Strumienie tekstowe (wprowadzanie danych z klawiatury) i bajtowe, otwieranie strumieni przez sieć - obiekty URL

1. Strumienie tekstowe – wprowadzania danych z klawiatury

1) Należy utworzyć obiekt (np. typu *InputStreamReader*), powiązany ze źródłem danych (np. *System.in*)
*InputStreamReader wejscie = **new** InputStreamReader(System.in);*

2) W celu buforowania znaków pochodzących z obiektu powiązanego ze źródłem danych np. typu *InputStreamReader* należy utworzyć obiekt klasy *BufferedReader*
*BufferedReader bufor = **new** BufferedReader(wejscie);*
lub *BufferedReader bufor = **new** BufferedReader(wejscie, **int**);* //**int**-rozmiar bufora

3) Znaki mogą być odczytywane metodami obiektu buforującego:
***int** read()*
*np. **int** dana = bufor.read()*

4) Całe ciągi znaków mogą być odczytane za pomocą:
***int** read(**char**[] cbuf, **int** off, **int** len)* – metoda, która czyta do tablicy *cbuf* od indeksu *off* liczbę *len* znaków i zwraca przez **return** liczbę faktycznie odczytanych znaków

5) Całe wiersze mogą być odczytywane za pomocą metody obiektu buforującego:
String readLine()
np. String dana = bufor.readLine(),

która czyta łańcuch typu String lub jego wartość **null**, gdy będzie osiągnięty koniec łańcucha (czyli koniec linii oznaczony znakami np.: '\n','\r')

6) Analiza odczytanego łańcucha może być przeprowadzona za pomocą obiektu klasy typu *StringTokenizer*

*StringTokenizer bon = **new** StringTokenizer(bufor.readLine());*

7) Innym sposobem jest analiza odczytanego łańcucha typu *String* za pomocą metod klasy *String*

