

semestr zimowy 2020/2021

Michał Cieśla (michal.ciesla@uj.edu.pl)

https://zfs.fais.uj.edu.pl/mc\_jezyk\_java

Konsultacje: piątek 10-12, pokój D-2-47

## JAVA





Java is one of several Indonesian islands that grow coffee. Originally, the term "Java coffee" or "Kopi jaw" identified the dark Arabica coffee specific to Java. In American slang, "Java" came to mean coffee in general.

By Tamasin Wedgwood, eHow Contributor

Read more: Why Is Coffee Called Java? | eHow.com

# JĘZYK JAVA

### ZAGADNIENIA:

- podstawy;
- przegląd biblioteki standardowej;
- Java w zastosowaniach.

### LITERATURA:

http://www.oracle.com/technetwork/java/javase/documentation/index.html,

Coursera, iTunesU, itp.

http://www.google.com/,

Bruce Eckel, Thinking in Java.

## PREREKWIZYTY

### JDK – Java Development Kit

http://www.oracle.com/technetwork/java/javase/downloads/index.html

### JRE – Java Runtime Environment

http://www.java.com

lub

http://www.oracle.com/technetwork/java/javase/downloads/index.html

## HELLO WORLD

### HelloWorld.java

URUCHOMIENIE:

java HelloWorld

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}

KOMPILACJA:
javac HelloWorld.java HelloWorld.class
```

## PODSTAWY: TYPY DANYCH

#### PODSTAWY JĘZYKA JAVA:

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

#### PRYMITYWNE TYPY DANYCH:

- byte (8-bit), short (16-bit), int (32-bit), long (64-bit)
- **float** (32-bit), **double** (64-bit),
- boolean (1-bit) flaga
- char (16-bit) znak w unikodzie, np. \u015b

#### **OBIEKTOWE TYPY DANYCH:**

• String, PrintStream, ... (wszystko inne).

## PRZEPŁYW STEROWANIA

### INSTRUKCJE WARUNKOWE

```
• if...then...else...
if(a>0){      // nawiasy klamrowe są wymagane jeśli w bloku
   return 1; // znajduje się więcej niż jedna instrukcja
}else{      // tak samo jak w C/C++
   return -1;
  switch
switch (a){
   case 1: makeSomething(a);
           break;
   case 2: makeSomethingElse(a);
   default: a++;
```

## PRZEPŁYW STEROWANIA

```
PETLE:
• for.
for(i=0; i<args.length; i++)</pre>
    System.out.printf(Locale.US, "%.2f\n", args[i]);
  while
String s="Ala";
while(s.length()<20)</pre>
    S = " " + S;
• do...while
do{
    String s = getValue();
}while(s!=null);
```

## PRZEPŁYW STEROWANIA

#### ZABURZENIA PRZEPŁYWU:

break, continue, return

```
String[] names = getNames();
for(int i=0; i<names.length; i++){</pre>
    if (names[i].equals("JAVA")){
                                  // znalezlismy i nie musimy dalej szukać
        found = true;
        break:
File[] f = dir.listFiles();
for(int i=0; i<f.length; i++){</pre>
    if (f[i].isDirectory())
                                   // chemy wypisac tylko nazwy plikow
        continue:
    System.out.println(f[i].getName());
```

### <u>SquareRoot.java</u>

```
public class SquareRoot {
   public static final double precision = 1.0e-5;
   public static double calculateSquareRoot(double x){
       double guess = 1.0;
       do{    // pierwiastek jest pomiedzy guess a x/guess
           guess = (guess + x/guess)/2.0;
       \ while ( (guess*guess/x < 1.0-precision) ||
                                          (guess*guess/x>1.0+precision));
       return guess;
```

SquareRoot.java (c.d)

### ParabolaRoots.java

```
public class ParabolaRoots {
   public static double[] getRoots(double a, double b, double c){
      double[] roots = new double[3];
      double delta = b*b-4*a*c;
      if (delta<0){
        roots[0] = 0;
      }else{
        roots[1] = (-b+Math.sqrt(delta))/(2*a);
        roots[2] = (-b-Math.sqrt(delta))/(2*a);
      }
      return roots;
}</pre>
```

<u>ParabolaRoots.java</u> (c.d)

```
public static void main(String[] args){
       double a=Double.parseDouble(args[0]);
       if(a==0)
           System.out.println("Nieprawid\u0142owe dane");
       double b=Double.parseDouble(args[1]);
       double c=Double.parseDouble(args[2]);
       double[] results = getRoots(a, b, c);
       String[] sa = {"Liczba rzeczywistych pierwiastk\u00f3w: ",
                       x1 = x1 = x2 = x3;
       for(int i=0; i<results[0]+1; i++)</pre>
           System.out.println(sa[i] + results[i]);
   } // koniec metody
} // koniec klasv
URUCHOMIENIE (po skompilowaniu):
java ParabolaRoots 1 2 -2
```

# ŚRODOWISKA DEWELOPERSKIE

NETBEANS

http://netbeans.org/

• ECLIPSE

http://www.eclipse.org/

INTELLIJ IDEA

http://www.jetbrains.com/idea/



# ĆWICZENIA

- Proszę napisać program obliczający pierwiastek n-tego stopnia z zadanej liczby (n i liczba są argumentami wywołania programu);
- proszę napisać program obliczający średnią geometryczną liczb podanych jako argumenty wywołania programu;
- proszę napisać program znajdujący największy wspólny dzielnik dwóch liczb podanych jako argumenty wywołania programu.

# DZIĘKUJĘ ZA UWAGĘ

# KLASY, INTERFEJSY, ITP

### ZAGADNIENIA:

- Klasy, modyfikatory dostępu, pakiety.
- Zmienne i metody statyczne.
- Klasy abstrakcyjne, dziedziczenie.
- Interfejsy.
- Komentarze i javadoc,

# DWIE KLASY

```
public class Klasa1{
    public void metodal(){
public class Klasa2{
    public void metoda2(){
        Klasa1 k1;
        k1 = new Klasa1(...);
        k1.metoda1();
```



```
package pakiet.podpakiet;
public class Klasa {
    public int publiczny; // public - dostępny wszędzie
    protected int chroniony; // dostępny tylko w danej klasie, klasach
                             // potomnych i klasach z tego samego pakietu
   int zwykly; // dostępny tylko w danej klasie i klasach z tego samego
               // pakietu
    private int prywatny; // dostępny tylko dla metod tej klasy
    protected Klasa(){
       // konstruktor moze nic nie robic, moze go nie byc,
       // nie musi byc publiczny
    public Klasa(int a, int b, int c, int d){
        this.publiczny = a;
        this.prywatny = b;
        this.chroniony = c;
       this.zwykly = d;
```

```
public void set(){
   this.publiczny = 7;
   this.prywatny = 13;
    this.chroniony = 27;
   this.zwykly = 11;
public void print(){
    System.out.println("publiczny: " + this.publiczny);
    System.out.println("prywatny: " + this.prywatny);
    System.out.println("chroniony: " + this.chroniony);
    System.out.println("zwykly: " + this.zwykly);
    System.out.println();
```

#### **Access Levels**

Modifier	Class	Package	Subclass	World
public	Y	Υ	Υ	Y
protected	Y	Υ	Υ	N
no modifier	Y	Υ	N	N
private	Y	N	N	N

```
public static void main(String args[]){
    Klasa k1 = new Klasa();
    k1.print();
    k1.set();
    k1.print();
    Klasa k2 = new Klasa(1,2,3,4);
    k2.print();
}
```

### URUCHOMIENIE:

java pakiet.podpakiet.Klasa

plik Klasa.class musi sie znajdowac w podkatalogu
./pakiet/podpakiet/

### PAKIETY

Klasy można grupować w pakiety. Nazwa pakietu, do którego należy klasa jest podana w pliku definiującym klasę:

```
package pakiet.podpakiet;
```

Jeśli chcemy użyć klasy z innego pakietu niż nasz, musimy ją uprzednio zaimportować:

```
import pakiet.podpakiet.Klasa;
```

lub

```
import pakiet.podpakiet.*;
```

hierarchia pakietów jest odwzorowana w systemie plików w hierarchie katalogów.

# KLASY - KONWENCJE

- nazwa pakietu: z <u>małej</u> litery,
- nazwa klasy: z <u>DUŻEJ</u> litery,
- nazwa atrybutu: z <u>małej</u> litery,
- nazwa metody: z <u>małej</u> litery,
- nazwa zmiennej: z <u>małej</u> litery, stosujemy przedrostki określające typ zmiennych, np. iValue, sName.
- nazwy pakietów odwrotne nazwy domenowe, np.
   pl.edu.uj.fais.java.wyklad2

## ZMIENNE I METODY STATYCZNE

```
public class Klasa1{
    public static void metodal(){
        ...
    }
}
public class Klasa2{
...
    public void metoda2(){
        Klasa1.metoda1();
    }
}
```

Atrybuty i metody <u>statyczne</u> są związane z klasą a nie z jej instancjami (obiektami). Metody statyczne nie mogą więc bezpośrednio wywoływać zwykłych metod lub korzystać ze zwykłych atrybutów, gdyż one są określone wyłącznie w kontekscie obiektów.

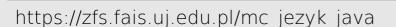
## ZMIENNE I METODY STATYCZNE

Atrybuty statyczne są inicjalizowane bezpośrednio po załadowaniu klasy przez JVM.

Atrybuty zwykłe (niestatyczne) są inicjowane w momencie utworzenia obiektu (wywołania konstruktora). Zadeklarowane a niezainicjowane atrybuty są ustawiane na **0** lub **null**.

# ZMIENNE I METODY STATYCZNE

```
public class OrderTest {
    static{
        System.out.println("static");
    public OrderTest(){
        System.out.println("constructor");
    public static void main(String[] args){
        System.out.println("main: begin");
        OrderTest o:
        System.out.println("main: middle");
        o = new OrderTest();
        System.out.println("main: end");
```



# KLASY ABSTRAKCYJNE

```
public abstract class AbstractClass {
    ...
    public abstract int doSomething();
    public int doSomethingElse(){
        ...
    }
    ...
}

public class SpecificClass extends AbstractClass{
    public int doSomething(){
        ...
    }
}
```

Klasa abstrakcyjna to zwykle klasa, której co najmniej jedna z metod jest abstrakcyjna. Nie można bezpośrednio tworzyć instancji klasy abstrakcyjnej.

## DZIEDZICZENIE

```
public class AnyClass extends AnotherClass{
    ...
}
```

Klasa może mieć tylko jednego, bezpośredniego rodzica (inaczej niż w C++). Jeśli klasa nie posiada rodzica, dziedziczy automatycznie po klasie Object (java.lang.Object). W związku z tym instancja dowolnej klasy jest obiektem (instancją klasy Object).

# INTERFEJSY

Interfejsy posiadają hierarchię dziedziczenia, jednak jest ona zupełnie niezależna od hierarchii klas.

### <u>TryAndCheck.java</u>

```
import java.io.IOException;
/**
 * Klasa umożliwiająca zgadywanie liczby, ktora wylosowal komputer
* @author Kubus Puchatek
public class TryAndCheck {
    private int number;
    /**
     * konstruktor, w nim odbywa sie losowanie liczby
     */
    public TryAndCheck(){
        this.number = (int)(Math.random()*10);
```

### <u>TryAndCheck.java</u> (c.d.)

```
/**
 * sprawdza, czy podana wartosc jest wieksza, mniejsza badz rowna
 * wylosowanej liczbie
 * @param iv
 * @return -1 gdy iv jest mniejsza, 1 gdy większa, 0 gdy rowna,
 */
public byte check(int iv){
   if (iv<this.number) return -1;
   if (iv>this.number) return +1;
   return 0;
}
```

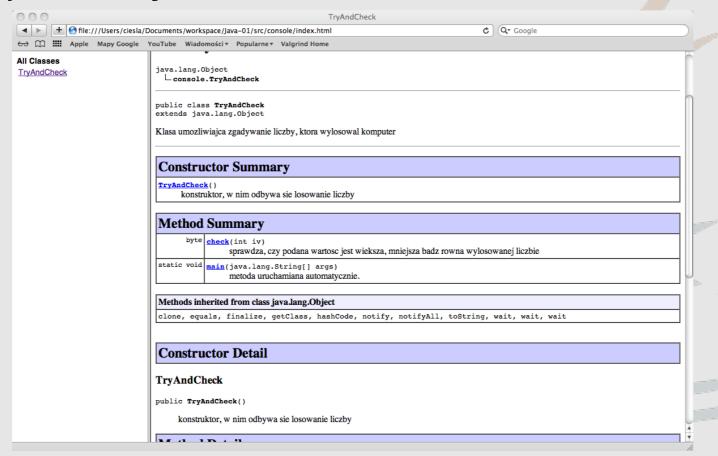
### <u>TryAndCheck.java</u> (c.d.)

