Składnia

komentarze	`// /* */ /** */`
litery	`'A'..'Z','a'..'z','\udddd'  dddd>00C0`
identyfikatory	pierwszy znak=`'_','$'` lub litera, następne dodatkowo `'0'..'9'`
liczby dziesiętne	Ddddd (D $\neq$ 0)
liczby ósemkowe	0ddddd
liczby szesnastkowe	0xdddddd
liczby rzeczywiste	IEEE 754 (zapis z literami: E, F, D)
znaki	Unicode

Znaki specjalne

`\n`	NL - nowa linia	`\`	kontynuacja
`\r`	CR - początek wiersza	`\\`	"back slash"
`\t`	HT - tabulator	`\'`	apostrof
`\f`	FF - nowa strona	`\"`	cudzysłów

`\ddd`	znak o kodzie ósemkowym `ddd`
`\xdd`	znak o kodzie szenastkowym `dd`
`\udddd`	znak kodu Unicode `dddd`

Operatory

`+ - * / %`	operatory arytmetyczne
`! && ||`	operatory logiczne
`~ & | ^`	operatory bitowe
`<<`	przesunięcie w lewo
`>>`	przesunięcie w prawo z roszerzeniem znaku
`>>>`	przesunięcie w prawo z uzupełnieniem zerami
`< > <= >= == !=`	porównania
`++ --`	inkrementacja i dekrementacja
`= += -= *= /=`	podstawienia
`&= |= ^= %=`	podstawienia
`<<= >>= >>>=`	podstawienia
`? :`	wyrażenie warunkowe
`(){}[];,.`	inne

Typy proste

typ	opis
`byte`	8 bitowa liczba ze znakiem
`short`	16 bitowa liczba ze znakiem
`int`	32 bitowa liczba ze znakiem
`long`	64 bitowa liczba ze znakiem
`float`	32 bitowa liczba rzeczywista ze znakiem
`double`	64 bitowa liczba rzeczywista ze znakiem
`char`	16 bitowy znak kodu Unicode
`boolean`	wartość logiczna `true` lub `false`

Wartości początkowe

`0, null, false.`

Stałe

```
public static final STAŁA=17
```

Przykłady:

```
math.PI
Color.red
Color.blue
```


Opakowania (wrappers)

typy: `Integer, Long, Float, Double, Character, Boolean`

Przykłady:

```
Integer I = new Integer(123);
Character CH = new Character('z');
Double D = I.doubleValue();
String s = I.toString();

StringBuffer buff;
double d = Double.valueOf(buff.toString()).doubleValue();
```

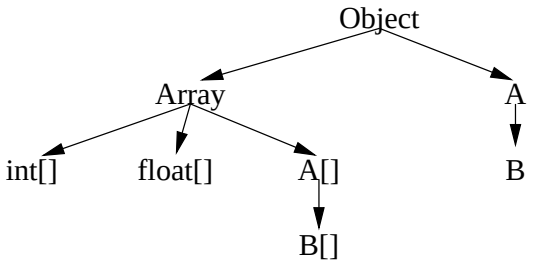

Inicjalizatory

```
class Abc {
  static int arr[] = new int[7];
  static { .... } // inicjalizator
  Abc() { .... } // konstruktor
}

int a = 3 * b; // błąd!!
int b = 4;

int c = 3 * d; // OK
static int d = 5;
```


Tablice

```
 int[] a;          ⇔  int a[];
 byte[] f(int n);  ⇔  byte f(int n)[];

for (i=0; i<a.length; i++) {
  a[i]=i;
  }
char[][]x = new char[14][];
x[0] = new char[7];
```

Hierarchia klas tablicowych



Klasy

```
/** Komentarz dokumentacyjny */
public class Klasa
extends Superklasa
implements Interfejs {
  Klasa() { super(17); ... }
  Klasa(int i) { this(); j=i; ...}
  void finalize() { ... }
  static public void main(String[] args) {...}
}
```


Zakres dostępu

`public`	Dowolna klasa niezależnie od tego do jakiego pakietu należy ma prawo dostępu do zmiennych zdefiniowanych jako publiczne.
`private`	Tylko obiekty klasy, która zdefiniowała tą zmienną mają prawo dostępu do niej. Nawet obiekty klas potomnych nie będą miały dostępu.
`protected`	Tylko obiekty klas definiujących zmienną i obiekty klas potomnych mają prawo dostępu.
`(bez niczego)`	Zmienna jest dostępna bez ograniczeń w ramach pakietu. (Jest to zakres domyślny.)

