I/O (STRUMIENIE, PLIKI, ...)

ZAGADNIENIA:

- pakiet java.io,
- strumienie bajtowe,
- strumienie znakowe,
- strumienie binarne, serializacja i kompresja
- narzędzie jar.

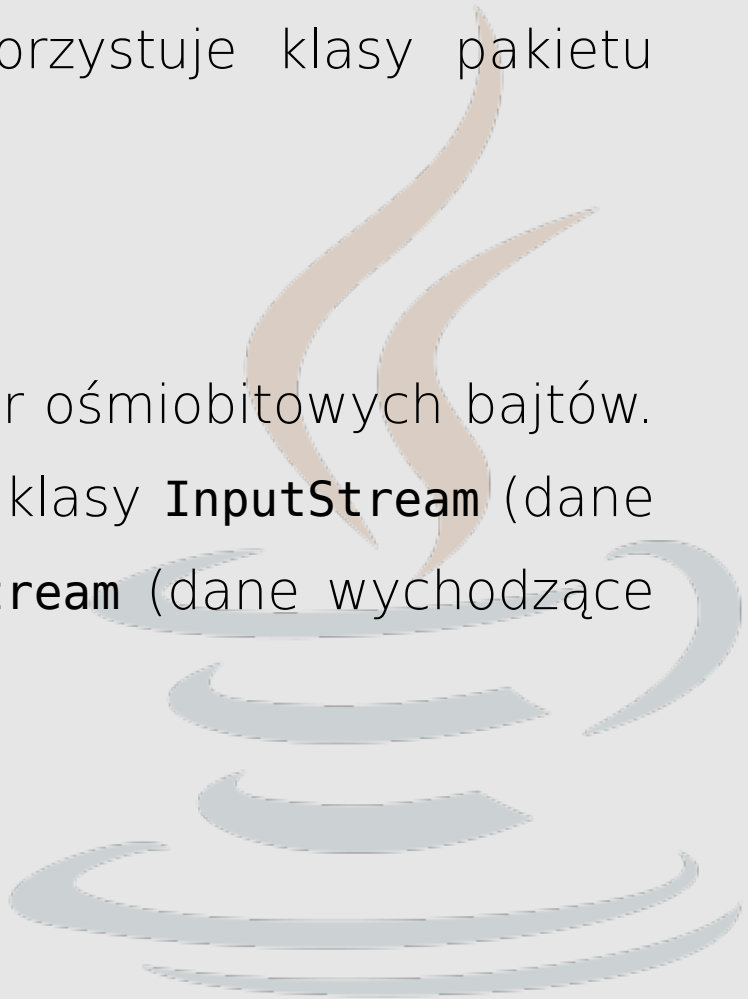
MATERIAŁY:

http://docs.oracle.com/javase/tutorial/essential/io/

STRUMIENIE BAJTOWE

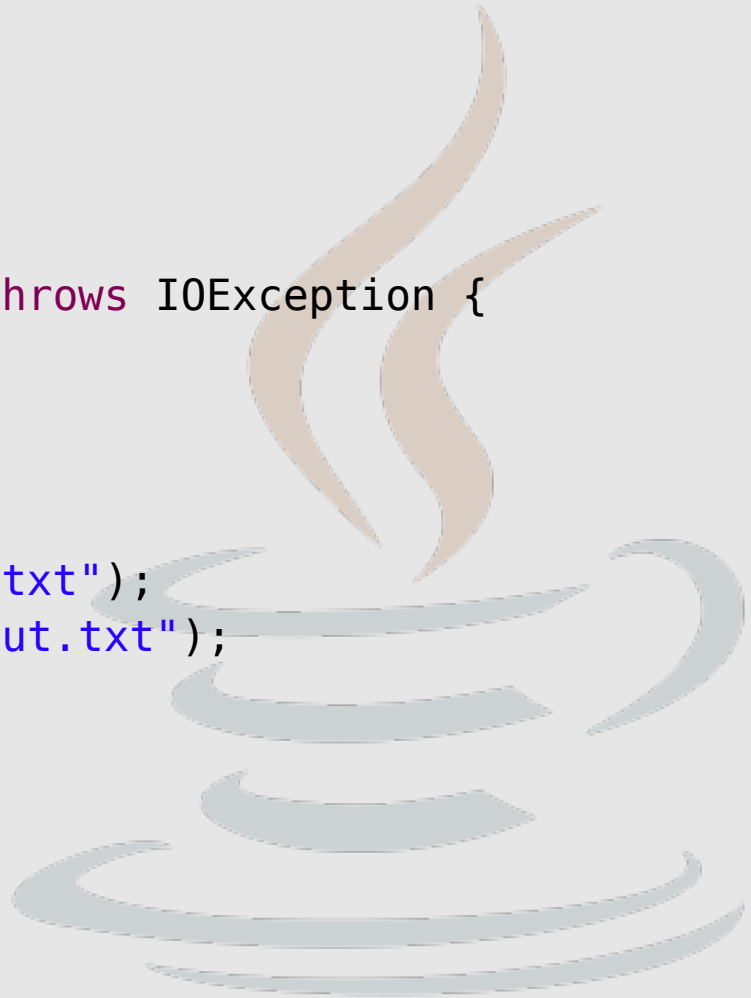
Większość operacji wejścia/wyjścia wykorzystuje klasy pakietu `java.io`.

Strumienie bajtowe traktują dane jak zbiór ośmiobitowych bajtów. Wszystke strumienie bajtowe rozszerzają klasy `InputStream` (dane przychodzące do programu) lub `OutputStream` (dane wychodzące z programu).



STRUMIENIE BAJTOWE

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
```



STRUMIENIE BAJTOWE

```
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Strumienie zawsze należy zamykać!

Strumienie bajtowe reprezentują "niskopoziomowy" dostęp do danych. Dlatego w konkretnych sytuacjach warto je zastąpić przez bardziej specjalistyczne rodzaje strumieni.