```
/**
 * metoda uruchamiana automatycznie. Przeprowadza rozgrywke
 * @param args nieobslugiwane
 * @throws IOException w przypadku niepoprawnych danych
 */
public static void main(String[] args) throws IOException{
    TryAndCheck play = new TryAndCheck();
    int res;
    char c;
```

```
<u>TryAndCheck.java</u> (c.d.)
```

#### AUTOMATYCZNE GENEROWANIE DOKUMENTACJI

javadoc TryAndCheck.java



# ĆWICZENIA

- Proszę zaimplementować klasę reprezentującą liczbę zespoloną, posiadającą metody umożliwiające dodawanie, mnożenie i dzielenie takich liczb,
- Proszę zaimplementować klasę abstrakcyjną reprezentującą funkcję jednej zmiennej posiadającą metodę abstrakcyjną double getValue(double x) oraz zwykłą metodę double findMinimum(double a, double b), która znajduje lokalne minimum funkcji w przedziale [a,b]

# DZIĘKUJĘ ZA UWAGĘ

## WYJĄTKI, KOLEKCJE

#### ZAGADNIENIA:

- 1. Wyjątki,
- 2. Kolekcje,
  - vector,
  - hashtable,
  - properties,
  - Klasy Arrays i Collections.



Błędy wykonania programu są sygnalizowane z wykorzystaniem obiektów (**Throwable**). Klasa **Throwable** posiada dwie klasy potomne:

- wyjątki (Exception),
- błędy (Error).

Wśród wyjątków znajduje się jedna szczególna klasa:

RuntimeException, określająca błędy pojawiające się w trakcie działania programu, których nie można było <u>łatwo</u> przewidzieć na etapie tworzenia oprogramowania np. NullPointerException lub IndexOutOfBoundsException.

Obsługa pozostałych wyjątków jest obowiązkowa, tzn. jeżeli korzystamy z metody mogącej zwrócić wyjątek musimy wykonać jedną z dwóch czynności:

obsłużyć wyjątek za pomocą try...catch...(finally...),
 try{
 ...
}catch(FileNotFoundException ex){
 ex.printStackTrace();
 ...
}finally{
 ...
}

zadeklarować, że nasza metoda może zwrócić ten wyjątek:

```
public void aMethod() throws FileNotFoundException{...}
```

```
Obsługa wielu wyjątków:
try{
}catch(FileNotFoundException ex){
}catch(NullPointerException ex){
}catch(IOException ex){
}finally{
Od Javy 7 możliwe łączenie obsługi:
catch(FileNotFoundException | NullPointerException ex){
```

```
Kolejność obsługi:
   String s=null;
   try{
        s.split(" "); // tutaj jest rzucany NullPointerException
    }catch(NullPointerException ex){
        System.out.println("NullPointerException");
    }catch(Exception ex){
        System.out.println("Exception");
    }finally{
        System.out.println("Finally");
Co wypisze ten fragment kodu?
NullPointerException
Finally
```

## BŁĘDY

Błędy informują o nieprawidłowym działaniu Wirtualnej Maszyny Javy (np. **OutOfMemoryError**). Aplikacja nie powinna próbować ich obsługiwać, gdyż zwykle nie są one wynikiem nieprawidłowego jej działania.

## KOLEKCJE

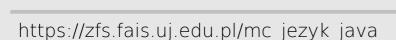
Najpopularniejszą kolekcją (zbiorem) danych jest tablica. Jednak w wielu zastosowaniach przydatne są inne struktury danych jak listy, zbiory, mapy, tablice haszujące itp. Standardowa biblioteka Javy zawiera implementacje najpopularniejszych kolekcji w pakiecie java.util. Ich podstawowa funkcjonalność jest zdefinowana w interfejsie java.util.Collection.

Dokumentacja:

http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

## java.util.Collection

```
Przykładowe metody:
boolean add(Object o)
void
        clear()
boolean contains(Object o)
boolean
        isEmpty()
Iterator iterator()
       remove(Object o)
boolean
int
        size()
Object[] toArray()
```



## java.util.Collection

Przykładowe klasy

rozszerzające:

ArrayList

HashSet

LinkedList

Stack

Vector

**PriorityQueue** 

TreeSet

. . .

Przykładowe interfejsy

rozszerzające:

List

SortedSet

Set

Queue

Deque

. . .

## PRZYKŁADOWE KOLEKCJE

**Vector** – w rzeczywistości to dynamiczna tablica, której rozmiar jest automatycznie dostosowywany do ilości danych.

```
Vector v = new Vector();
v.add("Ala");
v.add(true); // dawniej v.add(new Boolean(true));
v.add(128.5);
v.add("Ola");

for(int i=0; i<v.size(); i++){
    Object o = v.get(i);
    System.out.println(o.getClass().getCanonicalName() + "\t" + o);
}</pre>
```

#### VECTOR

```
for(Enumeration e = v.elements(); e.hasMoreElements(); ){
    Object o = e.nextElement();
    System.out.println(o.getClass().getCanonicalName() + "\t" + o);
}

for(Iterator it=v.iterator(); it.hasNext(); ){
    Object o = it.next();
    System.out.println(o.getClass().getCanonicalName() + "\t" + o);
}

for(Object o: v){
    System.out.println(o.getClass().getCanonicalName() + "\t" + o);
}
```

#### VECTOR<E>

```
v = new Vector();
v.add("Ala");
v.add("Ela");
v.add("0la");
String s = (String) v.get(0);
zaleca się określenie typu elementów w wektorze
Vector<String> v1=new Vector<String>();// od v.7 nie trzeba po prawej
                                       stronie pisac typu: new Vector<>()
String s = v1.get(0); // nie trzeba rzutować
Vector może zawierać też elementy bardziej skomplikowane:
Vector<Vector<String>> v1;
```

## DYGRESJA: CLONEABLE

Implementacja interfejsu **Cloneable** informuje, że nasz obiekt wspiera klonowanie. Bazowa metoda clone jest zaimplementowana w klasie **Object**.

protected Object clone() throws CloneNotSupportedException

## DYGRESJA: CLONEABLE

Implementacja kolonowania w klasie Vector:

```
public synchronized Object clone() {
    try {
        Vector<E> v = (Vector<E>) super.clone();
        v.elementData = Arrays.copyOf(elementData, elementCount);
        v.modCount = 0;
        return v;
    } catch (CloneNotSupportedException e) {
        // this shouldn't happen, since we are Cloneable
        throw new InternalError();
    }
}
```

#### HASHTABLE

Tablica haszująca to kolekcja (mapa) zawierająca pary (klucz, wartość). Zarówno klucz jak i wartość mogą być dowolnymi obiektami.

```
public class Hashtable<K,V> extends Dictionary<K,V>
    implements Map<K,V>, Cloneable, Serializable {
```

```
Hashtable<String,BufferedImage> ht=new Hashtable<String,BufferedImage>();
File dir = new File(System.getProperty("user.dir"));
File[] files = dir.listFiles();
for(File f: files){
    if (f.getName().endsWith(".jpg")){
        ht.put(f.getName(), ImageIO.read(f));
    }
}
```

#### HASHTABLE

```
final BufferedImage bi = ht.get("logo.jpg");

JFrame frame = new JFrame(){
    public void paint(Graphics g){
        super.paint(g);
        g.drawImage(bi, 0, 0, null);
    }
};

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setPreferredSize(new Dimension(bi.getWidth(), bi.getHeight()));

frame.pack();

frame.setVisible(true);
```

ĆWICZENIE: zmodyfikować program tak, aby wczytane obrazki zmieniały się.

#### HASHTABLE

```
Inne przydatne metody:
public Set<K> keySet();
public synchronized Enumeration<K> keys();
public Collection<V> values();
public synchronized Enumeration<V> elements();
public Set<Map.Entry<K,V>> entrySet();
public synchronized V remove(Object key);
```

```
Properties to rozszerzenie tablicy haszującej
public class Properties extends Hashtable<Object,Object>;
nastawione na przechowywanie par Stringów:
public synchronized Object setProperty(String key, String value);
public String getProperty(String key);
ZASTOSOWANIA:
Properties p;
p = System.getProperties();
```

p.list(System.out);

```
Implementacja metody list():
   public void list(PrintStream out) {
       out.println("-- listing properties --");
       Hashtable h = new Hashtable():
       enumerate(h);
       for (Enumeration e = h.keys(); e.hasMoreElements();) {
           String key = (String)e.nextElement();
           String val = (String)h.get(key);
           if (val.length() > 40) {
               val = val.substring(0, 37) + "...";
           out.println(key + "=" + val);
```

DO ZASTANOWIENIA: dlaczego metoda najpierw przepisuje dane do nowej tablicy i dopiero z niej je wypisuje?

Klasa Properties "współpracuje" z plikami tekstowymi zapisanymi w określonym formacie:

```
public synchronized void load(InputStream inStream) throws IOException
public void store(OutputStream out, String comments)
    throws IOException
```

```
public synchronized void loadFromXML(InputStream in)
    throws IOException, InvalidPropertiesFormatException
```

public synchronized void storeToXML(OutputStream os, String comment)
 throws IOException

```
# analizowany plik w formacie gif, png, jpg
image=obrazek.png
output=res.png
# inne ustawienia
moversCount=500
stepSize=1
ballRadius=1
Properties p = new Properties();
p.load(new FileInputStream("ustawienia.txt"));
p.storeToXML(new FileOutputStream("ustawienia.xml"), "");
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
cproperties>
   <comment/>
   <entry key="stepSize">1</entry>
   <entry key="ballRadius">1</entry>
   <entry key="output">res.png</entry>
   <entry key="moversCount">500</entry>
   <entry key="image">obrazek.png</entry>
</properties>
```

## DWIE UŻYTECZNE KLASY

Podstawowe operacje na tablicach lub kolekcjach danych można wykonać za pomocą klas **Arrays** i **Collections**.

np sortowanie.

```
public static <T extends Comparable<? super T>> void sort(List<T> list);
przy czym:

public interface Comparable<T> {
    public int compareTo(T o);
```

# ĆWICZENIA

 Proszę porównać wydajność (czas wykonywania) operacji: add(), remove(), contains(), toArray(), dla wszystkich klas rozszerzających interfejs Collection ze strony 9.

# DZIĘKUJĘ ZA UWAGĘ

## TYPY GENERYCZNE (GENERICS)

#### ZAGADNIENIA:

- wprowadzenie,
- konwencje, metody, typy surowe
- parametry ograniczone
- podtypy, dziedziczenie,
- symbole wieloznaczne,
- ograniczenia.

#### MATERIAŁY:

http://docs.oracle.com/javase/tutorial/java/generics/

#### TYPY GENERYCZNE

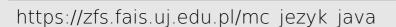
```
List<String> list = new ArrayList<String>();
list.add("Ala");
String s = list.get(0);
```

#### ZALETY:

- kontrola typów na poziomie kompilacji, a nie dopiero w trakcie działania programu
- brak potrzeby rzutowania,
- konstruowanie ogólnych algorytmów.

## PRZYKŁAD

```
public class Box<T> { // T oznacza Typ
    private T t;
    public void set(T t) {
        this.t = t;
    public T get() {
        return t;
ogólnie:
class NazwaKlasy<T1, T2, ..., Tn> {
```



## KONWENCJE

```
• T - typ,
• K - klucz,

    V – wartość,

• E - element (np. kolekcji),
• N - liczba
PRZYKŁAD:
public interface Pair<K, V> {
   public K getKey();
   public V getValue();
```



### PRZYKŁADY

```
public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;
    public OrderedPair(K key, V value) {
        this.key = key;
       this.value = value;
    public K getKey() { return key; }
    public V getValue() { return value; }
Pair<String, Integer> p = new OrderedPair<String, Integer>("Even", 8);
Pair<String, Integer> p = new OrderedPair<>("Even", 8); // Java 7
```

### TYPY SUROWE

```
public class Box<T> {
    public void set(T t){ ... }
    ...
}

Box<String> typedBox = new Box<String>();
Box rawBox = new Box(); // typ surowy <=> Box<Object>

Box rawBox = typedBox;

rawBox.set(8); // warning: w trakcie działania programu może
    // spowodować RuntimeException
```

Ze względu na możliwość popełnienia błędów, nie zaleca się stosowania typów surowych.

### METODY GENERYCZNE

```
public class Util {
   // Genericzna metoda statyczna
   public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
       return p1.getKey().equals(p2.getKey()) &&
              p1.getValue().equals(p2.getValue());
wywołanie metody generycznej:
Pair<Integer, String> p1 = new Pair<>(1, "jab{ko");
Pair<Integer, String> p2 = new Pair<>(2, "gruszka");
boolean same = Util.<Integer, String>compare(p1, p2);
możliwe również prostsze wywołanie dzięki mechanizmowi
wnioskowania typów (type inference).
boolean same = Util.compare(p1, p2);
```

#### PARAMETRY OGRANICZONE

```
public class Box<T> {
    public <U extends Number> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.set(new Integer(10));
        integerBox.inspect("some text"); // error: String to nie Number
inny przykład
public class NaturalNumber<T extends Integer>
```

#### PARAMETRY OGRANICZONE

wielokrotne ograniczenia:

```
<T extends B1 & B2 & B3>

przykład:

class A { /* ... */ }

interface B { /* ... */ }

interface C { /* ... */ }

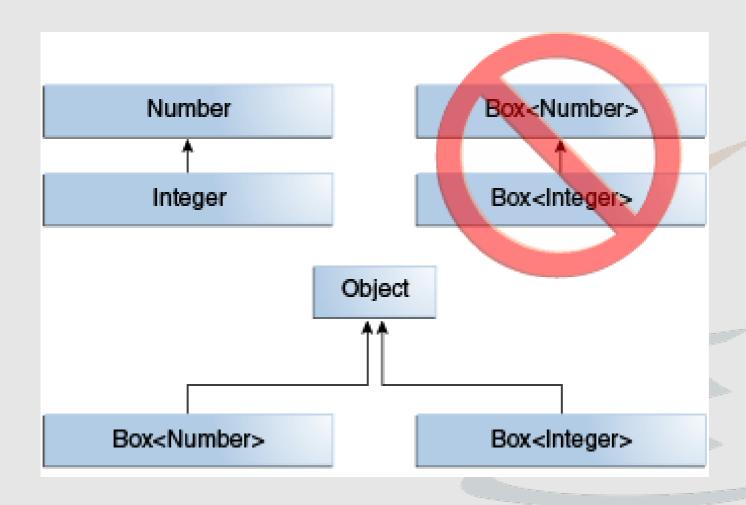
class D <T extends A & B & C> { /* ... */ }
```

typ **T** musi rozszerzać klasę **A** i implementować interfejsy **B** i **C**. UWAGA: **A** musi być zadeklarowana jako pierwsza (błąd w trakcie kompilacji)

#### PRZYKŁAD

```
public static <T> int countGreaterThan(T[] anArray, T elem) {
   int count = 0:
    for (T e : anArray)
        if (e > elem) // błąd kompilacji. Co oznacza '>'?
            ++count;
    return count;
poprawnie:
public static <T extends Comparable<T>> int countGreaterThan(T[] anArray,
                                                              T elem) {
   int count = 0;
    for (T e : anArray)
        if (e.compareTo(elem) > 0)
            ++count;
    return count;
```

# PODTYPY, DZIEDZICZENIE

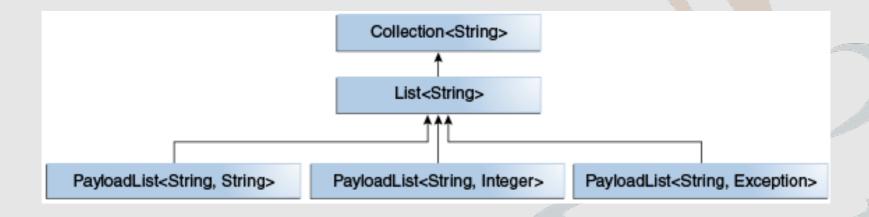


http://docs.oracle.com/javase/tutorial/figures/java/generics-subtypeRelationship.gif

## DZIEDZICZENIE PRZYKŁAD

## PODTYPY, DZIEDZICZENIE

```
interface PayloadList<E,P> extends List<E> {
 void setPayload(int index, P val);
```



http://docs.oracle.com/javase/tutorial/figures/java/generics-payloadListHierarchy.gif

```
public static double sumOfList(List<? extends Number> list) {
    double s = 0.0;
    for (Number n : list)
        s += n.doubleValue();
    return s;
}
Znak '?' ogranicza 'od góry' rodzaj parametru.

