Java - środowisko programisty

JDK = Java Development Kit (np JDK1.4.2)
zawiera programy do kompilowania i uruchamiania programów w Javie

Podstawowe środowisko programistyczne:

```
javac mojaKlasa.java
  # kompilator; (jest szybszy kompilator: jikes !!!)
  # powstanie plik mojaKlasa.class zawierający B-kod

java mojaKlasa
  # uruchomienie maszyny wirtualnej Javy (JVM)
  # która interpretuje B-kod mojaKlasa.class

appletviewer plik.html
  # zastępuje przeglądarkę; pozwala uruchamiać aplety Javy

jdb mojaKlasa
  # tekstowy debugger; aby go użyć kompiluj z opcją -g

# inne:
Jacl i tclBlend # eksperymenty z obiektami Javy w konsoli/skryptach tcl-a
```

Java - najprostszy program

```
// plik "test_CCC.java" (MUSI się tak nazywać)

class CCC {
  public int i, j;
  public void pisz() {
    System.out.println("CCC: i="+i+", j="+j+", this="+this);
      // System - klasa pakietu java.lang (domyślnego)
  }
}

public class test_CCC {
    // od metody main zaczyna się wykonanie aplikacji Javy ...
  public static void main(String args[]) {
    CCC c= new CCC();
      // tworzymy obiekt klasy CCC; "new" zwraca ref do obiektu
      // "c" to zmienna referencyjna (odnosnik) do obiektu!
    c.pisz();
    c.i=123; c.j=321;
    c.pisz();
  }
}
```

Java - cechy charakterystyczne

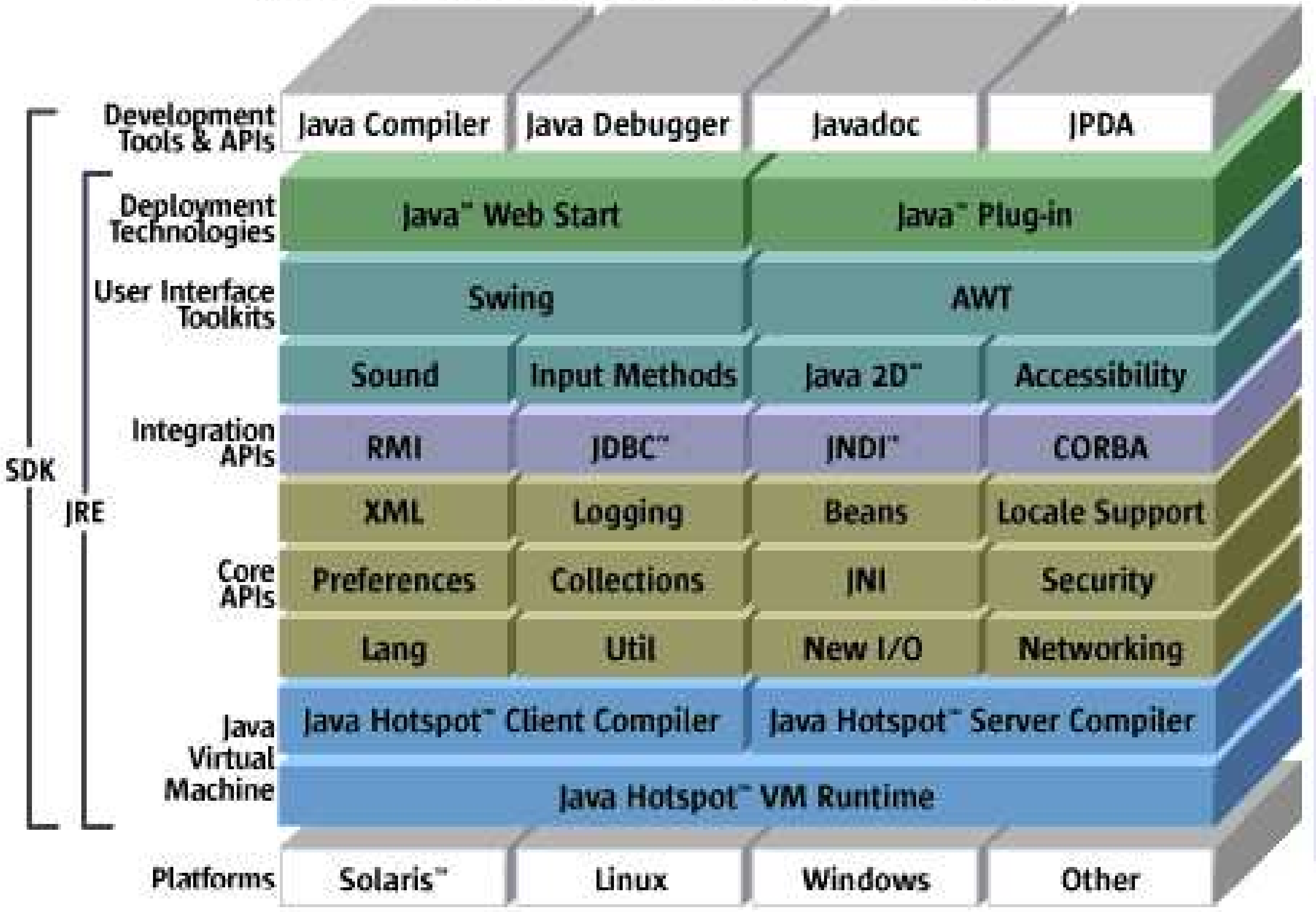
1. **"wszystko jest obiektem"** - nie ma zwykłych globalnych funkcji/procedur, są jedynie metody klas... wszystko (np operacje i/o) wykonuje się przy pomocy zestawu obiektów
2. po kompilacji uzyskujemy kod pośredni **B-kod** (Byte-kod), który można uruchomić przy pomocy maszyny wirtualnej Javy (=**JVM**) - jest to program "java"
3. B-kod jest "przenośny" (przenośność binarna a nie tylko źródłowa!)
   ten sam B-kod można uruchomić na różnych platformach (systemach operacyjnych)
   => można uruchomić B-kod ściągnięty z internetu (tzw mobilność kodu); applet-y
4. B-kod klasy jest pobierany dopiero w chwili tworzenia obiektu klasy
5. zrezygnowano z "niebezpiecznych" konstrukcji C/C++: wskaźników (zastąpiono je referencjami modyfikowalnymi)
6. OOP: zrezygnowano z dziedziczenia wielokrotnego, dodano interfejsy "samoistne"
7. automatyczne zwalnianie pamięci (GC = garbage collector); => klasy nie mają destruktorów
8. BARDZO obszerna biblioteka standardowa (pakiety standardowe)
9. Java dobrze się nadaje do tworzenia aplikacji rozproszonych

Java - tutorial i dokumentacja

Standardowy tutorial Javy:
http://java.sun.com/docs/books/tutorial/index.html

Standardowa dokumentacja (pakietów czyli "std biblioteki") Javy:
http://java.sun.com/j2se/1.4.2/docs/index.html