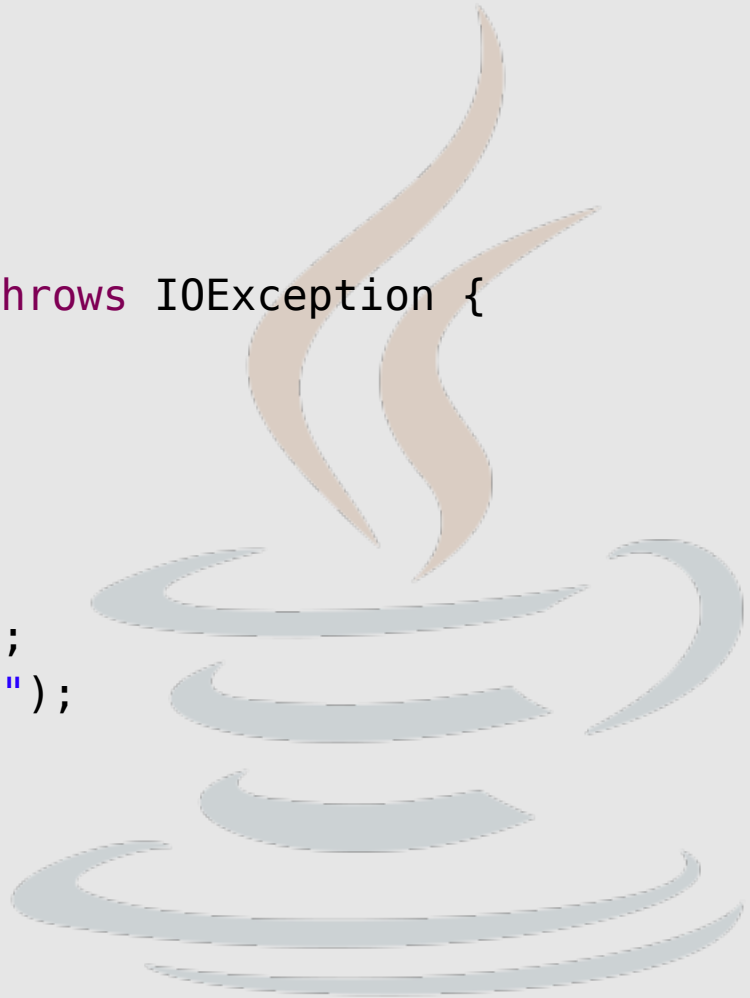
STRUMIENIE ZNAKOWE

Strumienie znakowe automatycznie konwertują dane tekstowe do formatu Unicode (stosowanego natywnie w Javie). Konwersja jest dokonywana w oparciu o ustawienia regionalne komputera, na którym uruchomiono JVM (Wirtualną Maszynę Javy), lub jest sterowana "ręcznie" przez programistę.

Strumienie znakowe rozszerzają klasy **`Reader`** (dane przychodzące do programu) lub **`Writer`** (dane wychodzące z programu).

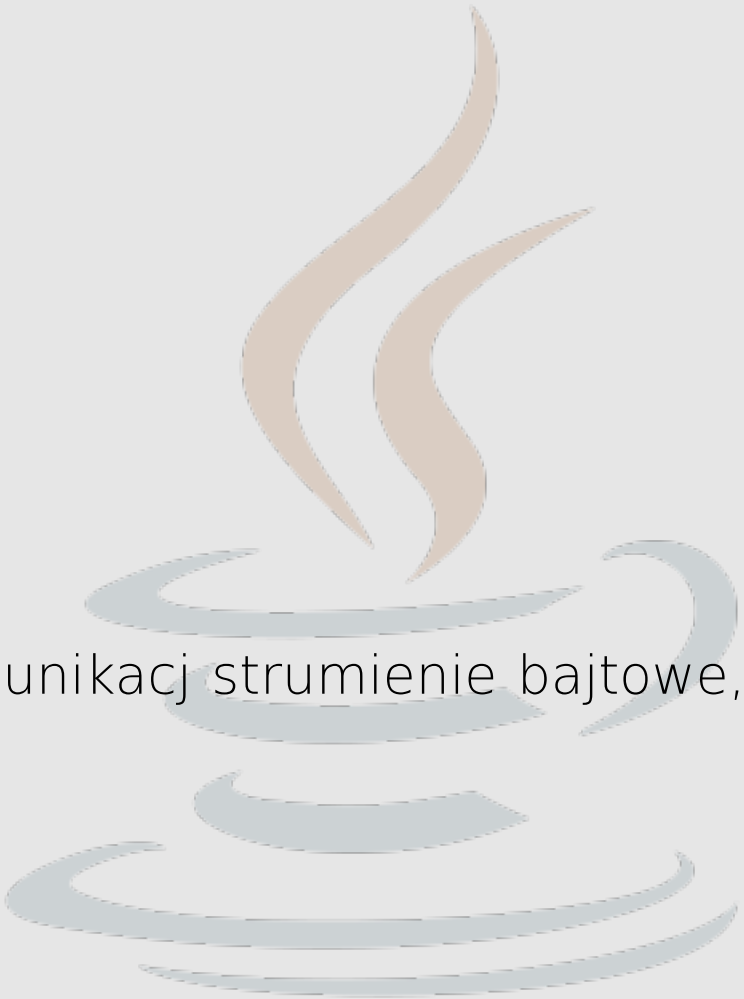
STRUMIENIE ZNAKOWE

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
    public static void main(String[] args) throws IOException {

        FileReader in = null;
        FileWriter out = null;

        try {
            in = new FileReader("input.txt");
            out = new FileWriter("output.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
```



STRUMIENIE ZNAKOWE