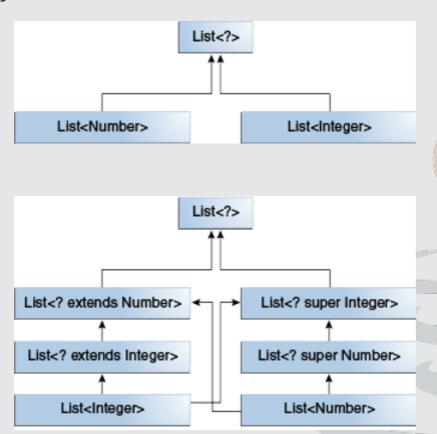
UWAGA: deklaracja / definicja metody
public static double sumOfList(List<T extends Number> list)
```

jest niepoprawna (składnia). Taka składnia jest douszczalna dla deklaracji klas lub wartości zwracanych (por. parametry ograniczone).

public static void printList(List<?> list) {

```
for (Object elem: list)
       System.out.print(elem + " ");
   System.out.println();
Znak '?' dopuszcza dowolny typ jako zawartość listy, np.
List<String>, List<Integer>, itd.
UWAGA: Metoda
public static void printList(List<Object> list)
nie obsłuży List<String> bo List<String> nie jest podklasą
List<Object>.
```

Dziedziczenie a symbole wieloznaczne



http://docs.oracle.com/javase/tutorial/figures/java/generics-wildcardSubtyping.gif

Przechwytywanie symboli wieloznacznych (wildcard capture):

```
public class WildcardError {
    void doSomething(List<?> l) {
        l.set(0, l.get(0)); // blad kompilacji
    }
}
```

Kompilator traktuje **l** jako listę dowolnych obiektów, podczas gdy wywołanie metody **l.set()** wstawia do listy konkretny obiekt, który niekoniecznie musi do niej pasować.

```
public class WildcardFixed {
    void foo(List<?> i) {
        fooHelper(i);
    }

    // metoda pomocnicza, która rozwiązuje problem
    private <T> void fooHelper(List<T> l) {
        l.set(0, l.get(0));
    }
}
```

Metoda wykorzystuje mechanizm wnioskowania (inference). Tutaj lista jest procesowana jak lista zawierająca konkretne elementy (typu **T**).

#### Wskazówki:

- zmienne IN górne ograniczenie extends,
- zmienne OUT dolne ograniczenie super,
- zmienne IN, mogące być używane jako obiekty brak ograniczeń,
- zmienne IN/OUT nie używamy symboli wieloznacznych,
- nie używamy symboli wieloznacznych jako wartości zwracanych.

Listy zadeklarowane z użyciem symboli wieloznacznych można traktować jako obiekty read-only

```
List<Integer> li = new ArrayList<Integer>();
List<? extends Number> ln = li; // OK. ( Object obj = new String() )
ln.add(new Integer(35)); // Blad kompilacji, ( obj.charAt(7) )
```

List<Integer> jest podtypem List<? extends Number>. A klasy nie można traktować tak jak podklasy. Jedyne operacje które można wykonać na takiej liście to:

- dodanie null
- wyczyszczenie clear()
- pobranie iteratora i wywołanie remove()
- zapisać elementy odczytane z listy z wykorzystaniem 'wildcard capture'.

## DZIAŁANIE

Typy generyczne zostały wprowadzone aby lepiej kontrolować typy na etapie kompilacji i aby umożliwić programowanie generyczne. W generowanym bytecodzie otrzymujemy zwykłe klasy i interfersy. Aby to zrealizować kompilator:

- zastępuje typy przez ich ograniczenia a nieograniczone typy przez Object,
- dodaje niezbędne operacje rzutowania,
- tworzy metody pomostowe implementujące polimorfizm w rozszerzonych typach generycznych.

### DZIAŁANIE

```
public class Box<T> {
                                         public class Box {
    private T t;
                                             private Object t;
    public void set(T t) {
                                             public void set(Object t) {
        this.t = t;
                                                 this.t = t;
public class Box<T extends Number> {
                                         public class Box {
                                             private Number t;
    private T t;
    public void set(T t) {
                                             public void set(Number t) {
        this.t = t;
                                                 this.t = t;
```

## METODY POMOSTOWE

```
public class MyBox extends Box<String>{
    public void set(String s){
        // jakiś kod
    }
```

metoda **set()** w klasie **Box** przyjmuje argument Object (wg bytecode wykonywanego przez JVM) podczas gdy metoda w klasie potomnej przyjmuje **String**. To powoduje brak polimorfizmu. Dlatego kompilator automatycznie dodaje metodę pomostową (do **MyBox**):

```
public void set(Object data) {
    set((String) data);
}
```

### OGRANICZENIA

- nie można używać typów prymitywnych (Box<int>),
- nie można używać operatora new (E e = new E();),
- nie można deklarować typów statycznych (private static T v;)
- typów parametryzowanych (List<Integer>) nie można rzutować ani używać jako argument w operatorze instanceof,

## OGRANICZENIA

- nie można używać tablic typów parametryzowanych (List<Integer>[] arrayOfLists = new List<Integer>[2]),
- typ generyczny nie może rozszerzać (bezpośrednio lub pośrednio) Throwable,
- nie można przeciążać metod, których argumenty sprowadzają się do tego samego typu.

```
public class Example {
    public void print(Set<String> strSet) { }
    public void print(Set<Integer> intSet) { }
}
```

## ĆWICZENIA

Proszę zaimplementować kolejkę priorytetową – strukturę, przechowującą pary (obiekt, priorytet), implementującą operacje:

- void add(T t, int priotity); // wstawia nowy element do kolejki
- T get(); // zwraca i usuwa z kolejki element o najwyższym priorytecie. Jeśli w kolejce znajduje się kilka obiektów o najwyższym priorytecie, zwracany jest ten z nich, który został tam wstawiony najwcześniej.

Złożoność czasowa operacji add() i get() powinna być możliwie najmniejsza.

# DZIĘKUJĘ ZA UWAGĘ

## 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)**):

## 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++)</pre>
        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 ClassLoadera 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:

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.

**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:

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

## WSPÓŁBIEŻNOŚĆ

#### ZAGADNIENIA:

- procesy,
- wątki,
- synchronizacja,
- blokady, egzekutory, zbiory wątków

#### MATERIAŁY:

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

## PROCESY I WĄTKI

Program w Javie uruchamiany jest w ramach **pojedynczego** procesu JVM. Z tego powodu implementacja współbieżności koncentruje się głównie na obsłudze wątków. Niemniej Java udostępnia także mechanizmy umożliwiające uruchamianie procesów systemu operacyjnego.

Proces można stworzyć korzystając z metody **Runtime.exec()** lub klasy **ProcessBuilder** z pakietu **java.lang**.

#### PROCESY

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class ProcessExample {
    public static void main(String[] args) {
        try {
            String s;
            Process ps = Runtime.getRuntime().exec("ls -l");
            BufferedReader bri = new BufferedReader(new InputStreamReader(
                                                     ps.getInputStream());
            BufferedReader bre = new BufferedReader(new InputStreamReader(
                                                     ps.getErrorStream());
            while ((s = bri.readLine()) != null)
                System.out.println(s);
            bri.close():
```

### PROCESY

```
while ((s = bre.readLine()) != null)
        System.out.println(s);
    bre.close();
    ps.waitFor();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
System.out.println("Gotowe.");
```

#### PROCESY

Klasa **ProcessBuilder** pozwala precyzyjniej określić środowisko, w którym działa proces:

```
ProcessBuilder builder = new ProcessBuilder("ls", "-l");
builder.directory(new File("."));
builder.redirectErrorStream(true);
builder.redirectOutput(Redirect.INHERIT);
Process ps;
try {
    ps = builder.start();
    ps.waitFor();
} catch (IOException | InterruptedException e) {
        e.printStackTrace();
}
System.out.println("Gotowe.");
```

Warto zajrzeć na: http://www.rgagnon.com/javadetails/java-0014.html

## WĄTKI

Istnieją dwa podstawowe sposoby tworzenia wątków. Pierwszy polega na rozszerzeniu klasy java.lang.Thread:

```
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Witam z watku");
    }

    public static void main(String args[]) {
        Thread t = new HelloThread();
        t.start();
        System.out.println("Witam z programu");
    }
}
```

## WĄTKI

Drugi opiera się na skonstruowaniu wątku w oparciu o klasę implementującą interfejs **java.lang.Runnable**: public class HelloRunnable implements Runnable { public void run() { System.out.println("Witam z watku"); public static void main(String args[]) { Thread t = new Thread(new HelloRunnable()); t.start(); System.out.println("Witam z programu"); Ten przypadek jest ogólniejszy (i zalecany), gdyż klasa implementująca watek może rozszeżać inną klasę.

## WĄTKI

```
Runnable task = new Runnable(){
    public void run() {
        System.out.println("Witam z watku");
Thread t = new Thread(task);
t.start();
Thread t = new Thread(new Runnable(){
    public void run() {
        System.out.println("Witam z watku");
});
t.start();
Thread t = new Thread(() -> { System.out.println("Witam z watku");} );
t.start();
```

#### KLASA THREAD

Wątki mogą być wstrzymywanie oraz wzbudzane/przerywane:

```
public class InterruptExample implements Runnable{
    public void run() {
        try {
            Thread.sleep(10000); // wstrzymanie na 10 sek.
        } catch (InterruptedException e) {
            System.out.println("interupted");
    public static void main(String[] args) throws InterruptedException{
        Thread t = new Thread(new InterruptExample());
        t.start();
        Thread.sleep(5000);
        System.out.println("budzenie");
        t.interrupt();
```

#### KLASA THREAD

Można również poczekać na zakończenie wskazanego wątku.

```
public class JoinExample implements Runnable{
    public void run() {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            return;
        System.out.println("watek");
    public static void main(String[] args) throws InterruptedException{
        Thread t = new Thread(new JoinExample());
        t.start();
        t.join(); // czekamy na zakonczenie t
        System.out.println("teraz ja");
```

## SYNCHRONIZACJA

Wątki mogą się komunikować przez dowolne współużytkowane zasoby (np. referencje do obiektów). Jest to bardzo efektywne, jednak może powodować problemy, gdy kilka wątków korzysta z jednego zasobu.

## SYNCHRONIZACJA METOD

Problem ten rozwiązuje się używając synchronizacji. Oznaczenie metod słowem **synchronized** powoduje, że w danej chwili może być wykonywana tylko jedna z nich (i tylko przez jeden wątek).

## BLOKADA WEWNĘTZNA

Każdy obiekt posiada powiązaną z nim blokadę wewnętrzna (intrinsic lock). Jeśli wątek chce uzyskać wyłączny dostęp do obiektu (lub jego atrybutów) może skorzystać z tej blokady.

```
public void addName(String s) {
    synchronized(this) {
        name = s;
        counter++;
    }
    nameList.add(name);
}
```

Wątek musi jednoznacznie wskazywać obiekt, z którym jest związana używana przez niego blokada. Taki tyb klokad nazywamy synchronizacją na poziomie instrukcji (synchronized statements)

### BLOKADA DROBNOZIARNISTA

Dostęp do c1 i c2 musi być synchronizowany niezależnie - nie chcemy blokować na raz obu liczników.

```
public class FineGrainedLockEx {
    private long c1 = 0, c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();
    public void incl() {
        synchronized(lock1) {
            c1++;
    public void inc2() {
        synchronized(lock2) {
            c2++;
```

## OPERACJE ATOMOWE

Typowo dostęp do zmienne/referencji nie jest realizowany jako pojedyncza operacja. Aby takie operacje (odczytu/zapisu zmiennej) były atomowe należy oznaczyć ją słowem **volatile**.

#### UWAGA:

więcej:

http://www.javamex.com/tutorials/synchronization\_volatile.shtml

#### TYPOWE PROBLEMY

W programach wielowątkowych występuje kilka rodzajów problemów, które mogą powodować niewłaściwe działanie programów:

- zakleszczenia (deadlock) wątek blokują wzajemnie zasoby potrzebne im do dalszego działania (pięciu filozofów).
- zagłodzenia (starvation) jeden wątek przez cały blokuje zasób potrzebny innym wątkom.
- livelock "odwrotność" deadlocka wątek reaguje na zachowanie drugiego wątku, które jest reakcją na zachowanie pierwszego wątku.

#### ZAKLESZCZENIE

```
public class Deadlock {
    static class Worker {
        public String name;
        public Worker(String name) {
            this.name = name;
        public synchronized void doWork(Worker w){
            System.out.println(this.name + " pracuje z " + w.name);
            try {
                Thread.sleep(1000); // pracujemy
            } catch (InterruptedException e) { }
            w.release();
        public synchronized void release() {
            System.out.println(this.name + " jest znowu gotowy");
```

#### ZAKLESZCZENIE

```
public static void main(String[] args) {
    final Worker w1 = new Worker("w1");
    final Worker w2 = new Worker("w2");

    new Thread(new Runnable() {
        public void run() { w1.doWork(w2); }
    }).start();

    new Thread(new Runnable() {
        public void run() { w2.doWork(w1); }
    }).start();
}
```

Obaj workerzy zaczną ze sobą pracować. Blokada nastąpi na metodzie release() w obu obiektach.

## WAIT / NOTIFY

Często wątek musi poczekać, aż inny wąteg wykona określoną część swoich zadań, np. jeden wątek oblicza wyniki, a drugi je sukcesywnie wypisuje na ekranie.

```
public synchronized consume() {
    while(!available) {
        try {
            wait(); // wstrzymuje dzialanie watku i zwalnia blokade
        } catch (InterruptedException e) {}
    }
    System.out.println("Skonsumowane");
    available = false;
}
public synchronized produce() {
    doProduce();
    available = true;
    notifyAll(); // powiadamia (budzi) wszystkie czekajace watki
}
```

## SYNCHRONIZACJA W JRE 5.0

```
Począwszy od wersji 5.0 Java udostępnia dodatkowe,
wysokopoziomowe API ułatwiające synchronizację wątków. Przykład -
blokady:
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class LockObjects {
   static class Worker {
       public Lock lock = new ReentrantLock();
       public String name;
       public Worker(String name) {
           this.name = name;
```

#### BLOKADY

```
public boolean tryWorking(Worker w) {
    boolean myLock = lock.tryLock();
    boolean wLock = w.lock.tryLock();
    if (!(myLock && wLock)) { // zwalniamy blokady
        if (myLock)
            lock.unlock();
        if (wLock)
            w.lock.unlock();
    return myLock && wLock;
public synchronized void doWork(Worker w) {
    boolean done = false;
```

#### BLOKADY

```
while (!done) {
    if (tryWorking(w)) {
        System.out.println(name + ": pracuje z " + w.name);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
}
        w.release();
        this.lock.unlock();
        done = true;
    } else {
       System.out.println(name+": jestem zajety wiec czekam");
        try {
            wait();
        } catch (InterruptedException e) { }
        System.out.println(this.name + ": probuje znowu");
    }
```

#### BLOKADY

```
public synchronized void release() {
        System.out.println(this.name + ": jestem znowu gotowy");
        this.lock.unlock();
       notifyAll();
public static void main(String[] args) {
    final Worker w1 = new Worker("w1");
    final Worker w2 = new Worker("w2");
    new Thread(new Runnable() {
                  public void run() { w1.doWork(w2); }
             }).start();
    new Thread(new Runnable() {
                  public void run() { w2.doWork(w1); }
             }).start();
```

Egzekutory oddzielają wątek od zarządzania nim

```
Executor exec = ...;
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2());
```

Executor jest rozszerzany przez ExecutorService, który dodatkowo umożliwia uruchamianie wątków Callable, które mogą zwracać wynik działania.