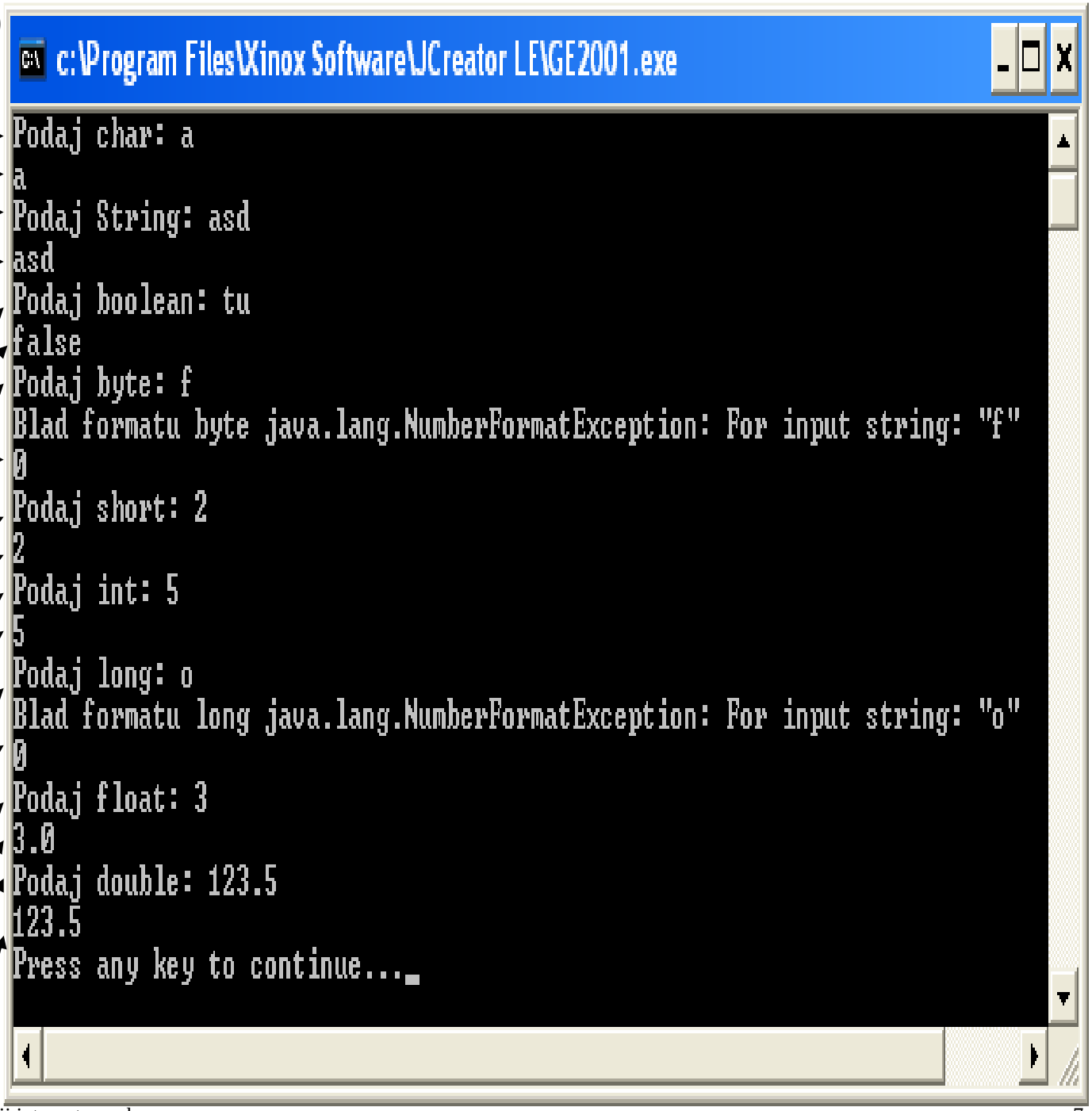
```
import java.io.*;
import java.util.*;

public class WEWY          //plik WEWY.java
 {                         //wejściowy strumień tekstowy odczytuje strumien stdin (System.in)
  static InputStreamReader wejscie = new InputStreamReader( System.in );
                           //bufor – klasa odczytuje wejściowy strumień znakowy wejscie i  przechowuje odczytywane znaki w buforze
  static BufferedReader bufor = new BufferedReader( wejscie );
  StringTokenizer bon;     //klasa do analizy składniowej jednostek lerksykalnych tzw. Leksemów (tokens) pobieranych metodą nextToken()
boolean weBoolean()
   { try
     { bon = new StringTokenizer(bufor.readLine());
      return new Boolean(bon.nextToken()).booleanValue();    }
    catch (IOException e)
     { System.err.println("Blad IO Boolean "+e);
      return false;  }
    catch (Exception e)
     { System.err.println( "Blad Boolean "+e);
      return false;   }
   }
 String weString()
    { try
      { return bufor.readLine(); }
     catch (IOException e)
      { System.err.println("Blad IO String");
       return ""; }
     catch (Exception e)
     { System.err.println( "Blad String "+e);
      return "";   } }
```

```
 char weChar()
   { try
      { String s = bufor.readLine();
       return s.charAt(0);   }
     catch (IOException e)
      { System.err.println("Blad IO char "+e);
       return 0;  }
    catch (Exception e)
     { System.err.println( "Blad char "+e);
        return 0;   }
   }

byte weByte()
  { try
     { bon = new StringTokenizer(bufor.readLine());
      return Byte.parseByte(bon.nextToken());
     }
    catch (IOException e)
     { System.err.println("Blad IO byte "+e);
      return 0;  }
    catch (NumberFormatException e)
     { System.err.println( "Blad formatu byte "+e);
      return 0;   }
    catch (Exception e)
     { System.err.println( "Blad formatu byte "+e);
      return 0;   }
  }
```

```
short weShort()
  { try
       { bon = new StringTokenizer(bufor.readLine());
         return Short.parseShort(bon.nextToken()); }
      catch (IOException e)
     { System.err.println("Blad IO short: "+e);
      return 0;   }
     catch (NumberFormatException e)
      { System.err.println( "Blad formatu short "+e);
       return 0;   }
     catch (Exception e)
     { System.err.println( "Blad formatu short "+e);
      return 0;   }
    }
 int weInt()
    { try
       { bon = new StringTokenizer(bufor.readLine());
         return Integer.parseInt(bon.nextToken()); }
   catch (IOException e)
    { System.err.println("Blad IO int "+e);
     return 0;        }
    catch (NumberFormatException e)
     { System.err.println( "Blad formatu int "+e);
      return 0;   }
    catch (Exception e)
     { System.err.println( "Blad formatu int "+e);
       return 0;   }
     }
long weLong()
  { try
      { bon = new StringTokenizer(bufor.readLine());
        return Long.parseLong(bon.nextToken());  }
   catch (IOException e)
    { System.err.println("Blad IO "+e);
     return 0L; }
   catch (NumberFormatException e)
    { System.err.println( "Blad formatu long "+e);
     return 0L;   }
   catch (Exception e)
    { System.err.println( "Blad formatu long "+e);
      return 0L;   }
    }
```

```
float weFloat()
 { try
     { bon = new StringTokenizer(bufor.readLine());
      return new Float(bon.nextToken()).floatValue(); }
    catch (IOException e)
     { System.err.println("Blad IO float "+e);
      return 0.0F;  }
    catch (NumberFormatException e)
     { System.err.println( "Blad formatu float "+e);
      return 0.0F;   }
    catch (Exception e)
     { System.err.println( "Blad formatu float "+e);
      return 0.0F;   }
  }

double weDouble()
 { try
    { bon = new StringTokenizer(bufor.readLine());
     return new Double(bon.nextToken()).doubleValue(); }
   catch (IOException e)
    { System.err.println("Blad IO double "+e);
     return 0.0; }
   catch (NumberFormatException e)
    { System.err.println( "Blad formatu double "+e);
     return 0;   }
    catch (Exception e)
     { System.err.println( "Blad formatu double "+e);
       return 0.0;   }
 }
```

```
public static void main (String args[])
 {
  WEWY we = new WEWY();
  System.out.print("Podaj char: ");
  System.out.println(we.weChar());

  System.out.print("Podaj String: ");
  System.out.println(we.weString());

  System.out.print("Podaj boolean: ");
  System.out.println(we.weBoolean());

  System.out.print("Podaj byte: ");
  System.out.println(we.weByte());

  System.out.print("Podaj short: ");
  System.out.println(we.weShort());

  System.out.print("Podaj int: ");
  System.out.println(we.weInt());

  System.out.print("Podaj long: ");
  System.out.println(we.weLong());

  System.out.print("Podaj float: ");
  System.out.println(we.weFloat());

  System.out.print("Podaj double: ");
  System.out.println(we.weDouble());
 }
}
```



2. Strumienie tekstowe plikowe

Procedura korzystania ze strumieni tekstowych buforowanych powiązanych z plikami tekstowymi

Aby utworzyć plik:

1) Należy utworzyć obiekt (np. typu *FileWriter*), powiązany ze plikiem danych tekstowych (np. "plik2.txt");
   *FileWriter plik = **new** FileWriter("plik2.txt");*

2) W celu buforowania znaków pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileWriter* należy utworzyć obiekt klasy *BufferedWriter*

   *BufferedWriter bufor = **new** BufferedWriter(plik);*

3) Znaki mogą być zapisywane do pliku za pomocą metody bufora:
   ***void** write(**int** c)*

4) Całe ciągi znaków mogą być zapisywane do pliku za pomocą metody bufora:

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

5) Część łańcucha można zapisać do pliku za pomocą metody bufora:

   ***void** write(String str, **int** off, **int** len)* – metoda, która czyta z łańcucha *str* od indeksu *off* liczbę *len* znaków i zapisuje do pliku

Aby odczytać plik:

6) Należy utworzyć obiekt (np. typu *FileReader*), powiązany ze plikiem danych tekstowych (np. "plik1.txt");
*FileReader plik = **new** FileReader("plik2.txt");*

7) W celu buforowania znaków pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileWriter* należy utworzyć obiekt klasy *BufferedReader*
*BufferedReader bufor = **new** BufferedReader(plik);*

8) Znaki mogą być odczytywane metodą bufora:
***int** read ();*
np. ***int** dane = plik.read()*

9) Całe ciągi znaków mogą być odczytane za pomocą metody bufora:

***int** read(**char**[] cbuf, **int** off, **int** len)* – metoda, która czyta plik i zapisuje do tablicy *cbuf* od indeksu *off* liczbę *len* znaków i zwraca przez **return** liczbę faktycznie odczytanych znaków

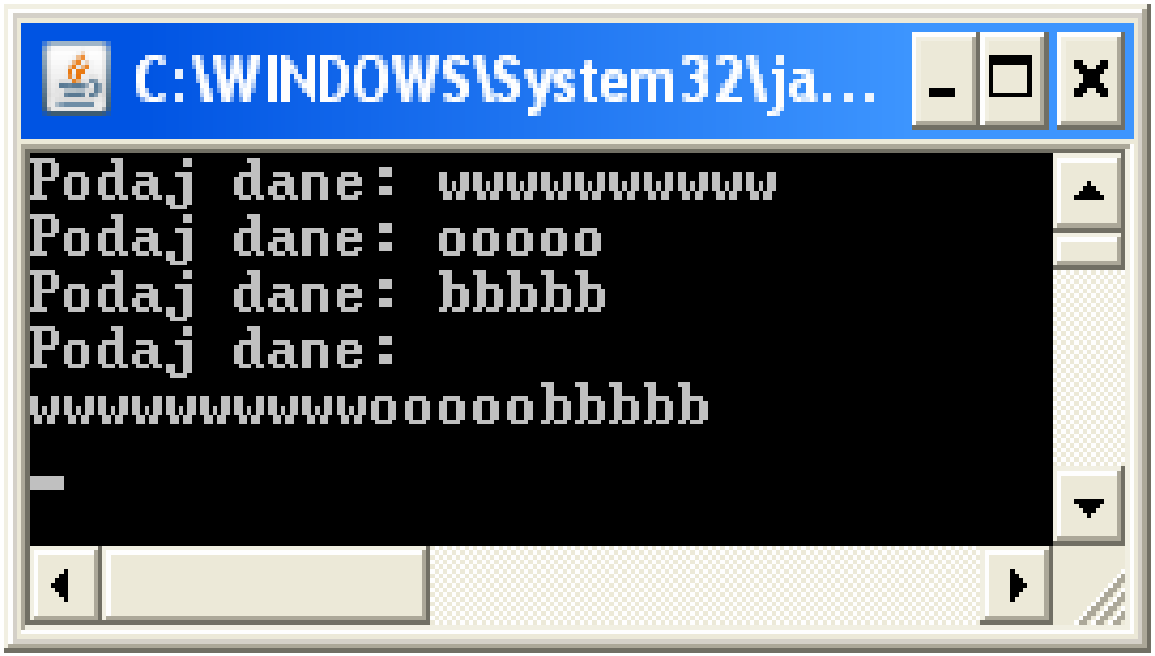
10) Całe wiersze mogą być odczytywane za pomocą metody obiektu buforującego:
String readLine()
np. String dana = bufor.readLine(),

która czyta w pliku łańcuch typu *String* lub jego wartość **null**, gdy będzie osiągnięty koniec łańcucha (czyli koniec linii oznaczony znakami np.: '\n','\r')