```
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

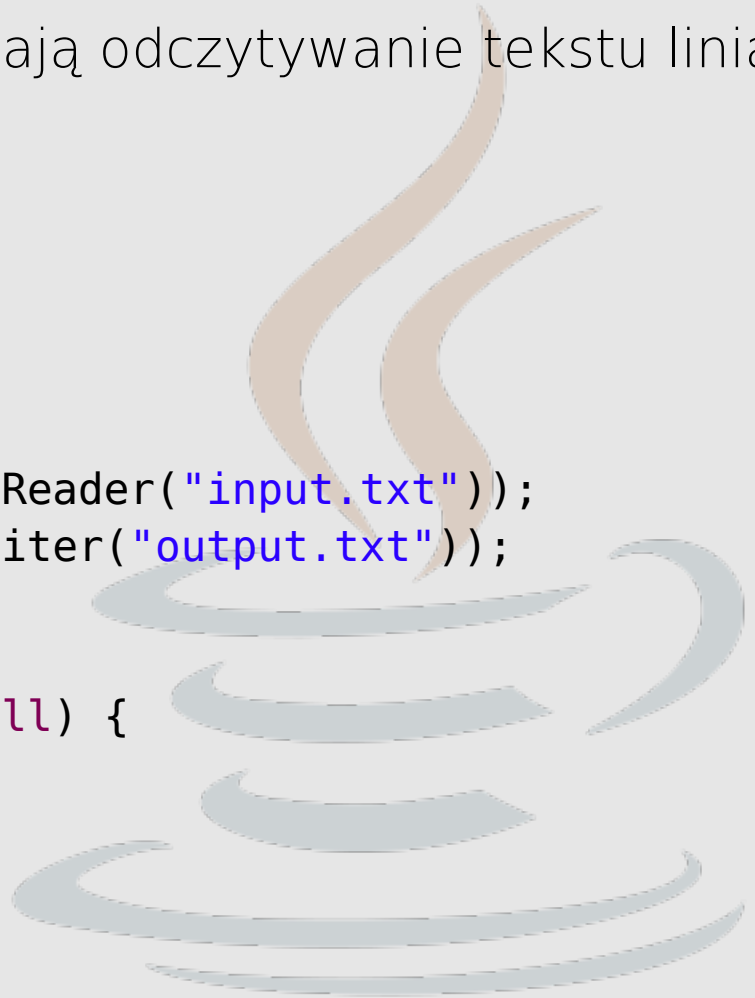


Strumienie znakowe wykorzystuję do komunikacj strumienie bajtowe, a same zajmują się konwersją danych.

STRUMIENIE BUFOROWANE

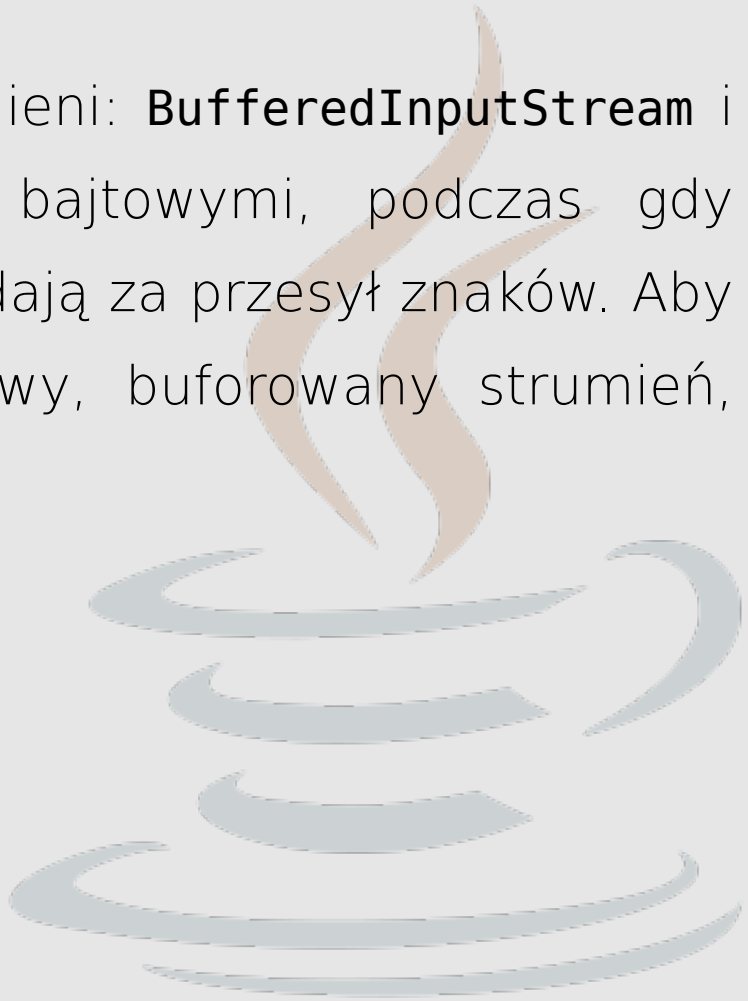
Strumienie znakowe buforowane umożliwiają odczytywanie tekstu linia po linii:

```
BufferedReader in = null;
PrintWriter out = null;

try {
    in = new BufferedReader(new FileReader("input.txt"));
    out = new PrintWriter(new FileWriter("output.txt"));

    String l;
    while ((l = in.readLine()) != null) {
        out.println(l);
    }
} catch {... }
```