Kolejnym rodzajem egzekutorów są tzw. **Thread Pools** – zbiory, wspólnie zarządzanych wątków. W JRE 7.0 wprowadzono ForkJoinPool, przeznaczony do wieloprocesorowych obliczeń równoległych.

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
public class ParallelFor {
    public static void main(String[] args){
        ExecutorService es = Executors.newFixedThreadPool(16);
        Vector<Future> vf = new Vector<>();
        for(int i=0; i<1000000; i++) {
            final int no = i:
            Future<Object> f = es.submit(new Callable<Object>() {
                @Override
                public Object call() throws Exception {
                    return anyMethod(no);
            });
            vf.add(f);
```

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;
public class Fibonacci extends RecursiveTask<Integer>{
    final int n;
    public Fibonacci(int n){
        this.n = n;
    protected Integer compute() {
        if (n>2){
            doRecursion():
        }else{
            return 1;
```

```
public Integer doRecursion(){
    Fibonacci f1 = new Fibonacci(n-1);
    Fibonacci f2 = new Fibonacci(n-2);
    f2.fork();
    return f1.compute()+f2.join();
}

public static void main(String[] args) {
    Fibonacci f = new Fibonacci(35);
    ForkJoinPool pool = new ForkJoinPool();
    System.out.println(pool.invoke(f));
}
```

Ponadto wprowadzono także wspólbieżny generator liczb losowych:

ThreadLocalRandom,

## ĆWICZENIA

 Proszę zaimplementować dowolny algorytm typu dziel i zwyciężaj (np. sortowanie szybkie) wykorzystując wątki. Proszę porównać wydajność algorytmu wielowątkowego z wersją iteracyjną dla różnych rozmiarów danych wejściowych, a następnie spróbować wytłumaczyć otrzymane wyniki.

# DZIĘKUJĘ ZA UWAGĘ

## SWING

#### ZAGADNIENIA:

- wprowadzenie,
- kontenery i komponenty,
- LayoutManager,
- komponenty tekstowe.
- inne przydatne komponenty.

#### MATERIAŁY:

http://docs.oracle.com/javase/tutorial/uiswing/

## SWING

```
import javax.swing.*;
public class HelloWorldSwing {
    private static void createAndShowGUI() {
        // nowe okno o tytule HelloWorldSwing
        JFrame frame = new JFrame("HelloWorldSwing");
        // zamkniecie okna spowoduje zakonczenie programu
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // nowy napis
        JLabel label = new JLabel("Hello World");
        // napis jest dodawany do zawartosci okna
        frame.getContentPane().add(label);
        // dopasowanie rozmiarow okna do umieszczonych w nim komponentow
        frame.pack();
        // wyswietlenie okna
        frame.setVisible(true);
```

## SWING

```
public static void main(String[] args) {
   // stworzenie nowego watku, w ktorym zostanie 'uruchomione' okno
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
    });
```

## KONTENERY

Każda aplikacja wykorzystująca Swing używa co najmniej jeden kontener najwyższego poziomu.

- JFrame,
- JDialog,
- JApplet.

W takim kontenerze umieszcza się kolejne kontenery lub komponenty:
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);

// A tutaj tworzymy panel (kontener) dodajemy do niego komponenty
// a nastepnie umieszczamy go w kontenerze najwyzszego poziomu (frame)
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE\_END);
frame.getContentPane().add(contentPane)

## KONTENERY

Za rozmieszczenie komponentów w kontenerze odpowiedxzialny jest obiekt typu LayoutManager. Możemy go określić za pomocą metody setLayout(). Domyślnie kontenery używają instancji FlowLayout(), Jeśli chcemy "samodzielnie" określać pozycje komponentów należy usunąć LayoutManagera: container.setLayout(null).

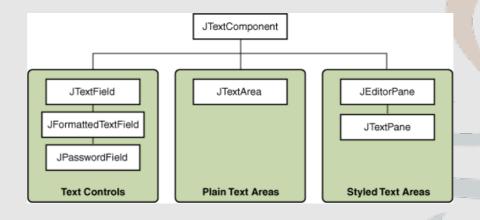
Do kontenerów najwyższego poziomu można także dodać menu: frame.setJMenuBar().

Wszystkie komponenty rozszerzają klasę **JComponent**. Klasa **JComponent** implementuje następujące funkcjonalności:

- podpowiedzi (setToolTipText()),
- ramki (setBorder()),
- styl (UIManager.setLookAndFeel()),
- dodatkowe właściwości (setClientProperties()),
- rozmiary (layout)
- przystępność (accessibility)
- przeciągnij i upuść,
- podwójne buforowanie,
- wiązanie klawiszy.

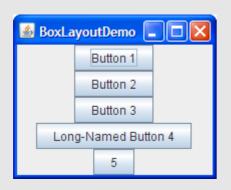
## KOMPONENTY TEKSTOWE

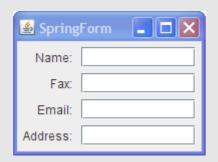
Komponenty tekstowe (JTextComponent) dzielą się na trzy ka<mark>t</mark>egorie:

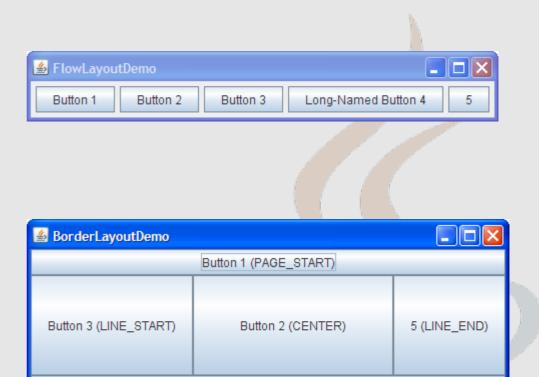


```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.URL;
import javax.swing.*;
public class Browser extends JFrame implements ActionListener {
    private static final String COMMAND GO = "go";
    private JEditorPane webpage;
    private JTextField url;
    private JTextArea htmlPage;
    private JPanel createMainPanel() {
        JPanel mp = new JPanel();
        // panel z kolumnowym (Y AXIS) ukladem elementow
        mp.setLayout(new BoxLayout(mp, BoxLayout.Y AXIS));
```

## LAYOUT MANAGER







Long-Named Button 4 (PAGE\_END)

```
// panel z domyślnym FlowLayout
JPanel p = new JPanel();
this.url = new JTextField():
// sugerowane rozmiary — mogą zostać zmienione przez layout
// managera
this.url.setPreferredSize(new Dimension(500, 20));
this.url.setText("http://www.simongrant.org/web/guide.html");
JLabel l = new JLabel("adres");
// label opisujacy url
l.setLabelFor(this.url);
// dodajemy do panelu JLabel I JTextField
p.add(l);
p.add(this.url);
```

```
// tworzymy przycisk
JButton b = new JButton("Go");
b.setActionCommand(COMMAND_GO);
// ktorego "akcje" beda obslugiwane przez biezacy obiekt
// (implementujacy interfejs ActionListener. Akcja "produkowana"
// przez przycisk będzie identyfikowane Stringiem COMMAND_GO
b.addActionListener(this);
b.setPreferredSize(new Dimension(100, 40));
// dodanie przycisku do panelu
p.add(b);
// dodanie panelu p (z komponentami rozmieszczonymi przez
// FlowLayout) do panelu, w ktorym obowiazuje BoxLayout
mp.add(p);
```

```
this.webpage = new JEditorPane();
this.htmlPage = new JTextArea();
try {
   // wczytujemy zawartosc strony "startowej"
    this.setPage(
             new URL("http://www.simongrant.org/web/guide.html"));
} catch (IOException e) { }
// Tworzymy panel z zakladkami
JTabbedPane tp = new JTabbedPane();
tp.setPreferredSize(new Dimension(600, 400));
// pole tekstowe webpage umieszczamy wewnatrz panela
// scrollowanego. Dzieki temu zawartosc okienka bedzie mogla
// zajmowac wiecej miejsca niz widok
JScrollPane sp = new JScrollPane(this.webpage);
// zakladka "page" bedzie zawierac webpage (wewnatrz JScrollPane)
tp.add("page", sp);
```

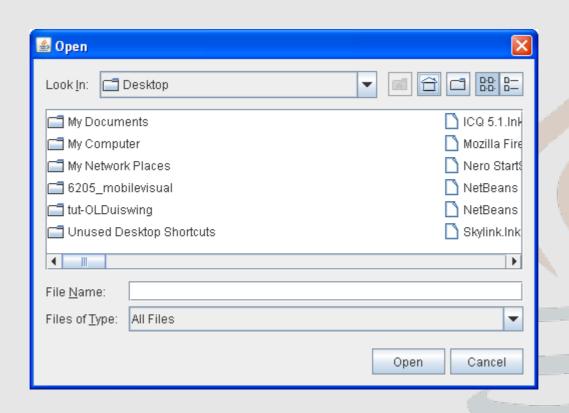
```
// zakladka "html" bedzie zawierac htmlPage (wewnatrz JscrollPane)
   sp = new JScrollPane(this.htmlPage);
   tp.add("html", sp);
   // przygotowany JTabbedPane zostaje dodany do panelu mp
   mp.add(tp);
   return mp;
private void setPage(URL page) throws IOException {
   String s;
   this.webpage.setPage(page);
   BufferedReader br = new BufferedReader(new InputStreamReader(
                                                   page.openStream());
   while ((s = br.readLine()) != null)
       this.htmlPage.append(s + "\n");
```

```
public Browser() {
   // zawsze na początku powinniśmy wywołać konstruktor nadklasy
   super();
   // zawartoscia okna Browser bedzie panel mp
   this.getContentPane().add(this.createMainPanel());
public static void createAndShow() {
   Browser b = new Browser();
   b.setDefaultCloseOperation(EXIT_ON_CLOSE);
   b.pack();
   // okno zostanie umieszczone na srodku ekranu
   b.setLocationRelativeTo(null);
   b.setVisible(true);
```

```
// Interfejs ActionListener implementuje jedna metode, ktora jest
// uruchamiana gdy nastapi zdarzenie na komponencie nasluchiwanym
// przez ten obiekt. Informacje o zrodle akcji sa przekazywane przez
// argument ActionEvent
@Override
public void actionPerformed(ActionEvent e) {
    if (COMMAND GO.equals(e.getActionCommand())) {
        try {
            // przeladowujemy strone
            this.setPage(new URL(this.url.getText()));
        } catch (IOException e2) {
            this.webpage.setText(
                           "Problem z adresem " + this.url.getText());
            this.htmlPage.setText(
                           "Problem z adresem " + this.url.getText());
```

```
// uruchomienie programu
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        createAndShow();
});
```

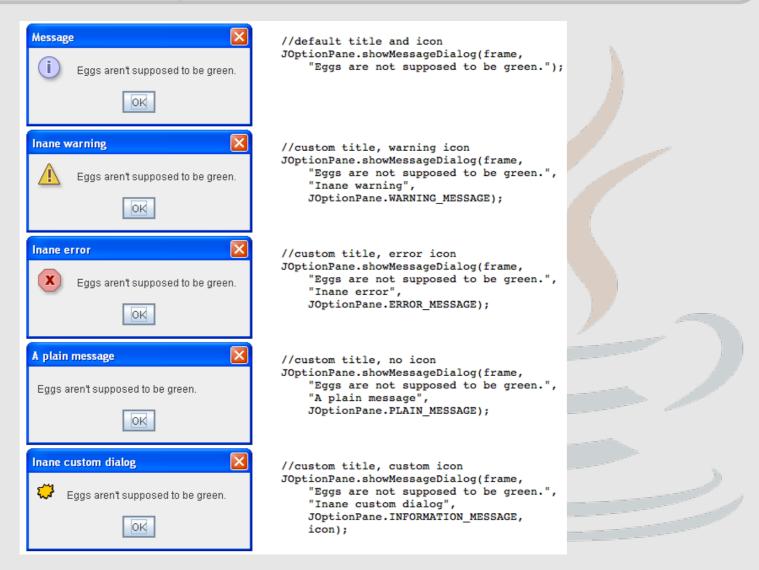
# JFileChooser



http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

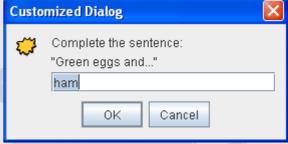
## JFileChooser

# JOptionPane



## JOptionPane





http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html#features

# ĆWICZENIA

- Proszę napisać kalkulator posiadający interfejs graficzny.
   (łatwiejsza wersja kalkulator działający w oparciu o odwrotną notację polską)
- Proszę napisać prosty edytor tekstowy umożliwiający tworzenie, odzczytywanie, modyfikację I zapis plików w formacie txt.

# DZIĘKUJĘ ZA UWAGĘ

## JAVA I SIECI

#### ZAGADNIENIA:

- URL,
- Interfejs gniazd,
- transmisja SSL,
- protokół JNLP.

#### MATERIAŁY:

http://docs.oracle.com/javase/tutorial/networking/index.html

#### URL

URL – Unified Resource Locator jest podstawową klasą identyfikującą zasoby w internecie:

### URLConnection

URLConnection zawiera metody umożliwiające nawiązanie połączenia z zasobem reprezentowanym przez URL.