Java™ 2 Platform, Standard Edition v 1.4



Java - klasy i dziedziczenie

```
class CCC {
  public CCC() {
    System.out.println("CCC: konstruktor, this="+this);
      // Uwaga: w javie NIE ma destruktorow!
  }
  public int i, j;
  public void pisz() {
    System.out.println("CCC: i="+i+", j="+j+", this="+this);
  }
  public void wykonaj() {
    pisz();
  }
}
class CCC1 extends CCC {
    // mozna dziedziczyc tylko z 1 klasy!
  public void pisz() {
    System.out.println("CCC1: i="+i+", j="+j+", this="+this);
  }
}
```

```
public class test_CCC {
  public static void main(String argsp[]) {
    CCC a= new CCC();
      // "a" jest odnosnikiem do obiektu (=zmienna referencyjną)
        // wartościa "a" jest odniesienie do obiektu
    CCC b; b= a; // zmienne ref sa modyfikowalne!!!
    a.pisz();
    b.pisz();
      // CCC: i=0, j=0, this=CCC@8ed465 - widac że to ten sam obiekt!
      // CCC: i=0, j=0, this=CCC@8ed465

    CCC c= new CCC1(); // konw. do nadklasy
    c.wykonaj();
      // CCC1: i=0, j=0, this=CCC1@11a698a
        // jak widac pisz() jest metoda polimorficzna;
          // wszystkie metody nieprywatne sa polimoficzne !!!

    CCC1 d;
    //d=c; // blad kompilatora, bo to konw. do podklasy
    d=(CCC1)c; // jawna konwersja niezbedna
    d.wykonaj();
  }
}
```

Java - interfejsy "samoistne"

Klasa może implementować dowolnie wiele interfejsów (co częściowo rekompensuje brak wielokrotnego dziedziczenia)!

```
class CCC implements III {
  public CCC() {
    System.out.println("CCC: konstruktor, this="+this);
  }
  public int i, j;
  public void pisz() {
    System.out.println("CCC: i="+i+", j="+j+", this="+this);
  }
  public void wykonaj() {
    pisz();
  }
}
class CCC1 extends CCC {
  public void pisz() {
    System.out.println("CCC1: i="+i+", j="+j+", this="+this);
  }
}
interface III {
  public void pisz();
  public void wykonaj();
}
```

```
public class test_CCC {
  public static void main(String argsp[]) {
    III a, b;
    a= new CCC();
    a.wykonaj();
    b=a;
    b.wykonaj();

    III c; c=new CCC1(); // ok, mimo ze CCC1 nie impl. bezposrednio III
    c.wykonaj();
/*
    CCC: konstruktor, this=CCC@8ed465
    CCC: i=0, j=0, this=CCC@8ed465
    CCC: i=0, j=0, this=CCC@8ed465
    CCC: konstruktor, this=CCC1@107077e
    CCC1: i=0, j=0, this=CCC1@107077e
*/
```

Java - widoczność klas i składowych klas

Klasy:

1. klasy public - wszyscy mogą ich używać (muszą być zdef w osobnym pliku o nazwie identycznej jak nazwa klasy + ".java"/ ".class")

2. klasy pakietowe (bez słowa public) - mogą być używane tylko w pakiecie w ktorym zdef klase

Składowe klas:

1. składowe public, private, protected - znaczenie jak w C++

2. składowe pakietowe (bez słowa public, ...) - widoczne tylko w pakiecie w ktorym zdef klase

Java - pakiet java.lang: klasa String

```
//
// klasa String - stringi niemodyfikowalne
//

String s1= "aaa"; // nietypowe zachowanie jak na obiekty Javy!!!
String s2= s1+"";
System.out.println(s1.equals(s2)); // porównywanie stringów; zwraca true
System.out.println(s1==s2); // porównywanie ref; zwraca false !!!
// true
// false
  // (jak napisac String s2= "aaa"; to tego efektu nie ma!!!)

// stringi w zmiennych klasy String sa niemodyfikowalne ...
String s3= "A ku ku";
s3= "123"+s2+"qwe";
  // mogę przypisac do s3 inny obiekt klasy String
  // ale nie mogę zmodyfikować jakiegoś znaku s3
System.out.println(s3.charAt(1));
  // mogę jedynie odczytać znak s3
```

```
//
// klasa StringBuffer - stringi modyfikowalne
//

StringBuffer s10= new StringBuffer("qwe");

System.out.println("s10="+s10);
s10.append("123");
System.out.println("s10="+s10);
s10.setCharAt(3,'x'); // mogę modyfikować znaki stringu!
System.out.println("s10="+s10);
/*
s10=qwe
s10=qwe123
s10=qwex23
*/
```

Java - tablice

Tablice są w Javie obiektami;

- posiadają pole "length" zawierające liczbę elementów tablicy (indeksowane od 0);

- tablice wielowymiarowe to tablice, to tablice których elementami są tablice; można więc zbudować np "macierz trójkątną"

...

```
// tablica integerow
int tab[]; int i=20; tab=new int[i];
for(int j=0; j<tab.length; j++) tab[j]=j*2;
for(int j=0; j<tab.length; j++) {
  System.out.print(tab[j]+", ");
}
System.out.println();

// tablica stringow (obiektow!)
String tab2[]= new String[30];
for(int j=0; j<tab2.length; j++) tab2[j]="A ku ku !!! ("+j+")";
for(int j=0; j<tab2.length; j++) {
  System.out.println(tab2[j]+", ");
}
```

```
// tablice wielowymiarowe
float tab4[][]; // deklaracja odnośnika do obiektu tablicowego
tab4= new float[10][20]; // tworzenie tablicy (obiektu tablicowego)
for(int j=0; j<tab4.length; j++) {
  for(int k=0; k<tab4[j].length; k++) tab4[j][k]=j+k*(float)1000.0;
}
for(int j=0; j<tab4.length; j++) {
  for(int k=0; k<tab4[j].length; k++) System.out.print(tab4[j][k]+", ");
  System.out.println();
}

// tablice wielowymiarowe nieregularne
String tab3[][];
tab3= new String[10][];
  // tworzymy 10-elem tablicę, której elementami są tablice stringów
for(int j=0; j<10; j++) {
  tab3[j]= new String[j+1];
    // tworzymy "wiersz" złożony z (j+1)- elementów
  for(int k=0; k<=j; k++)
    tab3[j][k]= new String("A ku ku "+j+","+k);
}
for(int j=0; j<10; j++) {
  for(int k=0; k<=j; k++) System.out.print(tab3[j][k]+", ");
  System.out.println();
}
```

```
/* wydruk przykładu z tab3
A ku ku 0,0,
A ku ku 1,0, A ku ku 1,1,
A ku ku 2,0, A ku ku 2,1, A ku ku 2,2,
A ku ku 3,0, A ku ku 3,1, A ku ku 3,2, A ku ku 3,3,
A ku ku 4,0, A ku ku 4,1, A ku ku 4,2, A ku ku 4,3, A ku ku 4,4,
A ku ku 5,0, A ku ku 5,1, A ku ku 5,2, A ku ku 5,3, A ku ku 5,4, A ku ku 5,5,
A ku ku 6,0, A ku ku 6,1, A ku ku 6,2, A ku ku 6,3, A ku ku 6,4, A ku ku 6,5, A ku ku 6,6,
A ku ku 7,0, A ku ku 7,1, A ku ku 7,2, A ku ku 7,3, A ku ku 7,4, A ku ku 7,5, A ku ku 7,6, A ku ku 7,7,
A ku ku 8,0, A ku ku 8,1, A ku ku 8,2, A ku ku 8,3, A ku ku 8,4, A ku ku 8,5, A ku ku 8,6, A ku ku 8,7, A ku ku 8,8,
A ku ku 9,0, A ku ku 9,1, A ku ku 9,2, A ku ku 9,3, A ku ku 9,4, A ku ku 9,5, A ku ku 9,6, A ku ku 9,7, A ku ku 9,8, A ku ku 9,9,
*/
```

Java - przekazywanie danych do/z metod przez parametry

Generalnie w Javie przekazuje się parametry *przez wartość*; oganiczenie to można obejść przy pomocy pomocniczych klas "holderów" ...

```
class intHolder {
  public intHolder(int _value) { value=_value; }
  public int value;
}
class mojaKlasa {
  static public void mojaMetoda(int a, intHolder b, intHolder c) {
    b.value=a+1;
    c.value=a+2;
  }
}
// użycie klas:
intHolder ih1= new intHolder(0);
intHolder ih2= new intHolder(0);
mojaKlasa.mojaMetoda(123, ih1, ih2);
System.out.println("ih1.value="+ih1.value);
System.out.println("ih2.value="+ih2.value);
// wydruk:
ih1.value=124
ih2.value=125
```

Java - pakiety i CLASSPATH

Pakiety to rodzaj modułów zawierających klasy i interfejsy ...