STRUMIENIE BUFOROWANE

Istnieją cztery klasy buforowanych strumieni: **`BufferedInputStream`** i **`BufferedOutputStream`** są strumieniami bajtowymi, podczas gdy **`BufferedReader`** i **`BufferedWriter`** odpowiadają za przesył znaków. Aby wymusić zapis danych poprzez wyjściowy, buforowany strumień, można użyć metody **`flush()`**.



SKANOWANIE

`Scanner` pozwala na przetwarzanie tokenów (domyślnie rozdzielonych przez `Character.isWhitespace(char c)`):

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

public class ScanXan {
    public static void main(String[] args) throws IOException {

        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(
                                        new FileReader("input.txt")));
            while (s.hasNext()) {
                System.out.println(s.next());
            }
```

SKANOWANIE

```
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

Obiekt należy zamknąć ze względu na strumień, z którym jest związany. Aby zmienić zachowanie obiektu Scanner, można skorzystać z metody: **`useDelimiter()`**. Przykładowo **`s.useDelimiter(",\\s*");`** zmienia znak rozdzielający na przecinek po którym następuje dowolna liczba "białych spacji".

FORMATOWANIE

Wyjściowe strumienie znakowe umożliwiają podstawowe formatowanie danych za pomocą kilku odmian metody **print()** i **format()**.

```
double d = 2.0;
double s = Math.sqrt(2.0);
System.out.println("Pierwiastek z  " + d + " to " + s + ".");
Pierwiastek z  2.0 to 1.4142135623730951

System.out.format("Pierwiastek z  %f to %.4f\n", d, s);
Pierwiastek z  2,000000 to 1,4142

System.out.format(Locale.US, "Pierwiastek z  %.1f to %.4f\n", d, s);
System.out.printf(Locale.US, "Pierwiastek z  %.1f to %.4f\n", d, s);
Pierwiastek z  2.0 to 1.4142
```

Opis wszystkich możliwości formatowania jest opisany w dokumentacji klasy **java.util.Formatter**.