### URLConnection

URLConnection umożliwia także zapis do wskazanego zasobu przez obiekt URL.

```
import java.io.*;
import java.net.*;
public class URLConnectionWriter {
    public static void main(String[] args) throws Exception {
        URL url = new URL(args[0]);
        URLConnection con = url.openConnection();
        con.setDoOutput(true);
        OutputStreamWriter out = new OutputStreamWriter(
                                                     con.getOutputStream());
        for (int i = 1; i < args.length; i++)</pre>
            out.write(args[i]);
        out.close():
```

## URLConnection

Zarówno w przypadku URL jak i URLConnection komunikacja odbywa się z wykorzystaniem odpowiedniego protokołu. Dokumentacja Javy gwarantuje standardowo obsługę następujących protokołów:

- http,
- https,
- ftp, ftp://login:haslo@serwer:port/katalog/podkatalog/plik
- file,
- jar.

Obsługa innych protokołów wymaga implementacji klasy URLStreamHandler.

## INTERFEJS GNIAZD

Standardowa obsługa sieci w Javie opiera się o tzw. interfejs gniazd. Najważniejsze klasy, umożliwiające komunikację poprzez sieć to:

- Socket klasa reprezentująca gniazdo służące do nawiązywania połączenia, wysyłania I odbierania danych,
- ServerSocket klasa reprezentująca gniazdo oczekujące na przychodzące żądania połączeń.

Ponadto istnieją także gniazda SSLSocket I SSLServerSocket obsługujące komunikację szyfrowaną protokołem SSL/TLS.

## PROGRAM KLIENCKI

```
import java.io.*;
import java.net.Socket;
public class ClientExample {
    public static Socket sock;
    public static void main(String[] args) throws IOException{
        // tworzymy gniazdo i nawiazujemy polaczenie z komputerem
        // identyfikowanym przez adres args[0] na porcie args[1]
        sock = new Socket(args[0], Integer.value0f(args[1]));
        // pobieramy strumienie zwiazane z gniazdem
        OutputStream os = sock.getOutputStream();
        InputStream is = sock.getInputStream();
        // tworzymy Reader na standardowym wejsciu (klawiaturze)
        BufferedReader br = new BufferedReader(new InputStreamReader())
                                                               System. in));
```

## PROGRAM KLIENCKI

```
// zmienne pomocnicze
        String sLine;
        byte[] bRes = new byte[100];
        // glowna petla programu, pobieramy dane z klawiatury
        while((sLine=br.readLine())!=null){
            // wysylamy je przez gniazdo
            os.write(sLine.getBytes());
            System.out.println("wyslalem: " + sLine);
            // odbieramy odpowiedz z serwera — to jest zle rozwiazanie
            // dobre rozwiązanie — odbieranie danych w osobnym wątku
            is.read(bRes):
            System.out.println("odebralem" + new String(bRes));
        // zamykamy strumien i gniazdo
        br.close();
        sock.close();
} // koniec programu
```

### PROGRAM KLIENCKI

Osobny wątek do odbioru danych:

```
Thread t = new Thread(new Runnable(){
    public void run(){
        byte[] bRes = new byte[100];
        InputStream is;
        int l:
        try {
            is = sock.getInputStream();
            while(true){
                l = is.read(bRes);
                System.out.println("odebralem: " + new String(bRes,0,1));
        } catch (IOException e) {
            e.printStackTrace();
});
t.start();
```

#### SERWER ECHO

```
import java.io.*;
import java.net.*;

public class ServerExample {

    // gniazdo oczekujace na polaczenia
    private static ServerSocket ss;

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

          // tworzymy gniazdo, ktore oczekuje na przychodzace polaczenia
          // na porcie przekazanym jako argument wywolania programu
          ss = new ServerSocket(Integer.valueOf(args[0]));
```

### SERWER ECHO

```
// nieskonczona petla
while(true){
    // akceptujemy polaczenie, dostajemy gniazdo do komunikacji
    // z klientem
    Socket s = ss.accept();
    // strumienie
    InputStream is = s.getInputStream();
    OutputStream os = s.getOutputStream();
    int b:
    // czytamy, piszemy na konsoli i odsylamy
    while((b=is.read())!=-1){
        System.out.print((char)b);
        os.write(b);
    s.close();
```

## SERWER ECHO

W serwerze nie ma problemu związanego z transmisją strumieniową ponieważ dane są przetwarzane bajt po bajcie, w związku z czym nie sytuacja, gdy dane dotrą w różnych pakietach nie spowoduje żadnych efektów ubocznych.

## SSL W JAVIE

Protokół SSL umożliwia bezpieczną (szyfrowaną) transmisję danych poprzez niezabezpieczoną sieć. Dodatkowo SSL umożliwia autoryzację stron komunikacji. W tym celu wykorzystywany jest mechanizm certyfikatów. Za transmisję z użyciem protokołu SSL odpowiedzialne są klasy zgrupowane w pakiecie javax.net.SSL.

Implementacja SSH jest dostępna poprzez zewnętrzne biblioteki. Jedną z nich jest jsch (http://www.jcraft.com/jsch/).

### SERWER SSL

```
import javax.net.ssl.*;
import java.io.*;
public class EchoServer {
    public static void main(String[] args) throws IOException {
        SSLServerSocketFactory factory = (SSLServerSocketFactory)
        SSLServerSocketFactory.getDefault();
        SSLServerSocket ss = (SSLServerSocket) factory
                                                 .createServerSocket(9999);
        SSLSocket s = (SSLSocket) ss.accept();
        BufferedReader br = new BufferedReader(new InputStreamReader(
                                                     s.getInputStream()));
        String sTmp = null;
        while ((sTmp = br.readLine()) != null) {
            System.out.println(sTmp);
            System.out.flush();
```

### KLIENT SSL

```
import javax.net.ssl.*;
import java.io.*;
public class EchoClient {
    public static void main(String[] args) throws Exception {
        SSLSocketFactory factory = (SSLSocketFactory) SSLSocketFactory
                                                             .getDefault();
        SSLSocket s = (SSLSocket) factory.createSocket("localhost", 9999);
        BufferedReader br = new BufferedReader(
                                        new InputStreamReader(System.in));
        OutputStreamWriter osw = new OutputStreamWriter(
                                                      s.getOutputStream());
        BufferedWriter bw = new BufferedWriter(osw);
        String sTmp = null;
        while ((sTmp = br.readLine()) != null) {
            bw.write(sTmp + '\n');
            bw.flush();
```

# SSL W JAVIE

Pierwsza czynność to wygenerowanie klucza:

keytool -genkey -keystore mySrvKeystore -keyalg RSA

Uruchomienie serwera:

java -Djavax.net.ssl.keyStore=mySrvKeystore Djavax.net.ssl.keyStorePassword=123456 EchoServer

Uruchomienie klienta:

java -Djavax.net.ssl.trustStore=mySrvKeystore Djavax.net.ssl.trustStorePassword=123456 EchoClient

Dodatkowe parametry wywołania pozwolą zobaczyć informacje związane z połączeniem SSL:

- -Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol-Djavax.net.debug=ssl
- Przykład ze strony: http://stilius.net/java/java\_ssl.php.

# SSL W JAVIE

Domyślnie tylko jedna strona komunikacji (serwer) musi potwierdzać swoją tożsamość. Aby wymusić autoryzację klienta należy użyć metod: setNeedClientAuth(true) lub setWantClientAuth(true) wywołanych na rzecz obiektu SSLServerSocket.

Jeśli chcemy aby żadna ze stron nie musiała potwierdzać swojej tożsamości musimy zmienić domyślne algorytmy kodowania. Najłatwiej zrobić to tworząc własne rozszerzenie klasy SSLSocketFactory.

Listę obsługiwanych i domyślnych algorytmów uzyskamy za pomocą metod: getSuppotredCipherSuites() oraz getDefaultCipherSuites().

# JAVA WEB START

Technologia Java Web Start jest stosowana do lokalnego uruchamiania programów w Javie umieszczonych w sieci.

#### JWS:

- jest w pełni niezależna od używanych przeglądarek internetowych,
- umożliwia automatyczne pobranie właściwej wersji środowiska JRE,
- pobierane są tylko pliki, które zostały zmienione,
- obsługuje prawa dostępu do zasobów lokalnego komputera (dysk, sieć, itp.),
- do opisu zadania do uruchomienia wykorzystuje pliki jnlp (Java Network Launch Protocol).

#### Więcej informacji:

http://docs.oracle.com/javase/tutorial/deployment/webstart/index.html

# PROTOKÓŁ JNLP

```
<?xml version="1.0" encoding="utf-8"?>
<inlp
 spec="1.0+"
 codebase="http://www.serwer.w.sieci.pl/katalog"
 href="plik jws.jnlp">
 <information>
   <title>Nazwa programu</title>
   <vendor>Producent programu</vendor>
   <homepage href="http://www.strona.programu.pl"/>
   <description kind="short">Krotki opis programu</description>
   <icon kind="splash" href="kat/splashscreen.gif"/>
   <icon href="kat/ikona.gif"/>
   <offline-allowed/>
 </information>
 <security>
     <all-permissions/>
 </security>
```

# PROTOKÓŁ JNLP

```
<resources>
    <j2se version="1.4+"/>
    <jar href="kat/archiwum1.jar"/>
        <jar href=" kat2/lib/biblioteka.jar"/>
        </resources>
    <application-desc main-class="pl.edu.uj.if.ExampleClass"/>
</jnlp>
```

Plik jnlp umieszczamy na serwerze www.

Często należy skonfigurować odpowiadający mu typ mime: application/x-java-jnlp-file JNLP

# ĆWICZENIA

- Proszę napisać sieciową grę w kółko i krzyżyk, umożliwiającą rozgrywkę sieciową dla dwóch osób.
- Proszę napisać czat internetowy (komunikacja wzajemna wielu osób).

# DZIĘKUJĘ ZA UWAGĘ

# JAVA I XML

#### ZAGADNIENIA:

- DOM,
- SAX,
- JAXB, XMLDecoder i XMLEncoder,
- ANT.

#### MATERIAŁY:

http://www.mkyong.com/tutorials/java-xml-tutorials/ http://ant.apache.org

#### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:dc="http://purl.org/dc/elements/1.1/" version="2.0">
 <channel>
    <title>Aktualności</title>
   <link>http://www.uj.edu.pl/uniwersytet/aktualnosci/...</link>
    <description>Aktualności Uniwersytetu Jagiellońskiego</description>
   <item>
      <title>Lista Pamieci - Stanisław Szczur</title>
      <link>http://www.uj.edu.pl/uniwersytet/aktualnosci/...</link>
      <description />
      <pubDate>Thu, 20 Dec 2012 07:44:00 GMT</pubDate>
      <dc:creator>Jolanta Herian-Ślusarska</dc:creator>
      <dc:date>2012-12-20T07:44:00Z</dc:date>
   </item>
   <item>...</item>
 </channel>
</rss>
```

Parsery DOM (*Document Object Model*) tworzą drzewo reprezentujące dane zawarte w dokumencie XML. Po zbudowaniu DOM przetwarzanie odbywa się na modelu w pamięci operacyjnej.

```
URL url = new URL(
"http://www.uj.edu.pl/uniwersytet/aktualnosci/-/journal/rss/10172/36018?
doAsGroupId=10172&refererPlid=10181"); // jakis RSS
        // domyslna fabryka "obiektow tworzących dokumenty"
        DocumentBuilderFactory dbFactory =
                                    DocumentBuilderFactory.newInstance();
        // obiekt "tworzący dokumenty"
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        // tworzenie modelu DOM na podstawie zrodla XML (parsowanie XML'a)
        Document doc = dBuilder.parse(url.openStream());
        // obiekt doc umozliwia dostep do wszystkich danych zawartych
        // w dokumencie XML
        System.out.println("Root element :" +
                                  doc.getDocumentElement().getNodeName());
```

```
NodeList nList = doc.getElementsByTagName("item");
for (int i = 0; i < nList.getLength(); i++) {</pre>
   // lista "dzieci" i-tego itema
    Node n = nList.item(i);
   // czy "dziecko" jest elementem?
    if (n.getNodeType() == Node.ELEMENT NODE) {
        Element e = (Element) n;
        System.out.println("Tytul
                                   + getTagValue("title", e));
        System.out.println("Link
                                  + getTagValue("link", e));
        System.out.println("dodane przez : "
                                  + getTagValue("dc:creator", e));
```

```
// zwraca wartosc zapisana w tagu s wewnatrz elementu e
private static String getTagValue(String s, Element e) {
    // lista "dzieci" e o nazwie s
    NodeList nl = e.getElementsByTagName(s)
    // pierwszy wpis z tej listy
                                            .item(0)
    // to co on zawiera — jego "dzieci"
                                                    .getChildNodes();
    // pierwsze z tych dzieci
    Node n = (Node) \ nl.item(0);
    // zawartosc, ktora tam jest
    return n.getNodeValue();
```

# DOM - MODYFIKACJA

Możliwa jest także modyfikacja zawartości DOM utworzonego w wyniku parsowania dokumentu XML. Przykładowo:

```
// pobieramy element title i zmieniamy go
if ("title".equals(node.getNodeName())) {
    node.setTextContent("Nowy tytul");
}
// kasujemy link
if ("link".equals(node.getNodeName())) {
    itemElement.removeChild(node);
}

// tworzenie nowego dokumentu
Document doc = docBuilder.newDocument();
Element e = doc.createElement("root");
doc.appendChild(e);
```

#### DOM - ZAPIS

Zapis dokumentu:

```
// domyslna fabryka transformatorow
TransformerFactory transformerFactory = TransformerFactory.newInstance();
// nowy tranformator
Transformer transformer = transformerFactory.newTransformer();
// wejscie transformatora (skad transformator bierze dane)
DOMSource source = new DOMSource(doc);
// wyjscie transformatora (gdzie "zmienione" dane zostana zapisane)
StreamResult result = new StreamResult(new File("file.xml"));
// uruchomienie transformatora - zapis DOM do pliku w formacie XML
transformer.transform(source, result);
```

### SAX - ODCZYT

SAX (*Simple API for XML*) w miarę czytania dokumentu wywołuje zdarzenia związane z parsowaniem. Parsery SAX są szybsze i nie wymagają tak dużej ilości pamięci jak DOM.

### DefaultHandler

- void characters(char[] ch, int start, int length) wywoływane przy odczycie znaku wewnątrz elementu,
- void endDocument() wywoływane gdy koniec dokumentu
- void endElement(String uri, String localName, String qName) koniec elementu,
- void error(SAXParseException e) błąd (możliwy do naprawienia),
- void fatalError(SAXParseException e) błąd,
- void ignorableWhitespace(char[] ch, int start, int length) ignorowany
  pusty znak,
- void startDocument() początek dokumentu,
- void startElement(String uri, String localName, String qName,
   Attributes attributes) początek elementu
- void warning(SAXParseException e) ostrzeżenie parsera.

. . .

### DefaultHandler

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
class ExampleSAXHandler extends DefaultHandler {
    public void startElement(String uri, String localName, String qName,
                              Attributes attributes) throws SAXException {
        System.out.println("Element :" + qName);
    public void endElement(String uri, String localName, String qName)
                                                     throws SAXException {
        System.out.println("Konie elementu :" + qName);
    public void characters(char ch[], int start, int length)
                                                     throws SAXException {
        System.out.println("zawartosc : "+new String(ch, start, length));
```