Modyfikatory

`final`	Nie można zdefiniować klas potomnych.
`abstract`	Nie można tworzyć obiektów danej klasy.
`native`	Metoda zaimplementowana w innym języku.
`synchronized`	Klasa jest jest monitorem Hoare.

Dziedziczenie

Najczęściej używane klasy bazowe:

`Object`	tylko elementarne własności
`Applet`	aplety dla WWW
`Thread`	wątki współbieżne
`Panel`	interfejsy użytkownika

Interfejsy

metody	`public abstract`
zmienne	`final public static`

```
interface Superinterfejs {
  public int f();
}
interface Interfejs extends Superinterfejs {...}
class Klasa implements Interferjs {
  public int f() {...}
}
```


Instrukcje

```
if (...) ... else ...
switch (..) { case ..: ... default: ... }
break ...;
continue ...;
return ...;
for (...; ...; ...) ...
while (...) ...
do ... while (...);
```


Wyjątki

```
class UjemnyWyjątek extends Exception {
}
...
int f(int i) throws UjemnyWyjątek {
  if (i < 0)
    throw new UjemnyWyjątek
}
...
try {
  f(k);
} catch (UjemnyWyjątek e) {
}
```

Łańcuchy

```
StringBuffer sb = new StringBuffer( "Cześć!" );
String s = sb.toString();
char c;

while ((c = (char) System.in.read()) != -1)
  sb.appendChar(c);
```


Wątki

`start()`	uruchomienie wątku
`stop()`	zakończenie działania wątku
`suspend()`	zawieszenie wątku
`resume()`	wznowienie wątku
`sleep()`	zawieszenie się wątku na określony czas
`wait()`	oczekiwanie na zdarzenie
`notify()`	wygenerowanie zdarzenia
`notifyAll()`	poinformowanie wszystkich wątków

Wątki

```
class X implements Runnable {
  public void m() {
    Thread t = new Thread();
    t.target = this;
    t.start();
  }
  public void run() {
    // treść wątku
  }
}
```


Prosty program

```
class Echo {
  static public void main(String args[]) {
    int i = 0;
    while (i < args.length) {
      System.out.println("Arg[" + i + "]=" + args[i++]);

    }
  }
}
```

Prosty program - rezultaty

```
dss<jurek>(98)$ javac Echo.java
dss<jurek>(99)$ java Echo a1 b2 c3
Arg[0]=a1
Arg[1]=b2
Arg[2]=c3
```


Najprostszy aplet

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
       g.drawString("Hello world!", 50, 25);
   }
}
```


Najprostszy aplet - plik HTML

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>

Here is the output of my program:
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```


Najprostszy aplet - rezultat

```
dss<jurek>(99)$ appletviewer HelloWorld.html
```

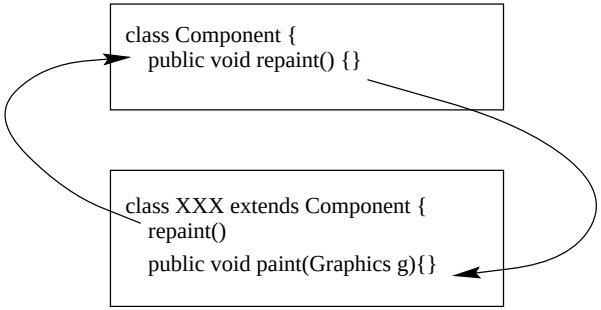
Cykl życiowy apleta

Akcja	metoda
przeglądarka pobiera aplet	init()
użytkownik otwiera stronę	start()
użytkownik opuszcza stronę	stop()
użytkownik opuszcza przeglądarkę	destroy()

Program - aplet

```
public class Świder extends Applet {
  public void init() { resize(200,60); }
  public void paint(Graphics g) { ..... }
  public static void main(String args[]) {
    Świder s = new Świder();
    s.init();
    Frame f = new Frame("Ni pies ni wydra"};
    f.resize(200,60);
    f.add("Center", s);
    f.show();
  }
}
```


AWT



Układacze

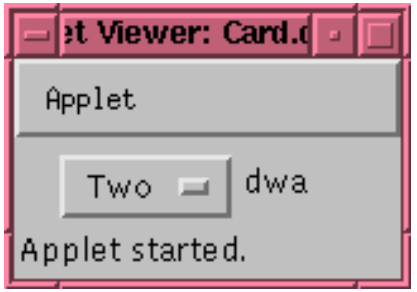
- BorderLayout
- CardLayout
- FlowLayout
- GridLayout
- GridBagLayout

BorderLayout

BorderLayout