METODY WIELOARGUMENTOWE

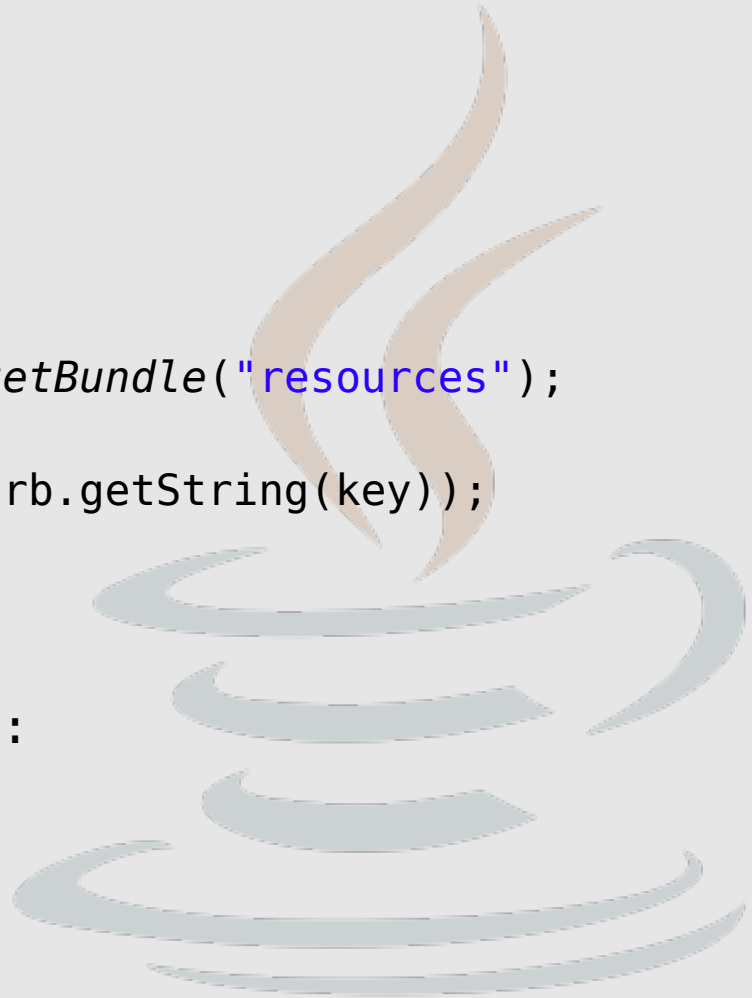
```
public static void multiint(int... ints){
    for (int i=0; i<ints.length; i++)
        System.out.println(ints[i]);
    System.out.println();
    for(int i: ints)
        System.out.println(i);
}


public static void main(String[] args){
    multiint(123,34,65,76,44,11,0);
    multiint();
    multiint(12, 28);
}
```

ZASOBY I LOKALIZACJA

```
import java.util.Locale;
import java.util.ResourceBundle;

public class LocalizationExample {

    public static void main(String[] args){
        ResourceBundle rb = ResourceBundle.getBundle("resources");
        for(String key: rb.keySet())
            System.out.println(key + ": " + rb.getString(key));
    }
}
```

Przykładowy plik **resources_pl.properties:**

```
KeyHello=witaj
KeyWorld=\u015bwiat
KeyKey=klucz
```



ZASOBY I LOKALIZACJA

Statuczna metoda `getBundle("resources")` jest równoważna wywołaniu

```
getBundle("resources", Locale.getDefault(), this.getClass().getClassLoader()).
```

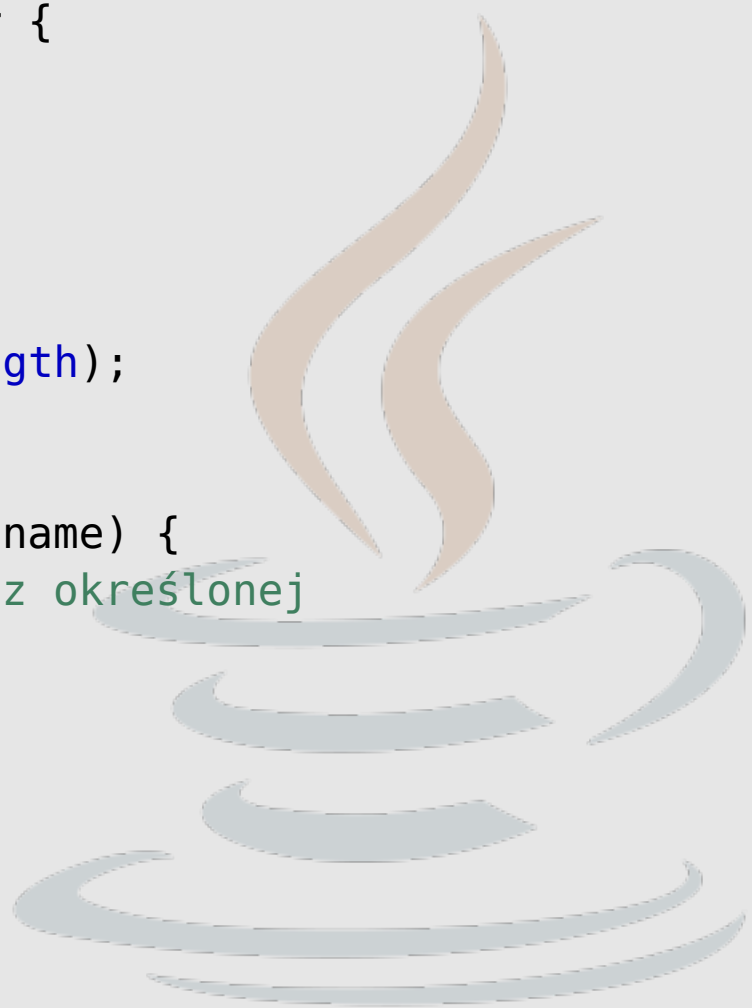
i za pomocą bieżącego **ClassLoader**a poszukuje pliku o nazwie:

```
baseName +"_"+ language +"_"+ script +"_"+ country +"_"+ variant +".properties"
```