## JAXB

JAXB (*Java Architecture for XML Binding*) to standard serializacji XML dla obiektów Javy. Został on zintegrowany z JDK/JRE od wersji 1.6

```
import java.io.File;
import javax.xml.bind.*;
import javax.xml.bind.annotation.*;
@XmlRootElement
class Person {
    String name;
    int age;
    public String getName() {
        return name;
    }
```

## JAXB

```
@XmlElement
public void setName(String name) {
    this.name = name;
public int getAge() {
    return age;
@XmlElement
public void setAge(int age) {
    this.age = age;
public String toString() {
    return this.name + " (" + this.age + ")";
```

## JAXB

```
public class JAXBExample{
    public static void main(String[] args) throws JAXBException {
        Person p = new Person();
        p.setName("Barnaba");
        p.setAge(33);
        File f = new File("person.xml");
        JAXBContext ctx = JAXBContext.newInstance(Person.class);
        Marshaller marshaller = ctx.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(p, System.out);
        marshaller.marshal(p, f);
        p = null;
        Unmarshaller unmarshaller = ctx.createUnmarshaller();
        p = (Person)unmarshaller.unmarshal(f);
        System.out.println(p);
```

# SERIALIZACJA I XML

Inna metoda serializacji niektórych obiektów do plików tekstowych w formacie XMI:

#### XMLEncoder:

```
XMLEncoder e = new XMLEncoder(new FileOutputStream("jbutton.xml"));
e.writeObject(new JButton("Hello world"));
e.close();
```

#### XMLDecoder:

```
XMLDecoder d = new XMLDecoder(new FileInputStream("jbutton.xml"));
obj = d.readObject();
d.close();
```

#### ANT

Ant jest narzędziem umożliwiającym automatyzację procesów związanych z budowaniem programów. Jego podstawowe cechy to:

- konfiguracja zadań zapisana w formacie XML,
- wieloplatformowość m. in. Linux, Unix (np. Solaris and HP-UX),
   Windows 9x i NT, OS/2 Warp, Novell Netware 6 oraz MacOS X.
- rozszerzalność w oparciu o klasy napisane w Javie.

Ant jest rozwijany w ramach Apache Software Foundation. Strona domowa projektu: http://ant.apache.org.

#### ANT

Przykładowy plik konfiguracyjny dla anta (domyślnie **build.xml**)

Aby go "wykonać" wpisujemy w konsoli polecenie ant.

# DZIĘKUJĘ ZA UWAGĘ

## JAVA I BAZY DANYCH

#### ZAGADNIENIA:

- wprowadzenie;
- JDBC;
- komunikacja z bazą danych;
- HSQLDB.

#### MATERIAŁY:

http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

## BAZY DANYCH - SQL

Tabela				
id	Imie	Nazwisko	Wiek	numer
1	Barbara	Tokarska	47	FSX458721
2	Tomasz	Czyżewski	24	HGJ874398
3	Jerzy	Pietras	35	VTF937836
4	Agata	Kałuża	17	JKD259077

```
SELECT Imie, Nazwisko FROM Tabela WHERE id = 3;
UPDATE Tabela SET Imie='Marek' WHERE id=4;
DELETE FROM Tabela WHERE Imie='Tomasz';
INSERT INTO Tabela (id, Imie) VALUES (5, 'Dorota');
```

## **JDBC**

Java Database Connectivity (JDBC) to specyfikacja określająca zbiór klas i interfejsów napisanych w Javie, które mogą być wykorzystane przez programistów tworzących oprogramowanie korzystające z baz danych. Implementacja JDBC jest dostarczany przez producentów baz danych.

Jedną z ważniejszych zalet takiego rozwiązania jest ukrycie przed programistą kwestii technicznych dotyczących komunikacji z bazą danych. Dzięki temu ten sam program napisany w Javie może współpracować z różnymi systemami baz danych (np. Oracle, Sybase, IBM DB2). Wystarczy podmienić odpowiednie biblioteki implementujące JDBC.

#### PRZYKŁAD

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDriver {
    public static void main(String[] args) {
        Statement stmt = null;
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con = DriverManager.getConnection(
                  "jdbc:mysql://localhost/test?user=monty&password=");
            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT * FROM Tabela");
            while (rs.next()) {
                System.out.println(rs.getString("Imie"));
```

### PRZYKŁAD

```
catch (Exception ex) { // obsluga bledow
} finally { // zwalnianie zasobow
   if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignorujemy
        rs = null;
   if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignorujemy
        stmt = null;
   // analogicznie con.close();
```

# NAWIĄZANIE POŁĄCZENIA

Istnieją dwie metody nawiązania połączenia z bazą danych:

za pomocą klasy DriverManager:

```
String url = "jdbc:odbc:bazadanych";
Connection con = DriverManager.getConnection(url, "login", "haslo");
```

• za pomocą klasy **DataSource** i usługi JNDI:

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/MojaDB");
Connection con = ds.getConnection("myLogin", "myPassword");
```

# DriverManager

DriverManager jest tradycyjną warstwą zarządzającą JDBC pomiędzy użytkownikiem a sterownikiem. Aktualną listę dostępnych sterowników można uzyskać za pomocą metody: Enumeration

DriverManager.getDrivers().

Aby załadować dodatkowy sterownik należy skorzystać z metody: Class Class.forName(String) podając jako argument klasę ze sterownikiem np:

```
Class.forName("jdbc.odbc.JdbcOdbcDriver"); // protokół jdbc:odbcClass.forName("com.mysql.jdbc.Driver"); //protokół jdbc:mysql..
```

### DataSource

DataSource reprezentuje źródło danych. Zawiera informacje identyfikujące i opisujące dane. Obiekt DataSource współpracuje z technologią Java Naming and Directory Interface (JNDI), jest tworzony i zarządzany niezależnie od używającej go aplikacji. Korzystanie ze źródła danych zarejestrowanego w JNDI zapewnia:

- brak bezpośredniego odwołania do sterownika przez aplikację,
- umożliwia implementację grupowania połączeń (pooling) oraz rozproszonych transakcji.

Te cechy sprawiają, że korzystanie z klasy **DataSource** jest zalecaną metodą tworzenia połączenia z bazą danych, szczególnie w przypadku dużych aplikacji rozproszonych.

## PRZESYŁANIE ZAPYTAŃ

Do przesyłania zapytań do bazy danych służą obiekty klasy:

- Statement typowe pytania (bezparametrowe),
- PreparedStatement prekompilowany pytania zawierające parametry wejściowe,
- CallableStatement procedury zapisane w bazie danych.

Obiekt **Statement** tworzy się w ramach nawiązanego wcześniej połączenia:

```
Connection con = DriverManager.getConnection(url, login, pass);
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO table(name, price) VALUE 'ser', 2.0");
```

### PRZESYŁANIE ZAPYTAŃ

Do przesyłania zapytań służą metody:

- executeQuery pytania zwracające dane: SELECT,
- executeUpdate pytania zmieniające dane: INSERT, UPDATE,
   CREATE TABLE, ....
- execute dowolne zapytania. Dzięki tej instrukcji można wykonać sekwencje pytań, przekazać i odebrać dodatkowe parametry.

W ramach jednego obiektu Statement można wykonać sekwencyjnie kilka zapytań. Po zakończeniu używania obiektu zaleca się wywołanie metody close().

### ODBIERANIE WYNIKÓW

Wyniki zwrócone w wyniku wykonania zapytania są dostępne poprzez obiekt typu **ResultSet**. Przykład:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM table");
while (rs.next()) {
    // odebranie i wypisanie wyników w bieżącym rekordzie
    int i = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
    System.out.println("Rekord = " + i + " " + s + " " + f);
}
```

## ODBIERANIE WYNIKÓW

getObject	getURL	getCharacterStream	getRef	getArray	getBlob	getClob	getBinaryStream	getUnicodeStream	getAsciiStream	getTimestamp	getTime	getDate	getBytes	getString	getBoolean	getBigDecimal	getDouble	getFloat	getLong	getInt	getShort	getByte	
ဝ္	뒤	ß	Re	1 ₹	B10	ĮΩ	먎	G,	As	밁	밁	Ď	Вy	St.	Bo	<u>8</u>	å	Ξ	8	lint.	ďΣ	Вy	
ec.	P	8	_	œy.	۱ĕ	Ď.	Į,š	8.	₽;	nes	ne	te	tes	Œ.	읂	ĝ	દ	ět	8		ä	æ	
14		8		`			52	6	St.	č			"	•	ĝ	ĝ.	6						
		52					Te.	ğ	69	æ						ma.							
		d.					1 1 1	ea.	¤							-							
		am						P															
ы														н	ы	ы	ы	ы	н	ы	ы	×	TINYINT
ы	П				$\vdash$	$\overline{}$							-	ы	н	×	н	н	н	н	X	н	SMALLINT
×														ы	н	×	н	н	н	×	н	н	INTEGER
н														н	н	н	н	н	×	н	н	н	BIGINT
×														н	ы	×	ы	×	ы	×	×	н	REAL
ы														ы	н	×	×	н	н	н	н	н	FLOAT
ы														н	н	н	×	н	н	н	н	н	DOUBLE
×														ы	×	×	×	×	×	×	×	×	DECIMAL
×														×	×	×	×	×	×	×	×	×	NUMERIC
×														м	X	м	×	×	н	×	×	н	BIT
×															X	×	×	×	×	×	×	×	BOOLEAN
×		×						×	×	×	×	×		X	×	×	×	×	×	×	×	×	CHAR
×		н						×	м	×	×	м		X	н	×	н	н	н	н	н	м	VARCHAR
×		×						×	×	×	×	×		×	×	×	×	×	×	×	×	×	LONGVARCHAR
×		×					×	×	×				×	×									BINARY
×		×					×	×	×				×	м									VARBINARY
×		×					×	×	×				×	×									LONGVARBINARY
×										×		×		×									DATE
×										×	×			м									TIME
×										×	×	м		м									TIMESTAMP
×						×																	CLOB
×					×																		BLOB
×				×																			ARRAY
×			×																				REF
×																							STRUCT
×	×													×									DATALINK
×																							JAVA OBJECT

### **HSQLDB**

HSQLDB to system zarządzania relacyjnymi bazami danych w całości napisany w Javie (open source).

Strona domowa projektu: http://www.hsqldb.org.

#### Niektóre własności:

- obsługa SQL'a,
- obsługa transakcji (COMMIT, ROLLBACK, SAVEPOINT), zdalnych procedur i funkcji (mogą być pisane w Javie), triggerów itp.
- możliwość dołączania do programów i appletów. Działanie w trybie read-only,
- rozmiar tekstowych i binarnych danych ograniczony przez rozmiar pamięci.

### HSQLDB - SERWER

- HyperSQL Server tryb preferowany. Klasa org.hsqldb.server.Server,
- HyperSQL WebServer używany, jeśli serwer może używać tylko protokołu HTTP lub HTTPS. W przeciwnym razie niezalecany. Klasa org.hsqldb.server.WebServer,
- HyperSQL Servlet używa tego samego protokołu co WebServer.
   Wymaga osobnego kontenera serwletów (np. Tomcat).
- Stand-alone "serwer" uruchamiany w ramach procesu, w którym nawiązujemy połączenie z bazą.

Wszystkie tryby pracy serwera umożliwiają korzystanie z JDBC.

## HSQLDB - POŁĄCZENIE

```
try {
     Class.forName("org.hsqldb.jdbc.JDBCDriver" );
} catch (Exception e) {
     System.err.println("ERROR: failed to load HSQLDB JDBC driver.");
     e.printStackTrace();
     return;
Connection c =
DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", "");
Connection c =
DriverManager.getConnection("jdbc:hsqldb:http://localhost/xdb", "SA", "");
Connection c =
DriverManager.getConnection("jdbc:hsqldb:file:/opt/db/testdb;shutdown=true")
", "SA", "");
```

### HSQLDB stand-alone

W trybie *stand-alone* "serwer" bazy danych działa w ramach tej samej wirtualnej maszyny Javy, co korzystający z niego program "kliencki".

Przykład uruchomienia bazy:

Niewielkie bazy danych mogą być uruchamiane do pracy w pamięci operacyjnej komputera:

W obecnej wersji HSQLDB istnieje możliwość jednoczesnego używania wielu "serwerów" baz danych działających w trybie *stand-alone*.

### PODSUMOWANIE

Podstawowe klasy służące do interakcji z systemami bazodanowymi są udostępniane za pośrednictwem obiektów: Connection, Statement i ResultSet. Wszystkie te klasy należą do pakietu java.sql. Klasy rozszerzające funkcjonalność JDBC (np. DataSource) znajdują się w pakiecie javax.sql.

Korzystanie z interfejsu JDBC umożliwia jednolity sposób dostępu do różnych systemów bazodanowych. Jako przykład przedstawiono bazę HSQLDB. Jej ważną zaletą jest możliwość działania w ramach jednej Wirtualnej Maszyny Javy wraz z programem klienckim.

# DZIĘKUJĘ ZA UWAGĘ

## REFLEKSJE

#### ZAGADNIENIA:

- Klasy,
- Atrybuty i metody,
- Dynamiczne klasy proxy.

#### MATERIAŁY:

http://docs.oracle.com/javase/tutorial/reflect/

### WPROWADZENIE

Programowanie refleksyjne polega na dynamicznym korzystaniu ze struktur języka programowania, które nie musiały być zdeterminowane w momencie tworzenia oprogramowania.

Najważniejsze klasy języka Java, które umożliwiają programowanie refleksyjne to **Class**, **Field**, **Method**, **Array**, **Constructor**. Są one zgrupowane w pakietach **java.lang** i **java.lang.reflect**.

#### KLASY

Każdy obiekt w Javie jest instancją klasy **Object**. Każdy typ (obiektowy, prymitywny, tablicowy itp.) jest reprezentowany przez insta<mark>c</mark>ję klasy Class, którą uzyskujemy za pomocą getClass(). import java.util.HashSet; import java.util.Set; enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY public class ReflectionExample { public static void main(String[] args){ Class c; c = "foo".getClass(); System. out.println(c.getName()); // wypisuje java.lang.String

#### KLASY

```
c = "foo".getClass();
System.out.println(c.getName()); // wypisuje java.lang.String
c = System.out.getClass();
System. out.println(c.getName()); // wypisuje java.io.PrintStream
c = Day. SUNDAY.getClass();
System. out. println(c.getName()); // wypisuje Day
byte[] bytes = new byte[1024];
c = bytes.getClass();
System.out.println(c.getName()); // wypisuje [B
Set<String> s = new HashSet<String>();
c = s.getClass();
System.out.println(c.getName()); // wypisuje java.util.HashSet
```

### class

Jeśli nie mamy obiektu (instancji klasy) możemy użyć atrybutu class.