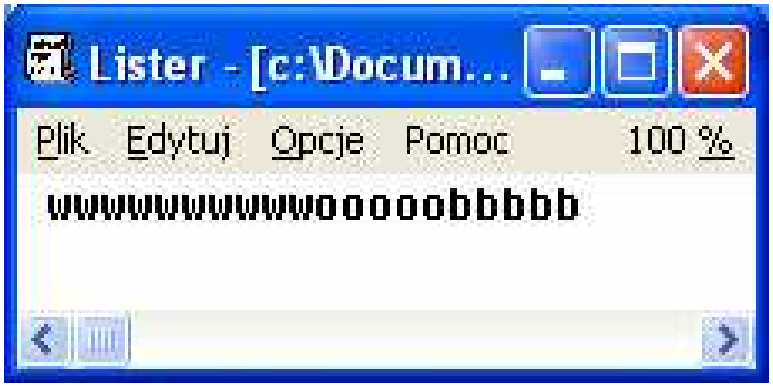
11) Po zapisie i odczycie *bufor* należy zamknąć metodą *close()*

```
import java.io.*;
import java.util.*;
public class WEWY5
 {
   static String weString()
    { InputStreamReader wejscie = new InputStreamReader( System.in );
      BufferedReader bufor = new BufferedReader( wejscie );
      try
       {  return bufor.readLine();  }
      catch (IOException e)
        {  System.err.println("Blad IO String");  return ""; }
    }
    static void Zapiszplik5()
     { String dane="1";
       try
        { FileWriter plik = new FileWriter ("plik2.txt");
          BufferedWriter bufor = new BufferedWriter (plik);
           while (!dane.equals(""))
             { System.out.print("Podaj dane: ");
               dane=weString();
               if (!dane.equals(""))
                  bufor.write(dane, 0, dane.length()); }
            bufor.close();
           } catch (IOException e)
                 {  System.out.println ("Blad zapisu pliku tekstowego"+e);  }
       }
```

```
  static void Odczytajplik5()
   {
     String dane="0";
      try
       {
         FileReader plik = new FileReader ("plik2.txt");
         BufferedReader bufor = new BufferedReader (plik);
          dane=bufor.readLine();
          while (dane!=null)
                {
                   System.out.print(dane);
                   dane=bufor.readLine();
                }
          bufor.close();
          System.out.println();
        } catch (IOException e)
            {  System.out.println ("Blad odczytu pliku tekstowego"+e);  }
      }

  public static void main(String[] args)
    {
      Zapiszplik5();
      Odczytajplik5();
    }
}
```



Wyświetlenia zawartości pliku znakowego w programie.



Odczyt zawartości pliku edytorem znakowym

3. Strumienie bajtowe plikowe

3.1. Procedura korzystania ze strumieni bajtowych buforowanych powiązanych z plikami binarymi

Aby utworzyć plik:

1) Należy utworzyć obiekt (np. typu *FileOutputStream*), powiązany ze plikiem danych binarnych (np. "plik2.dat");
   *FileOutputStream plik = **new** FileOutputStream("plik2.dat");*

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

   Przykład
   *BufferedOutputStream bufor = **new** BufferedOutputStream(plik);*

3) Pojedyncze bajty mogą być zapisywane do pliku za pomocą metody bufora:
   ***void** write(**int** c)*

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

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

Aby odczytać plik:

5) Należy utworzyć obiekt (np. typu *FileInputStream*), powiązany ze plikiem danych binarnych (np. "plik2.dat");

*FileInputStream plik = **new** FileInputStream("plik2.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) Pojedyncze bajty mogą być odczytywane metodą bufora:
***int** read ();*
np. ***int** dane = plik.read()*

8) Całe ciągi bajtów mogą być odczytane za pomocą metody bufora:

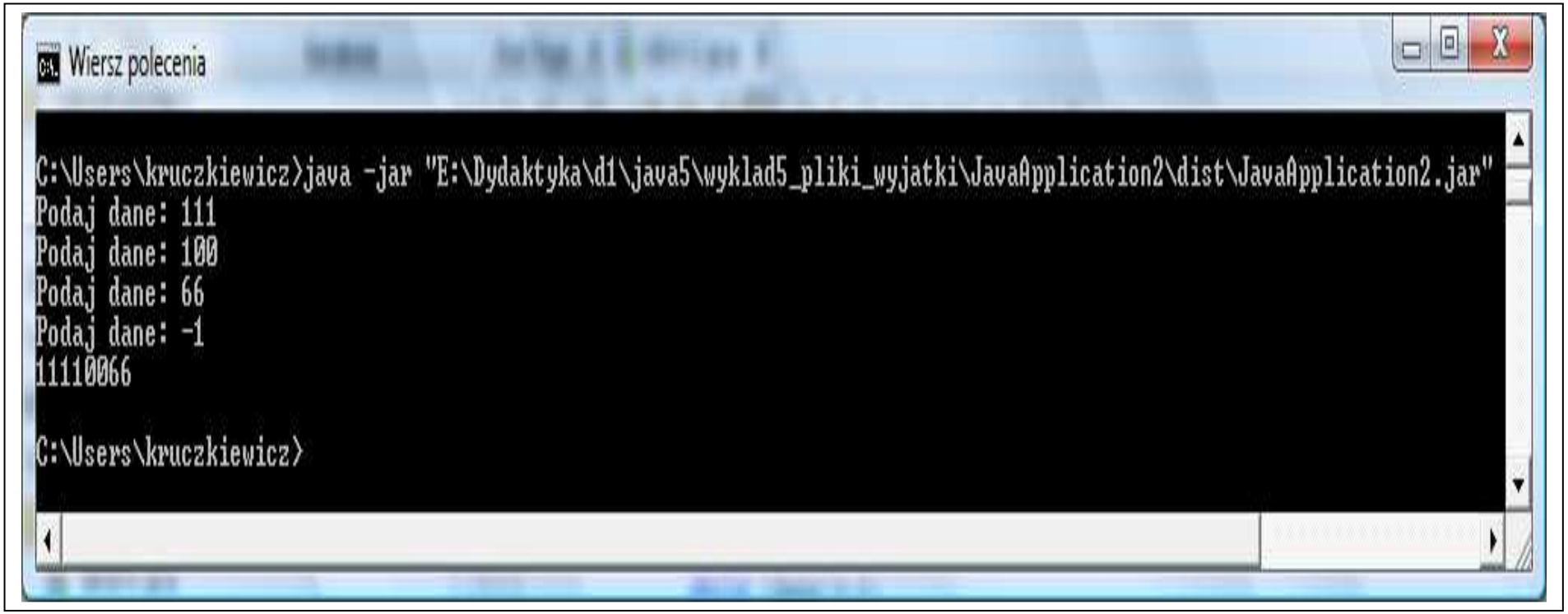
***int** read (**byte**[] cbuf, **int** off, **int** len)* – metoda, która czyta plik i zapisuje do tablicy *cbuf* od indeksu *off* liczbę *len* bajtów i zwraca przez **return** liczbę faktycznie odczytanych bajtów