Konkretna nazwa pliku jest ustalana na podstawie ustawien regionalnych systemu operacyjnego (**Locale.getDefault()**), np. **resources_en_US_WINDOWS_VISTA.properties**. Metoda ta wczytuje pary (klucz,wartość). Dzięki temu można łatwo dostosować komunikaty, używane przez program do użytkownika.

CLASS LOADER

```
class NetworkClassLoader extends ClassLoader {
    String host;
    int port;

    public Class findClass(String name) {
        byte[] b = loadClassData(name);
        return defineClass(name, b, 0, b.length);
        }

        private byte[] loadClassData(String name) {
            // wczytywanie bytecode'u klasy z określonej
            // lokalizacji sieciowej
            . . .
        }
    }
}
```



STRUMIENIE BINARNE

Strumienie binarne pozwalają efektywniej zarządzać zasobami. Istnieją dwa podstawowe rodzaje strumieni:

- strumienie danych: **DataInputStream** i **DataOutputStream**:

```
DataOutputStream dos = new DataOutputStream(System.out);
dos.writeDouble(123.12);
dos.writeUTF("Grzegrz\u00f3\u0142ka");
dos.writeInt(12345);
dos.close();
```

- strumienie obiektowe: **ObjectInputStream** i **ObjectOutputStream**:

```
ObjectOutputStream oos = new ObjectOutputStream(System.out);
oos.writeObject("Grzegrz\u00f3\u0142ka");
oos.close();
```

SERIALIZACJA

Podstawowym zastosowaniem strumieni obiektowych jest serializacja. Klasa wspierająca serializację musi implementować interfejs **`Serializable`**. Jeśli obiekty tej klasy wymagają specjalnego traktowania podczas serializacji należy zaimplementować metody:

```
private void writeObject(java.io.ObjectOutputStream out)
                                                    throws IOException;

private void readObject(java.io.ObjectInputStream in)
                                  throws IOException, ClassNotFoundException;
```

SERIALIZACJA

```
public class SerialisationTest implements Serializable{
    public int id;
    public String name;

    public SerialisationTest(int i, String s){
        this.id = i;
        this.name = s;
    }

    public static void main(String[] args) throws FileNotFoundException,
                                        IOException, ClassNotFoundException{

        SerialisationTest st1 = new SerialisationTest(7, "Ala");

        ObjectOutputStream oos = new ObjectOutputStream(
                                    new FileOutputStream("output_object"));
        oos.writeObject(st1);
        oos.close();
```

SERIALIZACJA

```
        SerialisationTest st2;

        ObjectInputStream ois = new ObjectInputStream(new
                                        FileInputStream("output_object"));
        st2 = (SerialisationTest)ois.readObject();
        ois.close();
        System.out.println(st2.id + "\t" + st2.name);
    }
}
```

Dla obiektów typu JavaBeans istnieje także możliwość serializacji tekstowej (do plików w formacie XML) z wykorzystaniem klas `XMLEncoder` i `XMLDecoder`.