```
c = java.io.PrintStream.class; // java.io.PrintStream
c = int[][].class; // [[[[I
c = boolean.class; // boolean
```

Ten sposób jest szczególnie użyteczny w przypadku typów prymitywnych:

```
boolean b;
Class c = b.getClass(); // compile-time error
```

### Class.forName()

Obiekt Class można także otrzymać znając jego nazwę:

```
Class cMyClass = Class.forName("pl.edu.uj.fais.java.Klasa");
Class cDoubleArray = Class.forName("[D");
Class cStringArray = Class.forName("[[Ljava.lang.String;");
```

W przypadku podania niepoprawnej nazwy klasy zwracany jest wyjątek **ClassNotFoundException**.

#### KLASY

#### Wybrane metody klasy Class:

- static Class<?> forName(String className)
- Constructor<T> getConstructor(Class<?>... parameterTypes)
- Constructor<?>[] getConstructors()
- Field getField(String name)
- Field[] getFields()
- Class<?>[] getInterfaces()
- Method getMethod(String name, Class<?>... parameterTypes)
- Method[] getMethods()
- int getModifiers()
- Class<? super T> getSuperclass()
- TypeVariable<Class<T>>[] getTypeParameters()
- boolean isArray()
- boolean isInterface()
- boolean isPrimitive()
- T newInstance()

### KLASY

```
class C1{}
class C2 extends C1{}
...
C1 o1 = new C1(); // rownowazne C1.class.newInstance()
C2 o2 = new C2();

o1.getClass().isAssignableFrom(o2.getClass()); // true
o2.getClass().isAssignableFrom(o1.getClass()); // false
o1.getClass().isInstance(o2); // true
o2.getClass().isInstance(o1); // false
...
```

#### ATRYBUTY

Atrybut jest reprezentowany przez instancję klasy **Field**. Wybrane metody:

- Object get(Object obj) // zwraca wartość atrybutu w obiekcie obj
- int getInt(Object obj) // zwraca wartość atrybutu typu int
- int getModifiers() // modyfikatory dostępu: public, private, ...
- Class<?> getType() // klasa reprezentująca typ atrybutu
- void set(Object obj, Object value) // ustawia wartosć atrybutu w obiekcie obj
- void setInt(Object obj, int i) // ustawia wartosć atrybutu typu int

### ATRYBUTY

```
import java.lang.reflect.Field;
public class FieldExample {
   public static String s;
   public int i;
   public static void main(String[] args) throws Exception{
        FieldExample fex = new FieldExample();
        Field f;
        f = FieldExample.class.getField("s");
        f.get(null)); // zwróci null bo s nie zainicjowane
        f.set(null, "Ala"); // FieldExamle.s = "Ala"
        f = fex.getClass().getField("i");
        f.set(fex, 10); // fex.i = 10
```

#### METODY

Metoda jest reprezentowana przez instancję klasy **Method**. Wybrane metody:

- Class<?>[] getExceptionTypes() // zwraca clasy reprezentujące zadeklarowane, zwracane wyjątki
- int getModifiers() // private, public, static, ...
- Object invoke(Object obj, Object... args) // wywołuje metodę na rzecz obiektu obj. Argumenty wywołania znajdują się w tablicy args. Watrość zwracana jest wynikiem działania wywołanej metody.
- boolean isVarArgs() // czy metoda ma nieokreśloną liczbę argumentów

## invoke()

```
Przykład:
```

metoda **invoke** może zwrócić kilka rodzajów wyjątków związanych z dostępem do metody i zgodnością typów argumentów. Wady wynikające z używania invoke:

- brak kontroli (w trakcie kompilacji) typów przekazywanych parametrów,
- ograniczenie obsługi wyjątków do Throwable.

Rozwiązaniem tych problemów może być tzw. Dynamic Proxy Class.

### DYNAMIC PROXY





klasa Proxy powinna być tworzona dynamicznie.

Klasa **Proxy** udostępnia metody statyczne służące do tworzenia tzw. dynamicznych klas proxy oraz ich instancji. Utworzenie proxy dla określonego interfejsu (np. **MyInterface**):

lub alternatywnie:

Stworzono obiekt **mi**, który "z zewnątrz" wygląda jak klasa implementująca MyInterface, natomiast obsługa metod będzie w rzeczywistości realizowana przez **handler**.

Obiekt handler <u>nie musi</u> implementować MyInterface!

Musi za to implementować interfejs InvocationHandler, czyli metodę:

Object invoke(Object proxy, Method method, Object[] args)

Jak to działa? Wszystkie wywołania metod na obiekcie **mi** są przekierowane do metody **invoke** obiektu **handler**, przy czym pierwszym parametrem jest obiekt **proxy**.

```
Pisarz obj = new PisarzImpl();
Method m = obj.getClass().getMethod(
        "pisz",
        new Class[] {String.class});
m.invoke(obj, new Object[]{"hello world"});
                                            class PisarzImpl implements Pisarz {
                                                 public void pisz(String s) {
                                                 interface Pisarz {
                                                     public void pisz(String s);
```

```
Pisarz p = (Pisarz) Proxy.newProxyInstance(
    Pisarz.class.getClassLoader(),
    new Class[] { Pisarz.class },
    new MyHandler());
p.pisz("hello world");
                                            class PisarzImpl implements Pisarz {
                                                 public void pisz(String s) {
class MyHandler{
  private Pisarz obj = new PisarzImpl();
  public static Object invoke(Object proxy,
                                                 interface Pisarz {
                                                     public void pisz(String s);
             Method m, Object[] args)){
    return m.invoke(obj, args);
```

# DZIĘKUJĘ ZA UWAGĘ

### PROGRAMOWANIE FUNKCYJNE

#### ZAGADNIENIA:

- Funkcje
- Wyrażenia lambda
- Strumienie

#### MATERIAŁY:

https://flyingbytes.github.io/.../Java8-Part0.html

http://www.oracle.com/technetwork/.../ma14-java-se-8-streams-2177646.html

```
Thread t = new Thread(new Runnable() {
    public void run() {
        System.out.println("Hello");
});
t.start();
Thread t = new Thread(()->System.out.println("Hello"));
t.start();
Runnable r = () -> System.out.println("Hello");
Thread t = new Thread(r);
t.start();
```

Przykłady definicji funkcji:

```
() -> {System.out.println("Hello");}// java.lang.Runnable
x -> {System.out.println(x);} // java.util.function.Consumer<T>
() -> 7 // java.util.function.Supplier<T>
Runnable r = ()->{System.out.println("Hello");};
Consumer<Integer> cons = x -> {System.out.println(x);};
Supplier<Integer> sup = () -> 7;
```

Inne przykłady:

```
Function<Integer, Integer> inc = x -> x+1;
```

BiFunction<Integer, Integer> sum =  $(x,y) \rightarrow x+y$ ;

Każdą funkcję wieloargumentową można zapisać jako złożenie funkcji jednoargumentowych:

Function<Integer, Function<Integer, Integer>> sum1 = x->y->x+y;

Funkcja może być argumentem innej funkcji:

BiFunction<Function<Integer, Integer>, Integer> fx = (f, x) -> f.apply(x);

W programowaniu funkcyjnym funkcje powinny działać zawsze tak samo i nie powinny powodować żednych "efektów ubocznych". Podstwowe zasady:

- wszystkie zmienne są stałe (final),
- nie ma zmiennych globalnych,
- funkcje mogą być zarówno argumentami innych funkcji oraz mogą być zwracane jako wynik innych funkcji.

## java.util.function

```
Function<T, R>
  R apply(T t);
  Function<T, V> andThen(Function<? super R, ? extends V> after);
  Function < V, R > compose(Function <? super V, ? extends T > before);
metoda apply() zwraca wartość funkcji, natomiast andThen() oraz
compose() zwracają funkcję będącą złożeniem dwóch funkcji.
Consumer<T>
  void accept(T t);
  Consumer<T> andThen(Consumer<? super T> after);
metoda accept() wykonuje funkcję, natomiast andThen() zwracaja
funkcję będącą złożeniem dwóch funkcji.
Supplier<R>
  R get();
```

## java.util.function

Inne (wybrane) interfejsy:

```
Predicate<T>
  boolean test(T t);
  Predicate<T> and(Predicate<? super T>, other);
  Predicate<T> or(Predicate<? super T>, other);
  Predicate<T> negate();
```

Predicate to funkcja, której wynik jest typu boolean

public interface UnaryOperator<T> extends Function<T,T>

UnaryOperator to funkcja, której wynik jest tego samego typu co argument

#### STRUMIENIE

Strumienie java.util.stream.Stream reprezentuje strumień danych (obiektów) i jest odpowiednikiem kolekcji (java.util.Collection) w programowaniu obiektowym

```
Stream<String> objectStream = Stream.of("Ala", "Ola");
```

```
String[] array = {"Ala", "Ola"};
Stream<String> arrayStream = Arrays.stream(array);
```

W interfejsie **Collection** istnieje metoda **stream()** lub **parallelStream()** przekształcająca kolekcję w strumień.

#### STRUMIENIE

Przetwarzanie danych w strumieniach opiera się na przekształcaniu strumieni I stosowaniu odpowiednich funkcji

```
Stream.of(1, 2, 3) // tworzymy strumień zawierający 1 2 3
.map(num -> num * num) // nowy strumień, w którym element
// zastępujemy jego drugą potęgą (1 4 9)
.forEach(System.out::println); // dla każdego elementu strumienia
// wywołujemy funkcję która
ekran

System.out::println – "wskaźnik" do funkcji. Równoważnie:
num -> System.out.println(num)
```

#### STRUMIENIE

### STRUMIENIE

```
String sentence = Stream.of("Hello", "world")
.collect(Collectors.joining(" "));

System.out.println(sentence); // Hello world

Integer sum = Stream.of(1, 2, 3)
.reduce(0, Integer::sum);

System.out.println(sum); // 6
```



# DZIĘKUJĘ ZA UWAGĘ

# JAVA NATIVE INTERFACE

#### ZAGADNIENIA:

- wprowadzenie
- przykład HelloWorld
- podstawy JNI (przekazywanie zmiennych, stringów, obiektów)

#### MATERIAŁY:

http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

http://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html

#### WPROWADZENIE

Java Native Interface umożliwia:

- korzystanie z bibliotek systemowych (napisanych np. w C/C++) wewnątrz programu w Javie.
- korzystanie z obiektów Javy wewnątrz programów w innych językach programowania (np. C/C++).

Najczęściej JNI wykorzystuje się do realizacji niskopoziomowej komunikacji IO (np. obsługa interfejsów USB, bluetooth), która nie może być zrealizowana z poziomu JVM, gdyż nie ma ona bezpośredniego dostępu do sprzętu.

### WPROWADZENIE

Najważniejsze ograniczenia i wady JNI.

- brak przenośności kodu pomiędzy różnymi platformami,
- utrudnione debugowanie drobne błędy JNI mogą
   "niedeterministycznie" destabilizować pracę JVM.
- brak garbage collectora,
- problemy z konwersją danych (np. kodowanie stringów).

# HELLO WORLD

Stworzenie aplikacji używającej JNI składa się typowo z następujących kroków:

- napisanie programu w Javie
- skompilowanie programu I wygenerowanie plików nagłówkowych dla metod natywnych
- implementacja metod natywnych (np. w C/C++)
- kompilacja metod natywnych do biblioteki
- uruchomienie programu.

# PROGRAM W JAVIE

```
public class JNIHello {
   static {
          System. loadLibrary("hello"); // nazwa biblioteki z kodem
                     // natywnym, moze zalezec od systemu operacyjnego
   }
   // deklaracja metody natywnej (ktora znajduje się w bibliotece)
   private native void sayHello();
    public static void main(String[] args) {
        JNIHello h = new JNIHello();
        h.sayHello(); // wywolanie metody natywnej
```

# KOMPILACJA

Kompilacja programu:

javac JNIHello.java

Generowanie plików nagłówkowych dla metod natywnych:

javah JNIHello

Do generowania używamy skompilowanego pliku z rozszerzeniem class.

# PLIK NAGŁÓWKOWY

```
#include <jni.h>
/* Header for class JNIHello */
#ifndef _Included_JNIHello
    #define _Included_JNIHello
    #ifdef __cplusplus
        extern "C" {
    #endif
     * Class: JNIHello
     * Method: sayHello
     * Signature: ()V
     */
     JNIEXPORT void JNICALL Java_JNIHello_sayHello(JNIEnv *, jobject);
     #ifdef __cplusplus
    #endif
#endif
```

# IMPLEMENTACJA

```
#include "JNIHello.h"
#include <stdio.h>
JNIEXPORT void JNICALL Java_JNIHello_sayHello(JNIEnv *env, jobject obj){
   printf("Hello World!\n");
   return;
}
```

# KOMPILACJA BIBLIOTEKI

#### OSX:

```
gcc -I /System/Library/Frameworks/JavaVM.framework/Headers
  -c hello.c
gcc -I /System/Library/Frameworks/JavaVM.framework/Headers
```

-shared -dynamiclib -o libhello.jnilib hello.o

#### Linux:

```
gcc -I /etc-java-config-2/current-system-vm/include
-I /etc-java-config-2/current-system-vm/include/linux -fPIC
-shared -o libhello.so hello.c
```

# URUCHOMIENIE

Kompilacja biblioteki

OSX:

java JNIHello

Linux:

java -Djava.library.path=. JNIHello



# PODSTAWY JNI

JNI definiuje następujące typy natywne:

- jint, jbyte, jshort, jlong, jfloat, jdouble, jchar, jboolean.
- jobject, jclass, jstring, jthrowable
- jarray (jintArray, jbyteArray, jshortArray, jlongArray, jfloatArray, jdoubleArray, jcharArray and jbooleanArray oraz jobjectArray).

Typy natywne mogą być przekazywane jako argumenty wywołania oraz mogą być zwracane. Jednak funkcje natywne operują na swoich natywnych typach (np int[], char\*). Dlatego wymagana jest konwersja.

## TYPY PRYMITYWNE