1. Klasy pakietu o nazwie "qqq" powinny się znajdować w katalogu "qqq"
2. jesli pakiet używany przez nasz program nie leży w podkatalogu katalogu głównego naszego programu to także należy dodać odpowiedni katalog do zmiennej CLASSPATH
3. można umieścić zawartość katalogu "qqq" w pliku zip lub jar, należy wtedy dodać do zmiennej CLASSPATH ścieżkę do tego pliku!
   export CLASSPATH=dollar CLASSPATH/sciezka do pliku zip

Przykład pakietu o nazwie "qqq" :

```
// to jest plik qqq/CCC1.java (qqq jest katalogiem!!!)

package qqq; // deklaracja ze to paktiet o nazwie qqq
  // musi sie nazywać tak samo jak katalog w którym ten plik się znajduje
  // kompilacja plików w katalogu: javac qqq/*.java

public class CCC1 {
    // klasa publiczna dostępna dla użytkowników pakietu qqq
  public int i, j;
  private int x,y;
```

```
  public CCC1a c1a;
  public CCC1b c1b;
  public CCC1() {
    c1a= new CCC1a();
    c1b= new CCC1b();
    i=111; j=222; x=333; y=444;
  }
  public void pisz() {
    System.out.println("qqq.CCC1: i="+i+", j="+j+", this="+this);
  }
  public void wykonaj() {
    c1a.pisz();
    c1b.pisz();
  }
};

class CCC1a {
    // to jest klasa pakietowa - dostepna TYLKO w tym pakiecie!!!
  public int i, j;
  public void pisz() {
    System.out.println("qqq.CCC1a: i="+i+", j="+j+", this="+this);
  }
};

/*
```

```
public class CCC1b {
    // klasy publiczne musza miec swoj wlasny plik, ktory sie tak nazywa jak klasa!!!
    // czyli ta klasa musi byc umieszczona w osobnym pliku CCC1b.java
    // (w katalogu qqq)
  public int i, j;
  public void pisz() {
    System.out.println("qqq.CCC1b: i="+i+", j="+j+", this="+this);
  }
};
*/
```

```
// sposób użycia pakietu qqq:

qqq.CCC1 o1= new qqq.CCC1();
o1.pisz();

//qqq.CCC1a o2= new qqq.CCC1a();
  // blad !!! bo to klasa pakietowa

qqq.CCC1b o3= new qqq.CCC1b();
o3.pisz();

o1.wykonaj();
  // o1 jest wlascicielem pewnego obiektu klasu qqq.CCC1a;
  // o1.wykonaj() wywołuje metodę pisz tego obiektu i wszystko dziala

  // widac ze moge sie odwolywac posrednio do obiektu klasy qqq.CCC1a
  // (za posrednictwem obiektu klasy qqq.CCC1)
  // choc nie moge go bezposrednio utworzyc !!!
```

Java - jak sobie radzić bez wskaźników

Jak zaprogramować klasę Lista bez wskaźników ???

```
class Element {
  public void pisz() {
    System.out.println("ta metoda powinna byc podmieniona!!! this="+this);
  }
  public Element nast; // czy nie da sie tego zrobic sprytniej?
}
class ElementLiczba extends Element {
  private int liczba;
  public ElementLiczba(int _liczba) {
    liczba=_liczba;
  }
  public void pisz() {
    System.out.println(liczba);
  }
}
```

```
class Lista {
  private Element glowa;
  public Lista() {
    glowa=null;
  }
  public void dodajElem(Element e) {
    e.nast=glowa; glowa=e;
  }
  public void usunElem(Element e) {
    if( e==glowa ) {
      glowa=e.nast;
    } else { // e!=null
      Element e1= glowa; while(e1.nast!=e) e=e.nast;
      e1.nast=e.nast;
    }
  }
  public void piszWszystkieElem() {
    Element e= glowa;
    while( e!=null ) { e.pisz(); e=e.nast; }
  }
}
```

```
// testujemy klase Lista ...
ElementLiczba l1= new ElementLiczba(111);
ElementLiczba l2= new ElementLiczba(222);
ElementLiczba l3= new ElementLiczba(333);

Lista lista= new Lista();

lista.dodajElem(l1);
lista.dodajElem(l2);
lista.dodajElem(l3);
System.out.println("lista 1");
lista.piszWszystkieElem();

lista.usunElem(l2);
System.out.println("lista 2");
lista.piszWszystkieElem();
/* wydruk:
  lista 1
  333
  222
  111
  lista 2
  333
  111
*/
```

Java - pakiet java.util: klasa Vector

```
java.util.Vector v1= new java.util.Vector();
v1.add("A ku ku !!!");
  // dodajemy obiekt klasy String !!!
for(int i=0; i<10; i++) v1.add("A ku ku "+i);

System.out.println("v1="+v1);

v1.set(2, "qqq www eee rrr");

System.out.println("v1="+v1);

for (java.util.Enumeration e = v1.elements() ; e.hasMoreElements() ; ) {
    // java.util.Enumeration to interfejs
    // e jest kolekcja elementow wektora v1 po ktorych przebiegamy ...
  System.out.println(e.nextElement());
}
```

```
/*
v1=[A ku ku !!!, A ku ku 0, A ku ku 1, A ku ku 2, A ku ku 3, A ku ku 4, A ku ku 5, A ku ku 6, A ku ku 7, A ku ku 8, A ku ku 9]
v1=[A ku ku !!!, A ku ku 0, qqq www eee rrr, A ku ku 2, A ku ku 3, A ku ku 4, A ku ku 5, A ku ku 6, A ku ku 7, A ku ku 8, A ku ku 9]
A ku ku !!!
A ku ku 0
qqq www eee rrr
A ku ku 2
A ku ku 3
A ku ku 4
A ku ku 5
A ku ku 6
A ku ku 7
A ku ku 8
A ku ku 9
*/
```

Java I/O - pakiet java.io

Klasa **java.io.File**

1. reprezentuje *scieżkę* do pliku lub katalogu
2. jeśli jest to katalog to pozwala operować na jego zawartości
3. pozwala odczytać informacje o pliku/katalogu np kiedy został ostatnio zmodyfikowany

```
java.io.File f= new java.io.File(".");
  // f reprezentuje biezacy katalog ...

java.io.File [] x= f.listFiles();
System.out.println("x.length="+x.length);
for(int i=0; i<4; i++)
  System.out.println(x[i].getName()+", "+x[i].isFile()+", "+x[i].isDirectory());

// wydruk:
x.length=64
jacl133, false, true
jacl01.tcl, true, false
jacl_editor.tcl, true, false
jaclsh, true, false
```