STRUMIENIE KOMPRESUJĄCE

Strumienie kompresujące służą do bsługi formatów gzip, zip, jar.

```
Scanner sc = new Scanner(System.in);
GZIPOutputStream gos = new GZIPOutputStream(
                                        new FileOutputStream(args[0]));
while(sc.hasNext()){
      String s = sc.nextLine() + "\n";
      gos.write(s.getBytes());
}
sc.close();
gos.close();
```

GZIPOutputStream nie zapisuje danych do pliku tylko je przetwarza (kompresuje). Do zapisu wykorzystuje on strumień, którego instancje dostaje w konstruktorze (tutaj **FileOutputStream**).

STRUMIENIE KOMPRESUJĄCE

Format ZIP obsługuje archiwa złożone z wielu plików:

```
ZipOutputStream zos = new ZipOutputStream(
                                        new FileOutputStream("plik.zip"));
for(int i=0; i<5; i++){
    ZipEntry ze = new ZipEntry("plik" + i);
    zos.putNextEntry(ze);
    for(int j=0; j< 1000; j++){
        zos.write("Ala ma kota".getBytes());
    }
    zos.closeEntry();
}
zos.close();
```

ZipEntry to znacznik informujący, że następujące po nim dane należą do wskazanego pliku.

ARCHIWA JAR

Java wyróżnia także szczególny rodzaj archiwym ZIP: JAR (JarOutputStream, JarInputStream). Archiwa JAR zawierają pliki klas wraz z dodatkowymi zasobami potrzebnymi do działania aplikacji. Podstawowe zalety dystrybucji programów w postaci plików jar to:

- bezpieczeństwo: archiwa mogą być cyfrowo podpisywane,
- kompresja: skrócenie czasu ładowania apletu lub aplikacji,
- zarządzanie zawartością archiwów z poziomu języka Java,
- zarządzanie wersjami na poziomie pakietów oraz archiwów (Package Sealing, Package Versioning),
- przenośność.

JAR

Archiwum jar tworzy sie używając komendy jar, np:

```
jar cf archiwum.jar klasa1.class klasa2.class ...
```

Użyte opcje:

- c – tworzenie pliku (create),
- `f` – zawartość archiwum zostanie zapisana do pliku archiwum.jar zamiast do standardowego wyjscia (stdout);

Inne najczęściej używane opcje:

- `m` – do archiwum zostanie dołączony plik manifest z określonej lokalizacji, np: jar cmf plik_manifest archiwum.jar *,
- `C` – zmiana katologu w trakcie działania archiwizatora, np: jar cf ImageAudio.jar -C images * -C audio *.

MANIFEST

W archiwum jar znajduje się katalog **META-INF** a w nim plik **MANIFEST.MF** zawierający dodatkowe informacje o archiwum. Przykładowa zawartość:

```
Manifest-Version: 1.0
Created-By: 1.5.0-b64 (Sun Microsystems Inc.)
Ant-Version: Apache Ant 1.6.5
Main-Class: pl.edu.uj.if.wyklady.java.Wyklad06
```

mówi, że po uruchomieniu archiwum wykonana zostanie metoda **main(String[] args)** zawarta w klasie **Wyklad06** znajdującej się w pakiecie **pl.edu.uj.if.wyklady.java**.

Uruchomienie pliku jar:

```
java -jar archiwum.jar
```

ĆWICZENIA

- Proszę napisać program, który dla wskazanego pliku tekstowego zlicza ilość wystąpienia wszystkich znaków ('a' – 20 razy, 'b' – 2 razy, 'c' – 1 raz, itd.), ilość wystąpienia wyrazów (“java” - 4 razy, “informatyka” - raz), a ponadto podaje ilość lini oraz zdań.

- Proszę napisać strumienie (**InputCesarCipher** i **OutputCesarCipher**), które obsługują (de)szyfrują wskazany plik tekstowy używając szyfru Cesara. Strumieie powinien działać analogicznie do **GZIPInputStream** i **GZIPOutputStream**.

DZIĘKUJĘ ZA UWAGĘ