9) Po zapisie i odczycie bufor należy zamknąć metodą *close()*

```
import java.io.*;
 import java.util.*;
 public class WEWY2
  {
    static byte weByte()
     {
       InputStreamReader wejscie = new InputStreamReader( System.in );
       BufferedReader bufor = new BufferedReader( wejscie );
       StringTokenizer zeton;
       try
        {
          zeton = new StringTokenizer(bufor.readLine());
          return Byte.parseByte(zeton.nextToken());
        }
       catch (IOException e)
        {
           System.err.println("Blad IO byte "+e);  return 0;  }
       catch (NumberFormatException e)
        {
           System.err.println( "Blad formatu byte "+e);   return 0;   }
     }
```

```
static void Zapiszplik2()
   {
     int dane=0;
     try
      {
        FileOutputStream plik = new FileOutputStream ("plik2.dat");
        BufferedOutputStream bufor = new BufferedOutputStream (plik);
        while (dane!=-1)
          {
             System.out.print("Podaj dane: ");
             dane=weByte();
             if (dane!=-1)   bufor.write(dane);
          }
        bufor.close();
      } catch (IOException e)
        {
          System.out.println ("Blad zapisu pliku bajtowego"+e);   }
    }
```

```
 static void Odczytajplik2()
   {
     int dane=0;
     try
      {
       FileInputStream plik = new FileInputStream ("plik2.dat");
       BufferedInputStream bufor = new BufferedInputStream (plik);
        dane=plik.read();
        while (dane!=-1)
          {
            System.out.print(dane);
            dane=bufor.read();
          }
        System.out.println();
        bufor.close();
      } catch (IOException e)
        {
           System.out.println ("Blad odczytu pliku bajtowego"+e);   }
    }

public static void main(String[] args)
   {
     Zapiszplik2();
     Odczytajplik2();
   }
}
```



Wprowadzanie danych do pliku binarnego i wyświetlanie jego zawartości w programie, interpretując znakowo zawartość jako liczb typu **int**

Interpretacja znakowa binarnego zapisu liczb typu int ()
w kodzie dwójkowym – wyświetlanie zawartości pliku w edytorze tekstowym

3.2. 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);*

Dalej podano metody strumienia danych do zapisu danych do pliku:

1. Pojedyncze bajty mogą być zapisywane do pliku za pomocą metody:

   void *write(**int** b)*

2. 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

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

Aby odczytać plik:

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

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

5) W celu buforowania bajtów pochodzących z obiektu powiązanego ze źródłem danych np. typu *FileInputStream* należy utworzyć obiekt klasy *BufferedInputStream*
*BufferedInputStream bufor = **new** BufferedInputStream (plik);*

6) 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);*

Dalej podano metody strumienia danych do odczytu danych z pliku:

1. Pojedyncze bajty mogą być odczytywane z pliku za pomocą metody:

   int *read()*

2. Całe ciągi bajtów mogą być odczytywane z pliku za pomocą metody:

3. ***int*** *read(**byte[]** cbuf, **int** off, **int** len)* – metoda, która czyta plik i zapisuje do tablicy *cbuf* od indeksu *off* liczbę *len* bajtów i zwraca przez **return** liczbę faktycznie odczytanych bajtów

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

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

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

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

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

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

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

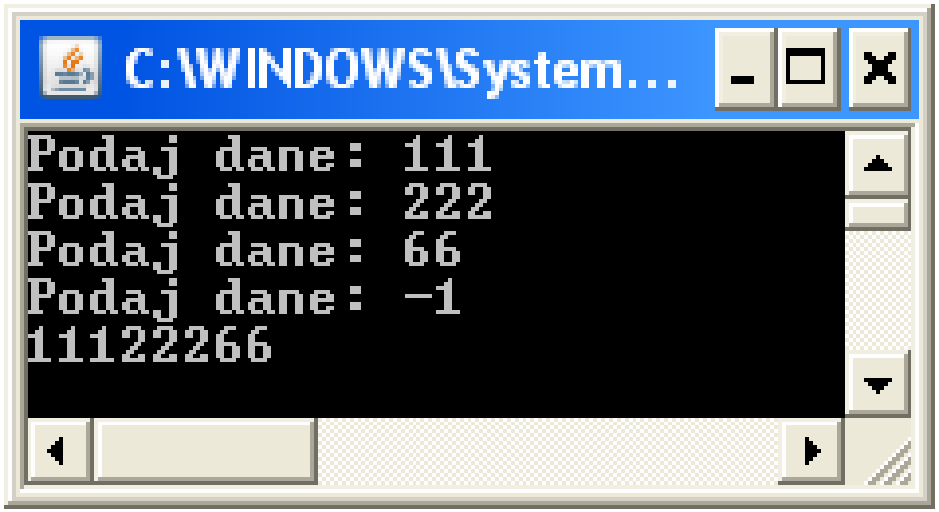
11. **short** readShort() – czyta 2 bajty z pliku i zwraca wartość typu **short**