Java I/O - pakiet java.io

Wejście i wyjście:

1. dwie abstrakcyjne klasy **java.io.InputStream** oraz **java.io.Reader** posiadające metodę "int read()" odczytującą jeden znak; podklasy tych klas służą do odczytu z plików i innych źródeł danych; można je nazwać "strumieniami wejścia"

2. dwie abstrakcyjne klasy **java.io.OutputStream** oraz **java.io.Reader** posiadające metodę "void write(int)" zapisującą jeden znak; podklasy tych klas służą do zapisu do plików i innych odbiorców danych; można je nazwać "strumieniami wyjścia"

3. dlaczego 2 klasy wykonują zbliżoną funkcję (np InputStream i Reader)?
   odp: InputStream jest zorientowaby bajtowo, Reader obsługuje znaki **Unicode**

4. aby wykonać operację na plikach trzeba utworzyć i połączyć kilka obiektów (które są podklasami Input/OutputStream)
   jest to wzorzec projektowy "Dekorator", lista obiektów z tym samym interfejsem, które przekazują komunikat do następnego obiektu

Java I/O - pakiet java.io

Podklasy java.io.InputStream = typy strumieni wejściowych

1. java.io.ByteArrayInputStream - dane są przechowywane w tablicy bajtów
2. java.io.StringBufferInputStream - dane są przechowywane w obiekcie StringBuffer
3. **java.io.FileInputStream** - czytanie plików
4. java.io.PipedInputStream - koncówka wyjściowa łącza (tego w potokach)
5. java.io.SequenceInputStream - łączenie kilku strumieni
6. java.io.FilterInputStream - klasa abstrakcyjna; to jest interfejs dekoratora! (zmieniamy zachowanie strumienia przy pomocy podklasy FilterInputStream)

```
// przykład użycia FileInputStream do odczytania pliku
java.io.FileInputStream fis1= new java.io.FileInputStream("Rysunek2.java");
while( fis1.available()!=0 ) {
  int c = fis1.read();
  System.out.print(c+", "); // wyswietla bajty! (0..255)
              // czyli czytamy z pliku po jednym bajcie
  //System.out.print((char)c); // wyswietla znaki
}
fis1.close();
  // Uwaga: FileInputStream pozwala JEDYNIE na odczyt sekwencyjny!!!
```

Java I/O - pakiet java.io

Podklasy java.io.FilterInputStream = zmienianie zachowania strumieni
to jest zgodne z wzorcem projektowym Dekorator - poniższe klasy to "dekoratorzy"

1. **java.io.DataInputStream** - odczyt danych (np int, char, float) zapisanych binarnie
2. **java.io.BufferedInputStream** - - dodaje do strumienia bufor przyspieszający operacje; powinniśmy używać

```
// przykład użycia buforowanego strumienia wejściowego do czytania z pliku
java.io.FileInputStream fis1= new java.io.FileInputStream("Rysunek2.java");
java.io.BufferedInputStream bis1= new java.io.BufferedInputStream(fis1);
while( bis1.available()!=0 ) {
    // metoda available() zwraca liczbe bajtow która zostały do przeczytania
  int c = bis1.read();
  System.out.print(c+", "); // wyswietla bajty! (0..255)
             // dlatego mówimy że InputStream jest zorientowany na bajty
  //System.out.print((char)c); // wyswietla znaki
}
bis1.close();
```

```
// DataInput/OutputStream ...

// (binarny) zapis floatow w pliku
java.io.FileOutputStream fos1= new java.io.FileOutputStream("mojedane.dat");
java.io.BufferedOutputStream bos1= new java.io.BufferedOutputStream(fos1);
java.io.DataOutputStream dos1= new java.io.DataOutputStream(bos1);
for( float f=0.0F; f<100.0F; f+=0.5F ) dos1.writeFloat(f);
dos1.close();

// odczytywanie kolejnych float-ow zapisanych binarnie z pliku
java.io.FileInputStream fis1= new java.io.FileInputStream("mojedane.dat");
java.io.BufferedInputStream bis1= new java.io.BufferedInputStream(fis1);
java.io.DataInputStream dis1= new java.io.DataInputStream(bis1);
while( dis1.available()!=0 ) {
    // available zwraca liczbe bajtow (a nie elementów!) które pozostały do odczytania
  float f = dis1.readFloat();
  System.out.print(f+", ");
}
dis1.close();
/* wydruk:
0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, ...
*/
```

Java I/O - pakiet java.io

Zapis danych w trybie tekstowym (czyli np int jako ciąg znaków/cyfr) ...
służy do tego klasa **java.io.PrintStream** oraz **java.io.PrintWriter**
Uwaga: **java.io.PrintStream** to kolejny dekorator dla OutputStream!

```
java.io.FileOutputStream fos1= new java.io.FileOutputStream("qqq.txt");
java.io.BufferedOutputStream bos1= new java.io.BufferedOutputStream(fos1);
java.io.PrintStream ps1= new java.io.PrintStream(bos1);
ps1.println(123); ps1.println("A ku ku !!!");
ps1.close();
```

??? co z tekstowym czytaniem ???
do przeczytania linii tekstu (zakończonej znakami $\backslash n$, $\backslash r$, $\backslash rn$) trzeba użyć klasy java.io.BufferedReader

```
java.io.FileInputStream fis1= new java.io.FileInputStream("qqq.txt");
java.io.InputStreamReader isr1= new java.io.InputStreamReader(fis1);
java.io.BufferedReader br1= new java.io.BufferedReader(isr1);
String s_1= br1.readLine(); String s_2= br1.readLine();
System.out.println(s_1);
System.out.println(Integer.parseInt(s_1));
  // mozemy przeksztalcic string z intem na inta!
System.out.println(s_2);
br1.close();
```

Java I/O - pakiet java.io

Klasy **java.io.Reader** i **java.io.Writer** ...

1. klasy te są zorientowane na znaki Unicode (a nie na bajty jak InputOutputStream)
2. typ char Javy zawiera znak Unicode zapisany na 2 bajtach (dotyczy to także znaków w string-ach!); jest to standardowy sposób kodowania znaków znany jako USC-2 lub UTF-16
3. w kodzie programu Javy znaki Unicode zapisujemy tak: "$\backslash uXXXX$", gdzie XXXX to 4 cyfry szesnastkowe
4. przy zapisie do pliku znaki Unicode mogą być kodowane na różne sposoby, np metodą UTF-8 (wtedy znaki ASCII zajmuja 1 bajt, inne znaki zajmują więcej bajtów); java.io.Writer pozwala swobodnie określić sposób kodowania znaków!
5. dodatkowo, metody DataOutputStream.writeUTF() i DataOutputStream.readUTF() pozwalają zapisywać stringi w kodowaniu UTF-8

Jak zmusic Writer aby zapisywal znaki Unicode w konkretnym kodowaniu???

```
java.nio.charset.Charset ch1= java.nio.charset.Charset.forName("UTF-16BE");
  // tworzy obiekt Charset reprezentujący sposób kodowania UTF-16BE
  // (UTF-16BE różni się od UTF-16LE kolejnością 2 bajtów na znak)
java.io.FileOutputStream s1= new java.io.FileOutputStream("qqq_16BE.txt");
java.io.OutputStreamWriter w1= new java.io.OutputStreamWriter(s1, ch1);
  // jeszcze by się przydał BufferedWriter !
//w1.write('\u0635'); w1.write('\u0635'); w1.write('\u0635');
w1.write('A'); w1.write('B'); w1.write('C');
w1.close();
  // jaka jest długość pliku qqq_16BE.txt? odp: 6 bajtów!!!
```

