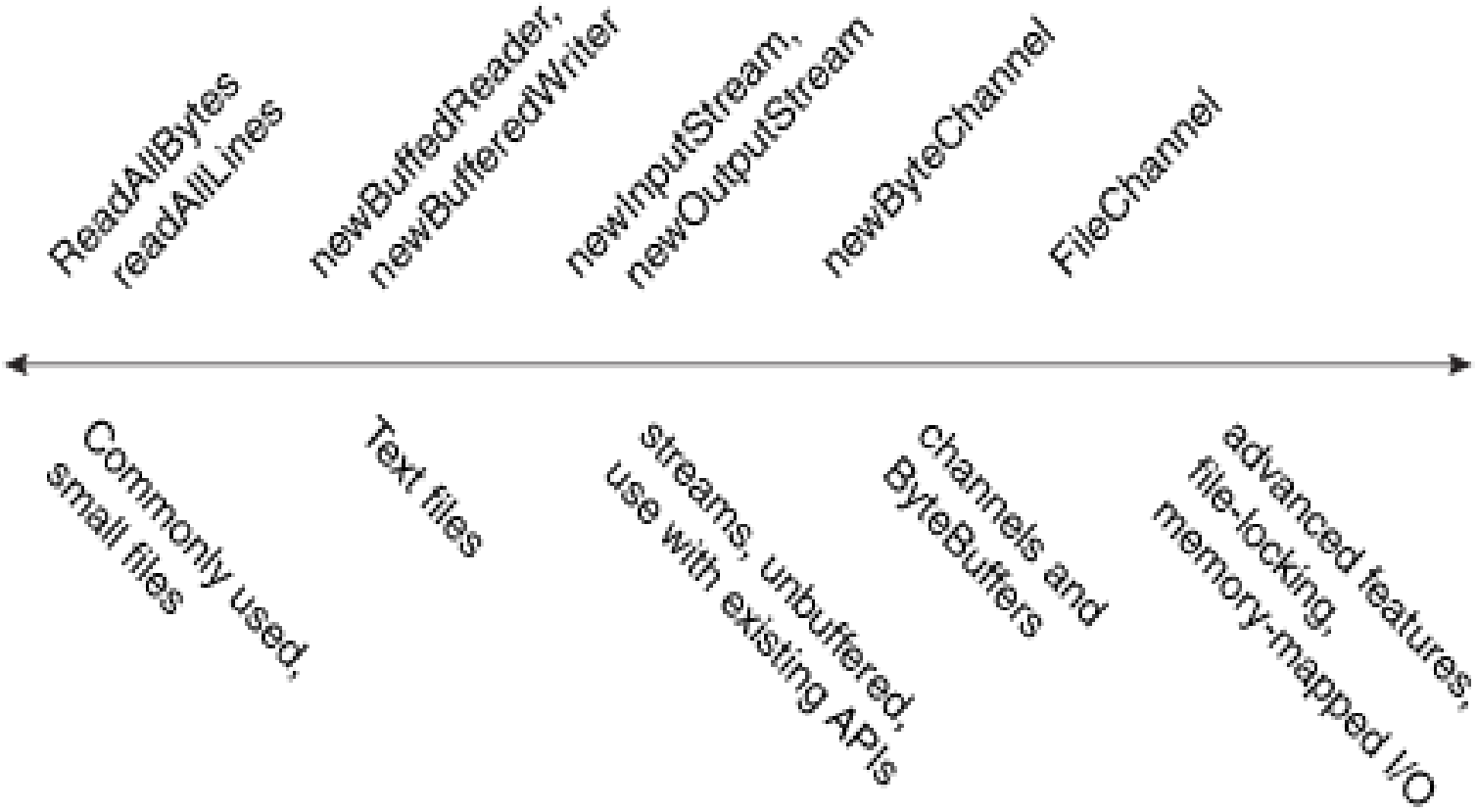
wynik: true
wynik: true
wynik: true
wynik: false

Wynik działania metody kopiowanie_plikow za pomocą metody copy po przesunięciu pliku metodą move podczas pierwszego wywołania programu

C:\Studia\Szkola\plik.txt: no such file or directory
java.nio.file.NoSuchFileException: C:\Studia\Szkola\info.txt

BUILD SUCCESSFUL (total time: 0 seconds)

2.2. Plikowe metody wejścia\wyjścia – złożoność



Wzrost złożoności operacji plikowych od lewej do prawej strony

Kanał We/Wy i bufory bajtowe

```
package channelio;

import java.nio.*;
import java.nio.channels.*;
import java.nio.file.*;
import java.io.*;
import java.nio.charset.Charset;

public class ChannelIO {
    public static void main(String[] args) {
        read();
    }
```

Odczytywany plik permissions.log

```
abcdefghijklmnop
ABCDEFGHIJKLMNOP
abcdefghijklmnop
```

Wynik odczytania pliku

```
run:
abcdefghijklmnop
ABCDEFGHIJKLMNOP
abcdefghijklmnop
bcdefghijklmn
 Liczba odczytów: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kiedy strumień We/Wy czyta znak, w tym samym czasie kanał We/Wy czyta bufor.

Kanał We/Wy i bufory bajtowe cd

```
static void read() {
    Path sciezka=Paths.get("C:\\Studia\\Szkola\\CalaJava\\Wyklady\\
                    Przyklady_wyklady\\wyklad4_1\\ChannelIO\\permissions.log");
    try (SeekableByteChannel sbc = Files.newByteChannel(sciezka)) {
        ByteBuffer buf = ByteBuffer.allocate(16);
         int i = 0;
        String encoding = System.getProperty("file.encoding");
        while (sbc.read(buf) > 0) {
            buf.rewind();
            System.out.print(Charset.forName(encoding).decode(buf));
            i++;
            buf.flip(); }
       System.out.println("\n Liczba odczytów: " + i);
    } catch (IOException x) {
       System.out.println("caught exception: " + x);  }
  }
}
```

Kiedy strumień We/Wy czyta znak, w tym samym czasie kanał We/Wy czyta bufor