12. Po zapisie i odczycie strumień danych należy zamknąć metodą *close()*

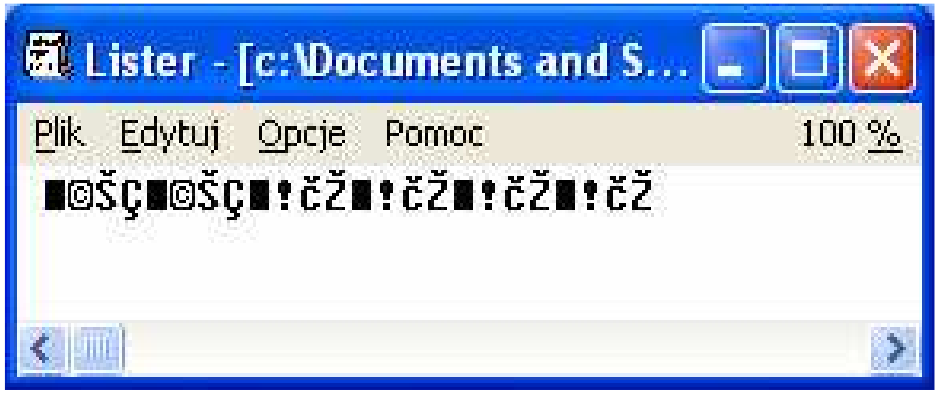
```
import java.io.*;     //zapis za pomocą writeInt i odczyt za pomocą readInt – z klawiatury są podawane 4-bajtowe wartości
import java.util.*;

  public class WEWY3_
  {
    static int weInt()
     {
       InputStreamReader wejscie = new InputStreamReader( System.in );
       BufferedReader bufor = new BufferedReader( wejscie );
       StringTokenizer zeton;
       try
          {
             zeton = new StringTokenizer(bufor.readLine());
             return Integer.parseInt(zeton.nextToken());
          }
       catch (IOException e)
          {
               System.err.println("Blad IO int "+e);   return 0;        }
       catch (NumberFormatException e)
          {
               System.err.println( "Blad formatu int "+e);   return 0;   }
      }
```

```
static void Zapiszplik3_()
   {
     int dane=0;
     try
       {
          FileOutputStream plik =  new FileOutputStream ("plik2.dat");
          BufferedOutputStream bufor = new BufferedOutputStream (plik);
          DataOutputStream dana= new DataOutputStream (bufor);
          while (dane!=-1)
              {
                 System.out.print("Podaj dane: ");
                  dane=weInt();                    //odczyt wartości 4-bajtowej
                  if (dane!=-1)
                       dana.writeInt(dane);        //zapis do pliku 4 bajtów
               }
          dana.close();
       } catch (IOException e)
              {
                    System.out.println ("Blad zapisu pliku bajtowego"+e);
               }
   }
```

```
static void Odczytajplik3_()
   {
     int dane=0;
     try
      {
         FileInputStream plik = new FileInputStream ("plik2.dat");
         BufferedInputStream bufor = new BufferedInputStream (plik);
         DataInputStream dana= new DataInputStream (bufor);
         try
          {
            while (true)
              {
                dane=dana.readInt();
                System.out.print(dane); }       //odczyt z pliku 4 bajtów
           } catch  (EOFException eof)
                {
                  bufor.close();          //zamknięcie bufora przez obsługę wyjątku od czytania poza plikiem
                 }
        System.out.println();
      } catch (IOException e)
         {  System.out.println ("Blad odczytu pliku bajtowego"+e);  }
    }
  public static void main(String[] args)
   {    Zapiszplik3_();
        Odczytajplik3_();
  } }
```



Wprowadzanie danych do pliku binarnego i wyświetlanie jego zawartości interpretując zawartość jako liczb typu **int**



Interpretacja znakowa binarnego zapisu liczb typu int ()
w kodzie dwójkowym – wyświetlanie zawartości pliku w edytorze tekstu

4. Serializacja 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 są serializowane)

- 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”;

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 ObjectOutputStream (plikobiektow);
```

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

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

Aby odczytać plik:

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

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

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

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

3) Odczytu obiektów z strumienia należy wykonać za pomocą metody