Java I/O - serializacja obiektów

Serializacja to zapis obiektu Javy w postaci ciągu bajtów, z których można potem odtworzyć oryginalny obiekt. *Nie mylić z metodą toString() !!!*

Klasa: java.io.ObjectOutputStream i java.io.ObjectInputStream
metody: "void writeObject(Object obj)" i "void readObject(Object obj)"

Interfejs: *java.io.Serializable* - nie zawiera żadnych metod!!!
klasy które chcemy serializować muszą implementować ten interfejs

Standardowa serializacja polega na zapisie wszystkich pól obiektu a także WSZYSTKICH (muszą być serializowalne!!!) obiektów agregowanych przez nasz obiekt; zostanie zapisana cała sieć/pajęczyna obiektów powiązanych z naszym obiektem!

Interfejs *java.io.Externalizable* z metodami readExternal(ObjectInput) oraz writeExternal(ObjectOutput) pozwala serializowac obiekty niestandardowo (np nie wszystkie pola zapisywac)

```java
// definicja klasy/obiektu który bedziemy serializowac ...
class CCC
  implements java.io.Serializable
{
  public int i, j;
  public CCC1 refC1; // agreguje obiekty klas CCC1 i CCC2
  public CCC2 refC2;
  public void pisz() {
    System.out.println("CCC: i="+i+", j="+j+", this="+this);
  }
  public CCC(int i1, int j1) {
    i=i1; j=j1;
    refC1= new CCC1(111,222);
    refC2= new CCC2(333,444);
    System.out.println("CCC: konstruktor, this="+this);
  }
}
```

```
class CCC1
  implements java.io.Serializable
{
  public int i, j;
  public DDD refD; // agreguje obiekt klasy DDD
  public void pisz() {
    System.out.println("CCC1: i="+i+", j="+j+", this="+this);
  }
  public CCC1(int i1, int j1) {
    i=i1; j=j1;
    refD= new DDD();
    System.out.println("CCC1: konstruktor, this="+this);
  }
}
class DDD
  implements java.io.Serializable
{
  public int i, j;
  public void pisz() {
    System.out.println("DDD: i="+i+", j="+j+", this="+this);
  }
  public DDD() {
    i=555; j=666;
    System.out.println("DDD: konstruktor, this="+this);
  }
}
```

```java
class CCC2
  implements java.io.Serializable
{
  public int i, j;
  public void pisz() {
    System.out.println("CCC2: i="+i+", j="+j+", this="+this);
  }
  public CCC2(int i1, int j1) {
    i=i1; j=j1;
    System.out.println("CCC2: konstruktor, this="+this);
  }
}
```

```
// tworzymy obiekt ...
CCC c= new CCC(123,321);
c.pisz();
c.refC1.pisz();
c.refC2.pisz();
c.refC1.refD.pisz();

// serializujemy obiekt ...
java.io.FileOutputStream s1= new java.io.FileOutputStream("ser1.dat");
java.io.ObjectOutputStream d1= new java.io.ObjectOutputStream(s1);
d1.writeObject(c);
  // co sie stanie jesli DDD nie jest Serializable ???
  // odp: wyjatek!!!
d1.close();

/* // wydruk:
DDD: konstruktor, this=DDD@bd0108
CCC1: konstruktor, this=CCC1@8ed465
CCC2: konstruktor, this=CCC2@107077e
CCC: konstruktor, this=CCC@7ced01
CCC: i=123, j=321, this=CCC@7ced01
CCC1: i=111, j=222, this=CCC1@8ed465
CCC2: i=333, j=444, this=CCC2@107077e
DDD: i=555, j=666, this=DDD@bd0108
*/
```

```
// deserializujemy obiekt ...
java.io.FileInputStream s1= new java.io.FileInputStream("ser1.dat");
java.io.ObjectInputStream d1= new java.io.ObjectInputStream(s1);
CCC c= (CCC) d1.readObject();
  // - jawna konwersja jest niezbedna!!
  // - żadne konstruktory się nie odzywają !!
  // - co sie stanie jesli ktoras z klas (CCC, CCC1, CCC2, DDD) jest niedostepna ???
  //   odp: wyjatek; nawet jesli: Object c= d1.readObject();
d1.close();
c.pisz();
c.refC1.pisz();
c.refC2.pisz();
c.refC1.refD.pisz();

/* // wydruk:
CCC: i=123, j=321, this=CCC@b162d5
CCC1: i=111, j=222, this=CCC1@1cfb549
CCC2: i=333, j=444, this=CCC2@186d4c1
DDD: i=555, j=666, this=DDD@f9f9d8
*/
```

Java I/O - interfejs do baz danych: JDBC

JDBC to Javowy interfejs do relacyjnych (SQL-owych) baz danych różnego typu ...

1. wszystkie bazy obsługuje się w jednolity sposób, przy użyciu klas/interfejsów:
   java.sql.Connection - nawiązywanie połączenia z bazą danych
   java.sql.Statement - zawiera kwerendę SQL
   java.sql.ResultSet - zawiera wynik działania kwerendy SQL (czyli tabele)

2. JDBC jest rozszerzalne: można dodać nowe moduły pozwalające obsługiwać nowy typ bazy danych (tzw drivery JDBC)

```
import java.sql.*;
import org.sqlite.JDBC;

public class sqlite01 {
  public static void main(String[] args) {
    try {
      String fileName = "./test.db";

      // ladujemy sterownik JDBC dla bazy SQLite
      // (plik jar ze sterownikim trzeba umiescic na CLASSPATH!)
      Class.forName("org.sqlite.JDBC");

      // nawiązujemy połączenie
```

```
      Connection conn = DriverManager.getConnection("jdbc:sqlite:"+fileName);

      // tworzymy obiekt reprezentujący kwerendę SQL
      Statement stmt = conn.createStatement();

      // wykonujemy kwerendę zwracająca ResultSet
      ResultSet rs = stmt.executeQuery("SELECT * FROM t1");

      while (rs.next()) {
        String a   = rs.getString("a");
        String b = rs.getString("b");
        System.out.println("a="+a+", b="+b);
      }

      conn.close();

    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Java GUI czyli interfejs graficzny/okienkowy

Podstawowe pojęcia:

1. komponent wizualny (widget, kontrolka, formant) - klasa/obiekt reprezentujący okienko lub inny obiekt graficzny na ekranie;
   jest widoczny i potrafi reagować na czynności użytkownika (klawiatura i mysz);
   GUI = graficzny interfejs użytkownika, jest zbudowany z komponentów wizualnych
2. **AWT** i **SWING**: java.awt i javax.swing to pakiety zawierające klasy do budowy GUI; SWING jest zbudowany "nad" AWT
3. podklasy **java.awt.Component** to np Button, Label, Container, ...;
   Button, Label - typowe komponenty wizualne;
   Container - komponent złożony, zawierający inne komponenty wizualne
4. podklasy **java.awt.Container** to Panel, ScrollPane, JComponent, Window/Frame;
5. podklasy **javax.swing.JComponent** to JLabel, JButton, JPanel, JTable, ... to już SWING!!!
6. główne okno aplikacji: może to byc klasa **java.awt.Frame** lub **javax.swing.JFrame**
7. układanie komponentów w kontenerze: Layout Manager
8. obsługa zdarzeń: interfejs *Listener

Java GUI - layout manager - zarządca układu komponentów

1. każdy kontener awt (np java.awt.Panel) może mieć własny layout manager
2. elementem kontenera awt może być inny kontener (np można zagnieżdżać panele)
3. zazwyczaj dla każdego formularza/ okna dialogowego tworzy się klasę Javy (np podklase java.awt.Frame) tworzącą wszystkie komponenty formularza (robi to konstruktor klasy lub specjalnie do tego przeznaczona metoda)

patrz przykaład w jacl ...

Java GUI - obsługa zdarzeń: interfejs *Listener

Interfejsy: **java.awt.event.*Listener**, np java.awt.event.ActionListener
interfejs Listener posiada metody wywoływane dla każdego zdarzenia:

```
// dotyczy zdarzeń typu "naciśnięcie guzika" dla komponentu java.awt.Button
public interface ActionListener extends EventListener {
  void actionPerformed(ActionEvent e)
}
```

Komponenty wizualne mają metodę **add*Listener**, np addActionListener; przy ich pomocy można zerejestrować w komponencie obiekty implementujące interfejs Listener, które będą odbierały zawiadomienia o zdarzeniach (=wywołania metod interfejsu)
jest to zgodne ze wzorcem projektowym "Observer"

Klasy pomocniczne **java.awt.event.*Adapter** np WindowAdapter zawieraja typową definicję metod interfejsu *Listener (przydatne gdy chcemy bezpośrednio zdefiniować niewiele metod spośród wielu)

```
// klasa reprezentująca formularz ...
class MojFormularz
  extends java.awt.Frame
  implements java.awt.event.ActionListener // ta klasa równocześnie obsługuje zdarzenia
                                    //emitowane przez komponenty na formularzu
{
  public MojFormularz() {
    utworzKomponenty();
    setSize(200,200);
    show();
  }
  public void utworzKomponenty() {
    setLayout(new java.awt.FlowLayout());
    b1= new java.awt.Button("guzik nr 1");
    b1.addActionListener(this); // tu definiujemy obsługę zdarzenia
    b2= new java.awt.Button("guzik nr 2");
    add(b1);
    add(b2);
  }
  public void actionPerformed(java.awt.event.ActionEvent e) {
    System.out.println("actionPerformed; "+e.toString());
  }
  public java.awt.Button b1;
  public java.awt.Button b2;
}
```

Współbieżność w Javie

Tworzenie wątków

1. do tworzenia wątków służą: klasa **java.lang.Thread** oraz interfejs **java.lang.Runnable** z metoda "void run()"; wątków dotyczą także metody wait(), notify() i notifyAll() klasy Object.

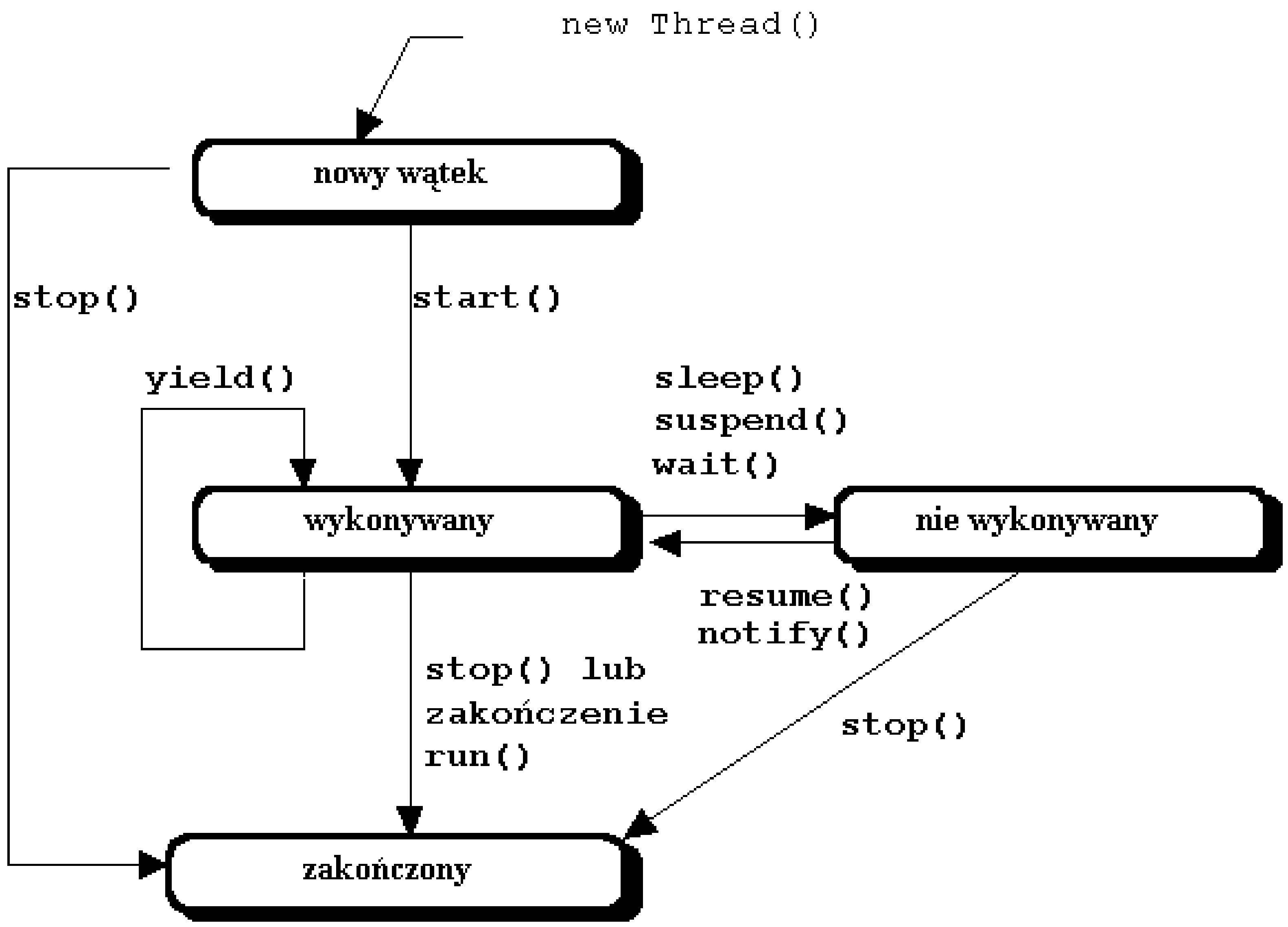
2. Każdy wątek w języku Java wymaga istnienia obiektu klasy Thread, sterującego tym wątkiem.

3. Konstruktor klasy Thread wymaga odniesienia do obiektu klasy implementującej interfejs Runnable, zawierającej metodę run(), której kod będzie wykonywany przez wątek.

4. Aby uruchomić wątek, należy wywołać metodę start() obiektu klasy Thread;
Uwaga: wątek można zniszczyć metodą stop(); do zawieszania i odwieszania wątków (terminologia J.B.), służą metody suspend() i resume(); do uśpienia wątku na określony czas służy metoda sleep(); *jednak metody uważa się za przestarzałe i niebezpieczne!!! lepiej ten sam cel osiągnąć przy pomocy zmiennych dostępnych dla wszystkich wątkow ...*