```
private native double add(double d1, double d2);
public static void main(String[] args){
    JNIExamples ex = new JNIExamples();
   System. out.println(ex.add(12.5, 5.3));
JNIEXPORT jdouble JNICALL Java_JNIExamples_add(JNIEnv *env, jobject obj,
jdouble d1, jdouble d2){
    printf("Otrzymade argumenty %f; %f \n",d1, d2);
    jdouble res = d1 + d2;
    return res;
```

#### STRINGI

```
private native String sayHello(String msg);
JNIEXPORT jstring JNICALL Java_JNIExamples_sayHello(JNIEnv *env, jobject
obj, jstring jin){
   // konwersja: (jstring) --> (char*)
    const char *in = (*env)->GetStringUTFChars(env, jin, NULL);
    if (NULL == in) return NULL;
   // dzialanie:
    printf("odebralem %s\n", in);
   // zwolnienie zasobow
    (*env)->ReleaseStringUTFChars(env, jin, in); // release resources
   // przygotowanie odpowiedzi
    char out[] = "from C";
   // konwersja
    return (*env)->NewStringUTF(env, out);
```

# STRINGI (UTF-8)

```
// UTF-8 String --> null-terminated char-array
const char * GetStringUTFChars(JNIEnv *env, jstring string, jboolean
*isCopy);
// Informije JVM ze kod natywny nie bedzie potrzebowal utf
void ReleaseStringUTFChars(JNIEnv *env, jstring string, const char *utf);
// Tworzy java.lang.String ze znakow
jstring NewStringUTF(JNIEnv *env, const char *bytes);
// zwraca długosc stringu
jsize GetStringUTFLength(JNIEnv *env, jstring string);
// w buforze zostanie zapisany wskazany fragment str
void GetStringUTFRegion(JNIEnv *env, jstring str, jsize start, jsize
length, char *buf);
```

# STRINGI (UNICODE)

```
// Unicode String --> jchar*
const jchar * GetStringChars(JNIEnv *env, jstring string, jboolean
*isCopy);
void ReleaseStringChars(JNIEnv *env, jstring string, const jchar *chars);
jstring NewString(JNIEnv *env, const jchar *unicodeChars, jsize length);
jsize GetStringLength(JNIEnv *env, jstring string);
void GetStringRegion(JNIEnv *env, jstring str, jsize start, jsize length,
jchar *buf);
```

### TABLICE

```
private native int[] swap(int[] numbers);
  int[] in = \{1,2\};
   int[] out = ex.swap(in);
  for(int i: out)
    System.out.println(i);
JNIEXPORT jintArray JNICALL Java_JNIExamples_swap(JNIEnv *env,
jobject obj, jintArray ia){
   jint *inArray = (*env)->GetIntArrayElements(env, ia, NULL);
   if (NULL == inArray) return NULL;
    jint outArray[] = {inArray[1], inArray[0]}; // dzialanie
    (*env)->ReleaseIntArrayElements(env, ia, inArray, 0); // zwolnienie
   jintArray out = (*env)->NewIntArray(env, 2); // wynik
   if (NULL == out) return NULL;
    (*env)->SetIntArrayRegion(env, out, 0 , 2, outArray); // kopiowanie
   return out;
```

# DOSTĘP DO OBIEKTÓW

```
public class JNIExamples {
    static {
        System.loadLibrary("examples");
   private int number = 88;
   private String message = "Hello from Java";
   private native void modifyVariables();
   public static void main(String[] args) {
        JNIExamples ex = new JNIExamples();
        ex.modifyVariables();
        System.out.println("In Java, int is " + ex.number);
        System.out.println("In Java, String is " + ex.message);
```

# DOSTĘP DO OBIEKTÓW

```
JNIEXPORT void JNICALL Java_JNIExamples_modifyVariables(JNIEnv *env,
jobject thisObj) {
   // referencja do obiektu
    jclass thisClass = (*env)->GetObjectClass(env, thisObj);
   // pobranie pola int
    jfieldID fidNumber = (*env)->GetFieldID(env, thisClass, "number", "I");
   if (NULL == fidNumber) return;
   // pobranie wartosci
   jint number = (*env)->GetIntField(env, thisObj, fidNumber);
   printf("In C, the int is %d\n", number);
   // zmiana wartosci
   number = 99;
    (*env)->SetIntField(env, thisObj, fidNumber, number);
```

# DOSTĘP DO OBIEKTÓW

```
// pobranie pola String
jfieldID fidMessage = (*env)->GetFieldID(env, thisClass, "message",
                                                 "Ljava/lang/String;");
if (NULL == fidMessage) return;
// pobranie zawartosci
jstring message = (*env)->GetObjectField(env, thisObj, fidMessage);
// konwersia
const char *cStr = (*env)->GetStringUTFChars(env, message, NULL);
if (NULL == cStr) return;
printf("In C, the string is %s\n", cStr);
(*env)->ReleaseStringUTFChars(env, message, cStr); // porzadki
// nowa wartosc
message = (*env)->NewStringUTF(env, "Hello from C");
if (NULL == message) return;
// modyfikacja - podstawienie nowej wartosci
(*env)->SetObjectField(env, thisObj, fidMessage, message);
```

# INNE MOŻLIWOŚCI

- zmiana zmiennych statycznych,
- wywoływanie metod z kodu natywnego,
- tworzenie obiektów Javy (i tablic) w kodzie natywnym,
- zarządzanie referencjami.

# DZIĘKUJĘ ZA UWAGĘ

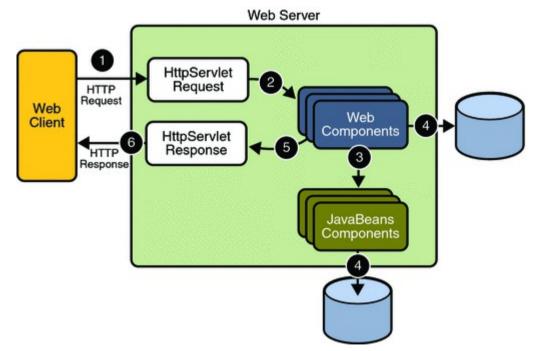
# **Elementy JEE**

- 1. Wprowadzenie.
- 2. Prerekwizyty.
- 3. Pierwszy servlet.
- obsługa parametrów żądań
- 4. JavaServer Pages.



# **Java Enterprice Edition**

Java Enterprice Edition (JEE) jest rozszerzeniem Java Standard Edition (JSE) o usługi wykorzystywane w tzw. aplikacjach biznesowych. Jednym z najczęściej wykorzystywanych elementów JEE są usługi związane z aplikacjami www.



http://java.sun.com/javaee/5/docs/tutorial/doc/

# Prerekwizyty

- Java SDK
- Apache Tomcat http://tomcat.apache.org/,
- ewentualnie plugin do Eclipse

http://www.eclipsetotale.com/tomcatPlugin.html#A3.

### Pierwszy serwlet

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet{
   public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.print("Hello world");
      out.close();
```

### Pierwszy serwlet - web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"</pre>
          version="2.5"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
             http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd">
  <servlet>
    <servlet-name>first</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>first</servlet-name>
    <url-pattern>/firstUrl</url-pattern>
  </servlet-mapping>
</web-app>
```



```
WEB-INF
classes
FirstServlet.class
lib
web.xml
```

Tak przygotowaną strukturę katalogów i plików umieszczamy w archiwum HelloTomcat.war i wgrywamy na serwer Tomcat. Uruchomienie przykładu: http://localhost:8080/HelloTomcat/firstUrl.



## Pierwszy serwlet - obsługa żądań

Żądania, przekazywane do serwletu, w zależności od formy są obsługiwane przez jedną z dwóch metod:

oraz

Z tego powodu serwlet chcący reagować na oba rodzaje wywołań powinien obsługiwać obie metody.



## Pierwszy serwlet - obsługa żądań

Dostęp do danych przekazanych w żądaniu (argumenty GET, POST, ustawienia Cookies, identyfikatory sesji) uzyskujemy za pośrednictwem obiektu HttpServletRequest.

Obiekt HttpServletResponse response jest przez serwlet używany do skonstruowania odpowiedzi.



### Pierwszy serwlet - obsługa żądań

```
public void doGet(...) throws ...
   performTask(request, response);
public void doPost(...) throws ... {
   performTask(request, response);
public void performTask(HttpServletRequest request,
                       HttpServletResponse response) {
   HttpSession session = request.getSession();
   Object obj = session.getAttribute("lp");
   if (obj==null)
      obi = "0";
   session.setAttribute("lp",
       String.valueOf(Integer.parseInt(obj.toString()) + 1));
   response.setContentType("text/html");
   PrintWriter out;
   try {
      out = response.getWriter();
      out.println("Hello world: " + obj.toString());
   } catch (IOException e) {
      e.printStackTrace();
```

9

# JavaServer Pages

Aby ułatwić programistom tworzenie stron internetowych wprowadzono technologię JSP, która przypomina inne, powszechnie stosowane języki skryptowe służące do generowania dokumentów HTML (ASP, PHP, ...). Dokument JSP jest zapisany w formacie XML.

### Pierwszy skrypt JSP

```
<%@ page language = "java" %>
<%@ page import = "java.util.*" %>
<%@ page contentType = "text/html" %>
<ht.ml>
  <title>JSP Demo</title>
  <body bgcolor="white" text="red">
    <H1>
    < %
     if (Calendar.getInstance().get(Calendar.AM PM) == Calendar.AM) {
    응>
      Milego poranka!
    <% }else{ %>
      Milego popoludnia!
    <% } %>
    </H1>
    <H3>
    Aktualny czas: <% out.println(new java.util.Date()); %>
    </H3>
  </body>
</html>
```



# **Uruchomienie strony JSP**

Skrypt umieszczamy np. w głównym katalogu aplikacji:

```
WEB-INF
classes
lib
web.xml
hello.jsp
```

Dostęp do niego odbywa się poprzez adres:

http://localhost:8080/HelloTomcat/hello.jsp

W rzeczywistości na podstawie skryptu jsp tworzona jest automatycznie obiekt Javy (przypominający servlet), który generuje opisaną w skrypcie dokument.

Aby nie przeplatać kodu Javy z elementami HTML'a używa się specjalnych tagów:

# Przykład JSP

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!-- import biblioteki StandardTagLibrary -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!-- import własnej biblioteki tagów -->
<%@ taglib uri="/WEB-INF/functions.tld" prefix="f" %>
<ht.ml>
<head>
  <title>Daty</title>
</head>
<body bgcolor="white">
<!-- użycie własnego obiektu w ramach strony -->
<jsp:useBean id="locales" scope="application" class="mypkg.MyLocales"/>
```

# Przykład JSP

```
<form name="localeForm" action="date.jsp" method="post">
<c:set var="selectedLocaleString" value="${param.locale}" />
<c:set var="selectedFlag" value="${!empty selectedLocaleString}" />
<b>Lokalizacja:</b>
<select name=locale>
<c:forEach var="localeString" items="${locales.localeNames}" >
  <c:choose>
   <c:when test="${selectedFlag}">
      <c:choose>
          <c:when test="${f:equals(selectedLocaleString, localeString)}">
          <option selected>${localeString}</option>
         </c:when>
          <c:otherwise>
             <option>${localeString}</option>
          </c:otherwise>
      </c:choose>
   </c:when>
   <c:otherwise>
      <option>${localeString}</option>
   </c:otherwise>
  </c:choose>
</c:forEach>
</select>
```

# Przykład JSP

```
<input type="submit" name="Submit" value="Pokaz date">
</form>
<c:if test="${selectedFlag}" >
  <jsp:setProperty name="locales"</pre>
                    property="selectedLocaleString"
                    value="${selectedLocaleString}" />
  <jsp:useBean id="date" class="mypkg.MyDate"/>
  <jsp:setProperty name="date"</pre>
                    property="locale"
                    value="${locales.selectedLocale}"/>
  <b>Data: </b>${date.date}
</c:if>
</body>
</html>
```

### Własne funkcje

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"</pre>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                       web-jsptaglibrary 2 0.xsd"
                       version="2.0">
  <description>Definicja funkcji</description>
  <tlib-version>1.0</tlib-version>
  <short-name>FunctionTagLibrary</short-name>
  <uri>/FunctionLibrary</uri>
  <function>
    <name>equals</name>
    <function-class>mypkg.MyLocales</function-class>
    <function-signature>
      boolean equals ( java.lang.String, java.lang.String )
    </function-signature>
  </function>
</taglib>
```



#### **Bean MyLocales**

```
package mypkg;
import java.util.*;
import java.text.DateFormat;
public class MyLocales {
    ArrayList<String> localeNames;
    HashMap<String, Locale> locales;
    Locale selectedLocale:
    String selectedLocaleString;
    public MyLocales() {
        locales = new HashMap<String, Locale>();
        localeNames = new ArrayList<String>();
        Locale[] list = DateFormat.getAvailableLocales();
        for (int i = 0; i < list.length; i++) {</pre>
            locales.put(list[i].getDisplayName(),
                        list[i]);
            localeNames.add(list[i].getDisplayName());
        Collections.sort(localeNames);
        selectedLocale = null; selectedLocaleString = null;
```

#### **Bean MyLocales**

```
public static boolean equals (
    String 11,
    String 12) {
    return 11.equals(12);
public Collection getLocaleNames() {
    return localeNames;
public void setSelectedLocaleString(String displayName) {
    this.selectedLocaleString = displayName;
    this.selectedLocale = (Locale) locales.get(displayName);
public Locale getSelectedLocale() {
    return selectedLocale;
public String getSelectedLocaleString() {
    return selectedLocaleString;
```

#### **Bean MyDate**

```
package mypkq;
import java.text.DateFormat;
import java.util.*;
public class MyDate {
    Date today;
    DateFormat dateFormatter;
    public MyDate() {
        today = new Date();
    public String getDate() {
        return dateFormatter.format(today);
    public void setLocale(Locale 1) {
        dateFormatter = DateFormat.getDateInstance(DateFormat.FULL, 1);
```

### Struktura przykładu

Skrypt umieszczamy np. w głównym katalogu aplikacji:

```
WEB-INF
classes
mypkg
MyDate.class
MyLocale.class
lib
jstl.jar
standard.jar
web.xml
functions.tld
date.jsp
```

Adres: http://localhost:8080/HelloTomcat/date.jsp

# Podsumowanie

Podstawową zaletą technologii servletów jest możliwość tworzenia serwisów internetowych z wykorzystaniem całej infrastruktury dostarczanej przez język programowania Java wraz z wszelkimi jego rozszerzeniami. Głównym obszarem zastosowaniem tego rodzaju rozwiązań są aplikacje webowe wymagające dużej skalowalności ze względu na używane zasoby. Kontener serwletów jest wtedy składnikiem odpowiedzialnym za interakcje z użytkownikiem poprzez interfejs www.