```
Object readObject()
```

Dalej podano część metod strumienia obiektów do zapisu różnych danych do pliku:

1) Pojedyncze bajty mogą być zapisywane do pliku za pomocą metody:

   void write(**int** b)

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

   void write(**byte**[]cbuf) – metoda, która czyta zawartosc tablicy bajtów i zapisuje ją do strumienia (buf.length bajtów)

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

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

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

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

7) **void** writeChar(**int** v) – zapisuje znak jako 2-bajtową wartość – pierwszy zapisywany jest starszy bajt (Unicode)

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

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

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

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

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

Dalej podano niektóre metody strumienia obiektów do odczytu danych z pliku:

1) Pojedyncze bajty mogą być odczytywane z pliku za pomocą metody: ***int*** *read()*

2) Całe ciągi bajtów mogą być odczytywane z pliku za pomocą metody:

int *read(**byte[]** cbuf, **int** off, **int** len)* – metoda, która czyta plik i zapisuje do tablicy *cbuf* od indeksu *off* liczbę *len* bajtów i zwraca przez **return** liczbę faktycznie odczytanych bajtów

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

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

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

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

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

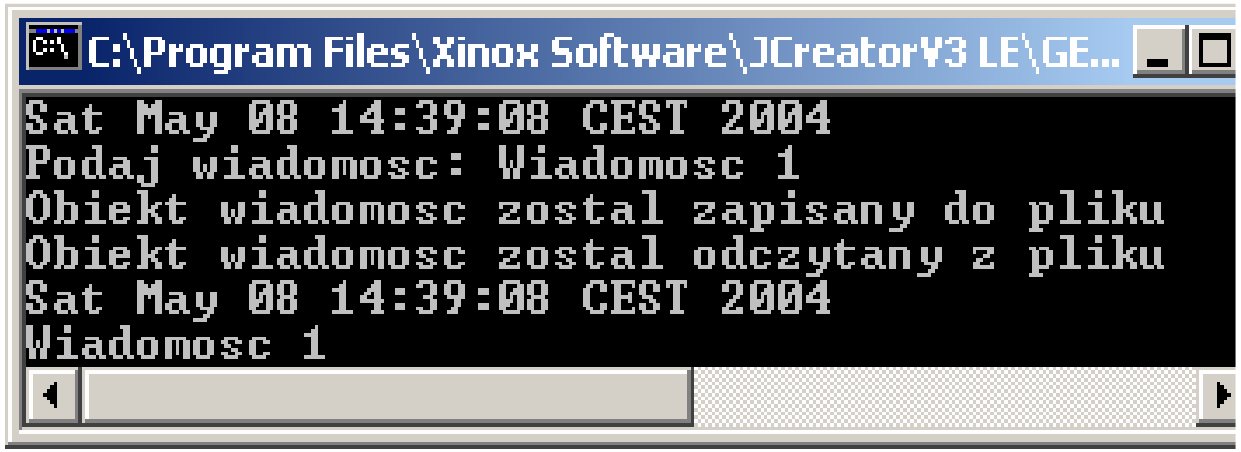
8) **int** readInt() – czyta 4 bajty z pliku i zwraca wartość typu **int**

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

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

11) Po zapisie i odczycie strumień obiektów należy zamknąć metodą *close()*

```
import java.io.*;
import java.util.*;
 class Wiadomosc implements Serializable
  { String dane;
    Date data;

  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 ""; }
    }

  public void zapiszWiadomosc()
    { data = new Date();
      System.out.println(data);
      dane =weString();}

   public void odczytajWiadomosc()
    {
      System.out.println(data);
      System.out.println(dane);}
 }
```




```
public class p6_6
 {
   static void Zapiszobiektydopliku()
    {
      Wiadomosc wiadomosc = new Wiadomosc();
      wiadomosc.zapiszWiadomosc();
      try
      { FileOutputStream plikobiektow = new FileOutputStream ("Wiadomosc.obj");
        ObjectOutputStream  strumienobiektow =
                                          new ObjectOutputStream (plikobiektow);
       strumienobiektow.writeObject(wiadomosc);
       strumienobiektow.close();
       System.out.println("Obiekt wiadomosc zostal zapisany do pliku");
       } catch (IOException e)
           { System.out.println ("Blad zapisu pliku obiektowego"+e);}
   }
```

```
static void Odczytajobiektyzpliku()
  {
    Wiadomosc wiadomosc = null;
    try
    {
      FileInputStream plikobiektow =  new FileInputStream ("Wiadomosc.obj");
      ObjectInputStream  strumienobiektow =  new ObjectInputStream (plikobiektow);
      wiadomosc = (Wiadomosc)strumienobiektow.readObject();
      System.out.println("Obiekt wiadomosc zostal odczytany z pliku");
      if  ( wiadomosc != null)
          wiadomosc.odczytajWiadomosc();
      strumienobiektow.close();
     } catch (Exception e)
       {  System.out.println ("Blad odczytu pliku obiektowego"+e);
        }
  }

 public static void main(String[] args)
  {
    Zapiszobiektydopliku();
    Odczytajobiektyzpliku();
  }
}
```

5. Otwieranie strumieni poprzez sieć

Obiekt URL (*Uniform Resource Lokator*) – obiekt typu URL reprezentuje adres zasobu w sieci www

Konstruktory klasy URL

1) URL strona= **new** URL("http://tycho.usno.navy.mil/cgi-bin/timer.pl");
2) URL strona1= **new** URL("http://tycho.usno.navy.mil");
   URL strona= **new** URL(strona1,"/cgi-bin/timer.pl");
3) URL strona= **new** URL("http", "tycho.usno.navy.mil", "/cgi-bin/timer.pl");
4) URL strona= **new** URL("http", "tycho.usno.navy.mil", 80 ,"/cgi-bin/timer.pl");

Procedura otwierania strumieni poprzez sieć:

1) Należy utworzyć obiekt *URL* z adresem zasobu www

```
URL strona = new URL("http://tycho.usno.navy.mil/cgi-bin/timer.pl");
```

2) Należy utworzyć obiekt *URLConnection,* który ładuje URL i tworzy połączenie do miejsca, w którym znajduje się dokument

```
URLConnection polaczenie = strona.openConnection();
polaczenie.connect();
```

3) Należy za pomocą metody *getInputStream* obiektu typu *URLConnection* utworzyć obiekt typu *InputStreamReader*, umożliwiający odczytanie strumienia danych z zasobu www wskazanego przez URL