```
public class Watek01 {
  static long IloscObrotow = 100;
  public static void main(String args[])
  {
    new Thread(new MojWatek01("AAA")).start();
    new Thread(new MojWatek01("BBB")).start();
    new Thread(new MojWatek01("CCC")).start();
  }
}
class MojWatek01 implements Runnable
{
  private String nazwa;
  MojWatek01(String _nazwa)
  {
    nazwa=_nazwa;
  }
  public void run()
  {
    int i;
    for(i=0; i<Watek01.IloscObrotow; i++) {
      System.out.println(nazwa+":"+i);
    }
  }
}
```

Stany wątku:



new Thread()
nowy wątek
stop()
start()
yield()
sleep()
suspend()
wait()
wykonywany
nie wykonywany
resume()
notify()
stop() lub
zakończenie
run()
stop()
zakończony

Współbieżność w Javie

Synchronizacja wątków

1. "Synchronizacja wątków" to wstrzymywanie działania wątków gdy to jest potrzebne; synchronizacja jest niezbędna gdy kilka wątków operuje równocześnie na tej samej strukturze danych.

2. W Javie do synchronizowania wątków mamy 2 narzędzia:
   - obiekty z metodami synchronizowanymi.
   - specjalna konstrukcja: *synchronized (obiekt) { kod }*

```
// przykład klasy z metodą synchronizowaną:
class Bank
{
   private long konto1=123456, konto2=0;
     // dzięki synchronized przelew jest operacją atomową
     // kilka wątków może równocześnie (bezpiecznie) dokonywać przelewów
   synchronized public void Przelew(long kwota)
   {
      konto1-=kwota; konto2+=kwota;
   }
}
```

Współbieżność w Javie

Zasada działania metod synchronizowanych i konstrukcji synchronized:

1. Jeśli wątek A wywoła metodę synchronizowaną obiektu X to mówimy, że wątek A zajął monitor obiektu X (terminologia J.B.). Gdy metoda ta się zakończy to mówimy że A zwolnił monitor obiektu X.
2. W danej chwili tylko 1 wątek może zajmować monitor danego obiektu. Tak więc metody synchronizowane zapewniają wzajemne wykluczanie.
3. Jeśli wątek A zajmuje monitor obiektu X i pewien wątek B spróbuje wywołać metodę synchronizowaną obiektu X, to wtedy wątek B zostanie zablokowany. Wątek B zostanie odblokowany dopiero gdy A zwolni monitor obiektu X, czyli gdy metoda synchronizowana którą A uruchomił się zakończy.
4. Inny sposób aby zająć monitor obiektu X to wykonać taki kod:
   *synchronized (X) { /*tu zajmuję monitor obiektu X*/ }; /*tu już nie zajmuje*/*
5. Każdy obiekt w języku Java posiada metody **wait()**, **notify()**, **notifyAll()** odziedziczone po klasie Object.
6. Wywołanie metody wait() na rzecz obiektu X powoduje zwolnienie monitora obiektu X, oraz wstrzymanie wykonania wątku wywołującego wait().
7. Wywołanie metody notify() na rzecz obiektu X powoduje wznowienie jednego spośród wstrzymanych na tym obiekcie wątków, ale dopiero wtedy, gdy wątek wywłujący notify() zwolni monitor obiektu.

8. Metody X.wait(), X.notify() i X.notifyAll() mogą być wykonywane jedynie przez wątki, które zajmują monitor obiektu X.

Uwagi:

1. Obiekt Javy z metodami synchronizowanymi to w rzeczywistosci monitor Hoare-go z pojedynczą zmienna condition!!!

2. Metoda notify() wznawia jeden wątek wstrzymany na wait() jednak NIE oznacza to, że wątek ten zaraz zacznie się wykonywać i wejdzie ponownie do monitora - mogą go w tym uprzedzić inne wątki - nasz obudzony wątek współzawodniczy ze wszystkimi innymi wątkami próbującymi zająć monitor.

3. NotifyAll() tym się różni od notify() że budzi wszystkie wątki wstrzymane na wait() - jednak monitor może zająć tylko jeden wątek!

```
/* przyklad synchronizacji wątkow w problemie producenta i konsumenta ...
     mamy tu:
       - 1 producenta i kilku konsumentow
       - nieskonczony bufor
*/
class Bufor
  // klasa reprezentująca bufor miedzy producentem i klientem
{
  private int buf[];
  private int buf_pisz, buf_czyt;
  static final int RozmiarBufora=500;
  Bufor()
  {
    buf=new int[RozmiarBufora];
    buf_pisz=0;
    buf_czyt=0;
  }
  synchronized public void WstawElement(int el)
  {
    buf[buf_pisz++]=el;
    notify();
  }
```

```
  synchronized public int PobierzElement()
  {
    while /*if*/ (buf_czyt==buf_pisz) // dlaczego MUSI byc while a nie if ???
      try { wait(); } catch(Exception e) {
        System.out.println("Blad !!! : "+e.getMessage());
        System.exit(1);
      }
    return buf[buf_czyt++];
  }
}

public class Watek04a // główna klasa programu
{
  public static void main(String args[])
  {
    Bufor bufor=new Bufor();

    new Thread(new Producent04(bufor)).start();

    new Thread(new Konsument04(bufor,1)).start();
    new Thread(new Konsument04(bufor,2)).start();
  }
}
```

```
class Producent04 implements Runnable
{
  private Bufor bufor;
  Producent04(Bufor _bufor)
  {
    bufor=_bufor;
  }
  public void run()
    {
    for(int i=0; i<Bufor.RozmiarBufora; i++)
    {
      bufor.WstawElement(i);

      try {
        Thread.currentThread().sleep(100);
      } catch(InterruptedException e) {}
    }
  }
}
```

```java
class Konsument04 implements Runnable
{
  private Bufor bufor;
  private int nr;
  Konsument04(Bufor _bufor, int _nr)
  {
    bufor=_bufor; nr=_nr;
  }
  public void run()
  {
    while(true)
    {
      int j=bufor.PobierzElement();
      System.out.println(j+"("+nr+")");
      try {
        Thread.currentThread().sleep(10);
      } catch(InterruptedException e) {}
    }
  }
}
```

```
// ten sam problem ale ze skonczonym buforem ...
//
abstract class Bufor
{
  private int buf[];
  private int buf_pisz, buf_czyt;
  protected int Wolne, Pelne;
  static final int RozmiarBufora=20;
  Bufor()
  {
    buf=new int[RozmiarBufora];
    buf_pisz=0;
    buf_czyt=0;
    Wolne=RozmiarBufora;
    Pelne=0;
  }
  public void WstawE(int el)
  {
    buf[buf_pisz++]=el;
    if(buf_pisz==RozmiarBufora) buf_pisz=0;
  }
  public int PobierzE()
  {
    int el=buf[buf_czyt++];
    if(buf_czyt==RozmiarBufora) buf_czyt=0;
```