```
import java.awt.*;
import java.applet.Applet;

public class Border extends Applet {
  public void init() {
    setLayout(new BorderLayout());
        add("North", new Button("North"));
    add("South", new Button("South"));
    add("East", new Button("East"));
    add("West", new Button("West"));
    add("Center", new Button("Center"));
  }
}
```


CardLayout

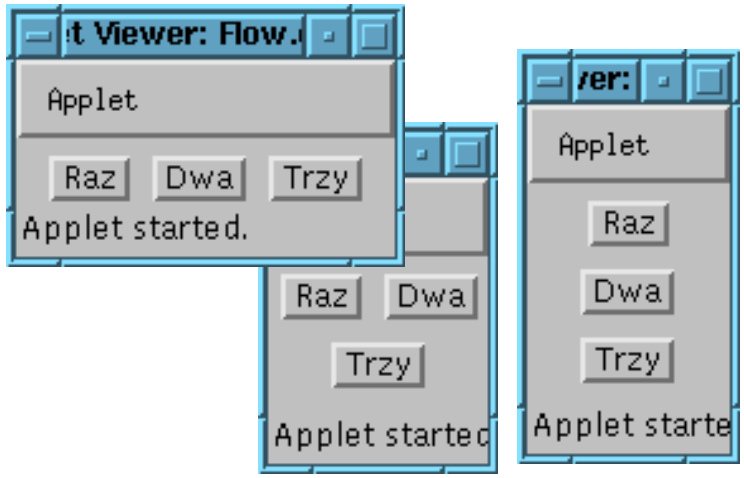


CardLayout

```
import java.awt.*;
import java.applet.Applet;

public class Card extends Applet {
  Panel w = new Panel();

  public void init() {
    Choice c = new Choice();

    w.setLayout(new CardLayout());
    w.add("One", new Label("jeden"));
    c.addItem("One");
    w.add("Two", new Label("dwa"));
```

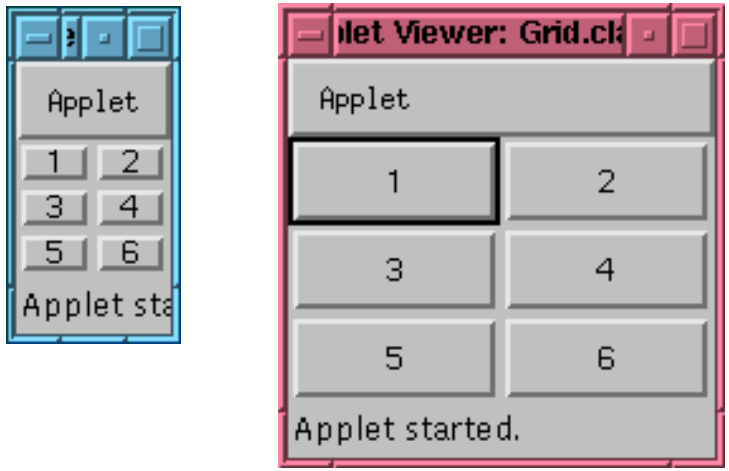
```
    c.addItem("Two");
    w.add("Three", new Label("trzy"));
    c.addItem("Three");
    add(c);
    add(w);
  }

  public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Choice) {
      ((CardLayout)w.getLayout()).show(w,(String)arg);
      return true;
    }
    return false;
  }
}
```


FlowLayout




```
import java.awt.*;
import java.applet.Applet;
public class Flow extends Applet {
Button raz, dwa, trzy;
public void init() {
    raz  = new Button("Raz");
    dwa  = new Button("Dwa");
    trzy = new Button("Trzy");
    add(raz);
    add(dwa);
    add(trzy);
  }
}
```


GridLayout



GridLayout

```
import java.awt.*;
import java.applet.Applet;
public class Grid extends Applet {
  public void init() {
    setLayout(new GridLayout(3,2));
        add(new Button("1"));
    add(new Button("2"));
    add(new Button("3"));
    add(new Button("4"));
    add(new Button("5"));
    add(new Button("6"));
  }
}
```


GridBagLayout