```
InputStreamReader wejscie;
wejscie=new InputStreamReader(polaczenie.getInputStream());
```

4) Należy utworzyć obiekt typu *BufferedReader* buforujący odczyt ze strumienia typu *InputStreamReader*

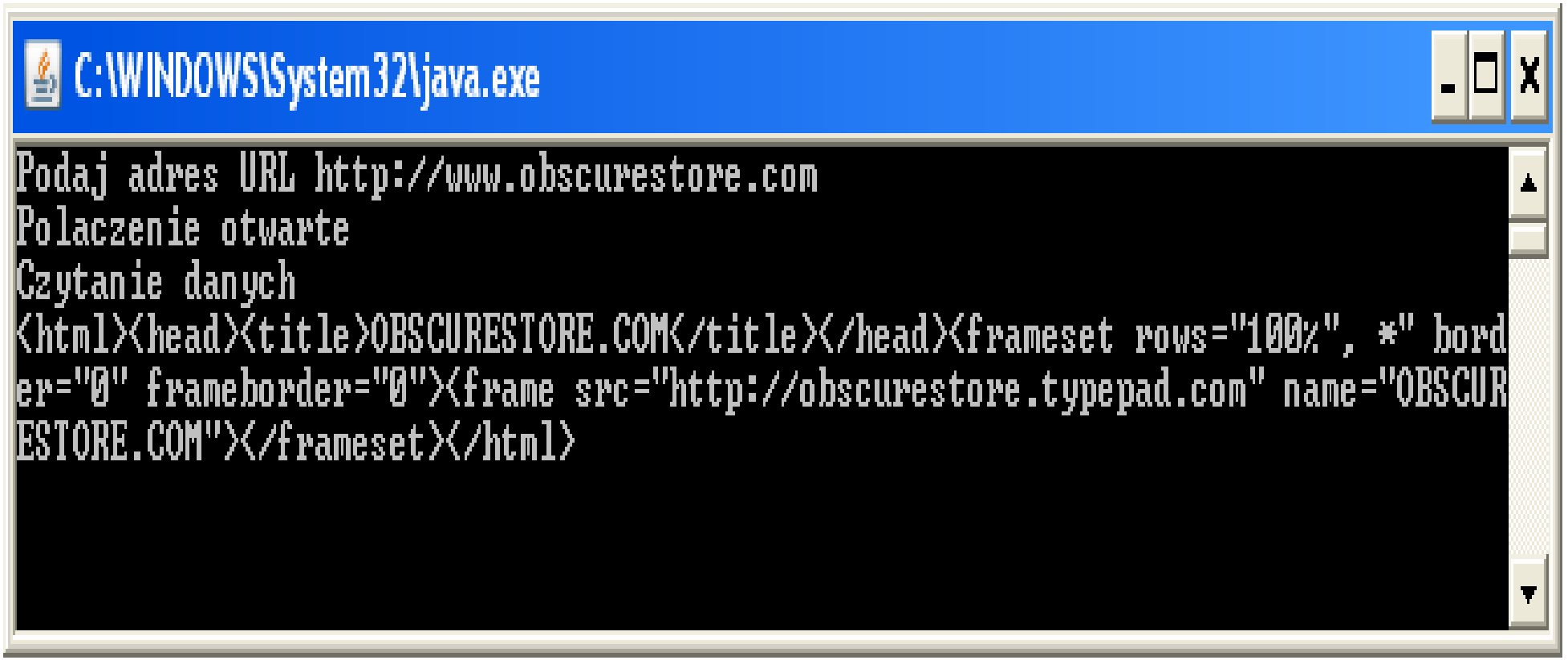
```
BufferedReader bufor;
 bufor = new BufferedReader(wejscie);
```

5) Należy przechować dane odczytane z zasobu www (*dane=bufor.readLine()*;) np. za pomocą metody *append()* obiektu typu *StringBuffer*

```
StringBuffer strbufor = new StringBuffer();
 …..
dane=bufor.readLine();
strbufor.append(dane + "\n");
```

```
import java.net.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;

public class WEWY6
 {
   static URL strona;
   static String weString()
    { InputStreamReader wejscie = new InputStreamReader( System.in );
      BufferedReader bufor = new BufferedReader( wejscie );
      try
        { return bufor.readLine(); }
      catch (IOException e)
       { System.err.println("Blad IO String");  return ""; }
    }
public static void main(String[] args)
    {   String pom;
          try
          {      System.out.print(“Podaj adres URL ”);
                 pom = weString();
                  strona= new URL (pom);
                 Odczytajplik6();
          } catch(MalformedURLException e)
            {  System.out.println("Zly URL: "); }
 }
```

```
static void Odczytajplik6()
   {
      String dane="0";
      URLConnection polaczenie = null;
      InputStreamReader wejscie;
      BufferedReader bufor;
      StringBuffer strbufor = new StringBuffer();
      try
       {
          polaczenie = strona.openConnection();
          polaczenie.connect();
         System.out.println("Polaczenie otwarte");
          wejscie = new InputStreamReader(polaczenie.getInputStream());
         bufor = new BufferedReader(wejscie);
         System.out.println("Czytanie danych");
         while ((dane=bufor.readLine())!=null)
             strbufor.append(dane + "\n");
        System.out.println(strbufor.toString());
       } catch(IOException e)
         {
            System.out.println("Blad wejscia/wyjścia"+e);        }
   }
```


C:\WINDOWS\System32\java.exe
Podaj adres URL http://www.obscurestore.com
Polaczenie otwarte
Czytanie danych
<html><head><title>OBSCURESTORE.COM</title></head><frameset rows="100%", *" bord
er="0" frameborder="0"><frame src="http://obscurestore.typepad.com" name="OBSCUR
ESTORE.COM"></frameset></html>