```
    return el;
  }
  abstract synchronized public void WstawElement(int el);
  abstract synchronized public int PobierzElement();
}

class Bufor1 extends Bufor
{
  synchronized public void WstawElement(int el)
  {
    while(Wolne==0) try { wait(); } catch(Exception e) {};
    WstawE(el); Wolne--; Pelne++;
    notifyAll();
  }
  synchronized public int PobierzElement()
  {
    while(Pelne==0) try { wait(); } catch(Exception e) {};
    int el=PobierzE(); Wolne++; Pelne--;
    notifyAll();
    return el;
  }
}
```

Rozproszone obiekty Javy - obiekty RMI

materiały: SOP322/ Temat F1 i G

Rysowanie w oknie AWT

1. Można rysować na każdym komponencie wizualnym AWT o ile podomienimy metode **void paint(java.awt.Graphics g)** klasy java.awt.Component; parametr g to "kontekst graficzyny" pozwalający rysowac na okienktu (np można rysować linie i inne obiekty geometryczne)

2. Metoda paint() jest wywolywana automatycznie gdy okno wymaga odmalowania, lub można wymusić odmalowanie okna przez **repaint()** lub repaint(?,?,?,?)

3. Duże i złożone rysunki: można je wcześniej narysować na "obrazku" (java.awt.Image), a metodzie paint() zlecić jedynie rysowanie obrazka, czyli: g.drawImage(img, x1, y1, this);

```
public class Rysunek extends Frame {
  public void paint(Graphics g) {
    Insets ins= getInsets();
    int x1=ins.left,y1=ins.top,x2=getWidth()-ins.right,y2=getHeight()-ins.bottom;
    g.drawLine(x1, y1, x2, y2);
        // rysuje przekątną (w obszarze na ktorym powinno się rysować)
    super.paint(g);
  }
}
```

Rysowanie w oknie AWT c.d.

1. Komponenty wizualne "ciężkie" i "lekkie":
   (a) ciężkie to te związane z oknami (pod)systemu okienkowego
   (b) lekkie są malowane w 100% przez awt/swing; lekkie to głównie te dziedziczące z JComponent
2. Przyczyny malowania okna: "systemowe" i "na żądanie" aplikacji (repaint())
3. Malowanie systemowe/ na żadanie, dla komponentów ciężkich/lekkich działa trochę inaczej w każdym z 4 przypadków ...
4. Usprawnienia w rysowaniu złożonych obrazków:
   jeśli komp. cięzki jest malowany na żądanie to metoda paint() nie jest wywoływana bezpośrednio lecz za pośrednictwem metody **update()**; update() standardowo czyści okno i wywołuje paint(), które odmalowuje CAŁE okno; można podmienić metodę update(): na podstawie ClipBounds kontekstu graficznego można się zorientować co powinno zostać zmienione w obrazku i przerysować tylko to!
   - patrz: przykład Jacl
5. Okno z podmienioną metodą paint() MOŻE zawierać komponenty wizualne (sa one odmalowywane PO odmalowaniu okna).

Introspekcja w Javie

???

JavaBeans

1. Komponent JavaBeans to klasa Javy spełniająća pewne dodatkowe wymagania;
   dotyczą one głównie nazewnictwa metod ...

2. Podstawowe składniki komponentu JB:
   (a) **metody**
       takie same jak w zwykłych klasach
   (b) **właściwości** (ang. properties)
       właściwość "xxx"; metody getXxx() i setXxx(?); "setters/getters"
   (c) **zdarzenia** (ang. events)
       takie same jak te używane przez komp. wizualne, czyli interf. *Listener, metody add*Listener, remove*Listener

3. Uwaga: komponenty wizualne AWT/SWING spełniają definicję JB (czyli SĄ komponentami JB)

4. Komponentów JavaBeans można używać: w toolbox-ach builderów (np NetBeans), w stronach JSP, w stronach www na podobieństwo appletów, w Jacl-u, ...

5. Komponent JB zawiera informację o sobie samym (introspekcja);
   jest to introspekcja wyższego poziomu niż w przypadku zwykłych klas Javy;
   Komponent JB może zawierać kod edytora właściwości który jest używany przez builder

JavaBeans c.d.

Przykład klasy spełniającej wymagania JB ...

```
public class MojKomponent {
  private int licznik=0;
  private Vector targets= new Vector();
    // pamięta wszystkich "słuchaczy" (obserwatorów) zdarzenia odczytanoLicznik
  public MojKomponent () {}

  // def wlasciwosci "licznik"
  public void setLicznik(int _licz) {
    licznik=_licz;
  }
  public int getLicznik() {
    // powiadamiamy "sluchaczy" ...
    for(int i=0; i<targets.size(); i++) {
      ((OdczytanoListener)targets.elementAt(i)).odczytanoLicznik(
        new OdczytanoEvent(this,licznik)
      );
    }
    return licznik;
  }
```

```
  // dodawanie/usuwanie "sluchaczy" oczekujacych na zdarzenia interf OdczytanoListener
  public void addOdczytanoListener(OdczytanoListener l) {
    targets.addElement(l);
  }
  public void removeOdczytanoListener(OdczytanoListener l) {
    targets.removeElement(l);
  }
}
```

```java
// interfejs Listener opisujący zbiór możliwych zdarzeń (dziedziczy z EventListener)
public interface OdczytanoListener extends EventListener {
  void odczytanoLicznik(OdczytanoEvent e);
    // można zdef więcej zdarzeń!
}

// def klasy opisującej zdarzenie (dziedziczy z EventObject)
public class OdczytanoEvent extends EventObject {
  public OdczytanoEvent(MojKomponent k, int licznik) {
    super(k);
    this.licznik=licznik;
      // nie mozna wywolywac getLicznik() bo to generuje zdarzenie!
  }
  public int licznik;
}
```

Obsługa XML w Javie

1. XML to "samoopisujący się" sposób zapisu dokumentów (XML pokazuje strukturę dokumentu i pozwalający domyśleć się przeznaczenia składników dokumentu)

2. dokumenty XML składają się głównie z zagnieżdżonych elementów posiadających atrybuty

3. *zastosowanie XML*: dokumenty zapisane w XML mogą być w dużym stopniu czytane przez programy które nic nie wiedzą o programach które stworzyły dokument

4. dokumenty XML mogą być przetwarzane jako "drzewo DOM" (= Document Object Model); dokument może być odczytany z pliku i "sparsowany", powstaje wtedy drzewo DOM (w pamięci komputera), można przeglądać i modyfikować to drzewo, na końcu można je zapisać spowrotem do pliku

5. standard XML i pokrewne jest tworzony przez: http://www.w3.org/TR/
   np opis DOM znajduje się tu: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

Przykład dokumentu XML ...
"book" i "chapter" to elementy (zagnieżdżone), "ID*" to atrybuty

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <chapter ID="ch01" ID2="ch01_2" ID3="ch01_3">
    qqqqqqqqqqqqqqqqqqqqqq
  </chapter>
  <chapter>
    wwwwwwwwwwwwwwww
  </chapter>
  <chapter>
    eeeeeeeeeeeeeeeeeeeeeeee
  </chapter>
  <chapter>rrrrrrrrrrrrrrrrrr
    <sec ID="???" QQQ="qqq???">rrrrrrrrrrrrrrr1</sec>
    <sec>rrrrrrrrrrrrrrr2</sec>
    rrrrrrrrrrrrrr-srodek
    <sec>rrrrrrrrrrrrrrr3</sec>
    rrrrrrrrrrrrrr-koniec
  </chapter>
  <chapter IDREF="ch01"/>
  <chapter IDREF="ch01"/>
</book>
```

Przetwarzanie XML w Javie:

```
// ... pokazać przykład dom01.java !!!
```

Protokół HTTP w Javie: Servlet-y i strony JSP

???