Język Java

Literatura

- Dokumentacja dostępna on-line w witrynie https://docs.oracle.com/javase/tutorial/
- Bruce Eckel Thinking in Java
  wydanie polskie Helion
  lub książka dostępna on-line w języku angielskim
  http://www.mindview.net/Books/TIJ/
- Cay S. Horstmann, Gary Cornell Core Java (Java Podstawy)

Technologia Java

Java jest zarówno **językiem** programowania, jak i **platformą**.

Platforma

Platforma jest środowiskiem zawierającym sprzęt i oprogramowanie, w którym można uruchamiać programy.

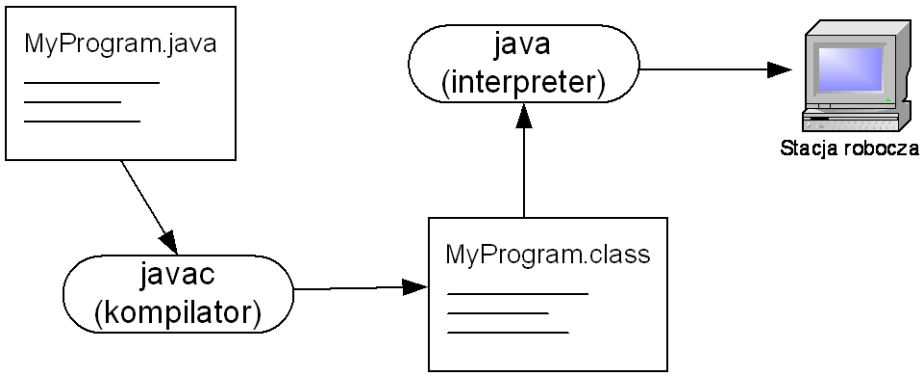
Platforma może być opisana jako kombinacja systemu operacyjnego i sprzętu. Przykładowe platformy to Windows 2000, XP, Linux, Solaris, MacOS.

Programy dedykowane dla danej platformy są ściśle związane ze sprzętem i systemem operacyjnym.

Bytecode

Java uniezależnia się od platform sprzętowych przez wprowadzenie specjalnego formatu zapisu instrukcji programu – *bytecode*.

Język Java jest zarówno językiem *kompilownym* jak i *interpretowanym*.



- Kompilator (*javac*) nie tworzy kodu dla docelowej platformy (Windows, Linux, Solaris), ale dokonuje translacji kodu do postaci przejściowej *bytecode*
- Interpreter odczytuje kolejne instrukcje zapisane jako *bytecode* , tłumaczy je na instrukcje docelowej platformy i wykonuje je.

Postać bytecode może być traktowana jak rozkazy maszynowe dla wirtualnej maszyny Java (*Java VM*). Rozkazy te są w pełni niezależne od platformy.

Każdy interpreter języka Java (działający jako samodzielna aplikacja, moduł wbudowany w przeglądarkę lub sprzęt) jest specyficzną implementacją wirtualnej maszyny Java dedykowanej dla danej platformy.

Write once, run anywhere

Program może zostać skompilowany do postaci bytecode na platformie, dla której zaimplementowano kompilator języka Java.

Rozkazy bytecode mogą następnie uruchomione na dowolnej wirtualnej maszynie Javy.

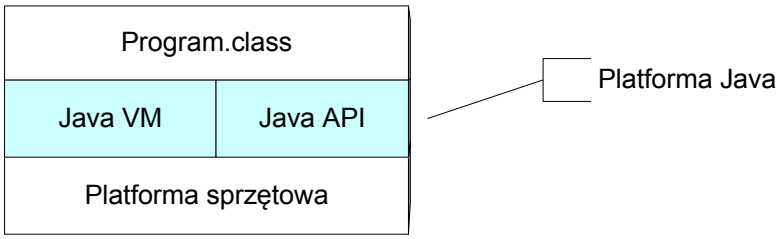
Zazwyczaj zarówno kompilator, jak i większość kodu bibliotek jest dystrybuowana w formacie bytecode.

Platforma Java

Platforma Java jest wyłącznie platformą programową nadbudowaną na platformy sprzętowe.

Składa się na nią:

- Java Virtual Machine (*Java VM*)
- Java Application Programming Interface (*Java API*)



Java VM

- Wirtualna maszyna jest podstawą platformy Java. Jest ona odpowiedzialna za niezależność od platformy sprzętowo-programowej, pozwala zachować niewielkie rozmiary skompilowanego kodu programów języka Java, a także dzięki wbudowanym zabezpieczeniom chroni użytkowników przed błędnym lub niszczącym działaniem programów.
- Maszyna wirtualna ładuje pliki `class` i wykonuje zawarte w nich rozkazy bytecode.
- Konstrukcja maszyny wirtualnej dostosowania jest do własności języka Java (obsługa prostych typów danych, referencji, zarządzanie stertą, stosem, inicjalizacja obiektów, wywołanie metod, wielowątkowość).
- W przypadku aplikacji o krytycznym czasie wykonania możliwa jest automatyczna kompilacja załadowanego kodu do postaci rozkazów maszynowych danej platformy sprzętowej (JIT – *just in time compilation*)

Java API jest obszernym zbiorem komponentów programowych (interfejs użytkownika, grafika, kolekcje, komunikacja sieciowa, komunikacja z bazami danych, itd.).

Java API zawiera *klasy* i *interfejsy*. Są one zgrupowane w biblioteki nazywane pakietami (*packages*).

W większości komponenty Java API są napisane w języku Java i dostępny jest ich kod źródłowy. W przypadku wąskiej grupy są one zaimplementowane w postaci bibliotek/modułów dla dedykowanych platform (kwestie wydajności i dostępu do specyficznych mechanizmów systemu operacyjnego).

Pojęcia

- Java Runtime Environment (JRE) składa się z maszyny wirtualnej oraz bibliotek.
- Java Development Kit (JDK) zawiera dodatkowe narzędzia wspomagające tworzenie aplikacji:
  - javac – kompilator
  - appletviewer – narzędzie do wyświetlania i testowania apletów
  - javadoc – generator dokumentacji na podstawie komentarzy osadzonych w kodzie
  - jar – narzędzie do tworzenia archiwów klas
  - inne (patrz kartoteka *{jdk}/bin*

Zastosowania

- Aplikacje – samodzielne programy uruchamiane bezpośrednio na platformie Java.
- Aplety– programy ładowane dynamicznie i uruchamiane w przeglądarce internetowej wyposażonej w interpreter języka Java. Aplety muszą respektować określone konwencje programistyczne, np.: dziedziczyć po klasie `Applet`, implementować metody określone w ich cyklu życia: `init()`, `destroy()`, `start()`, `stop()`. Zaletą apletów jest łatwość dystrybucji – nie ma konieczności instalacji oprogramowania. [Technologia zamierająca, podobnie jak Java Web Start.]
- Serwery – samodzielne programy dostarczające usług sieciowych (serwer HTTP – Java Web Server, FTP, mail, serwery aplikacji, itd.) Popularne: Tomcat, JBoss, Glassfish

- Serwlety (*servlet*) – umożliwiają budowę interaktywnych aplikacji internetowych i obecnie coraz częściej stanowią alternatywę dla aplikacji (skryptów) CGI.
  Serwlety są niedużymi programami ładowanymi dynamicznie i uruchamianymi po stronie serwera.
  W odpowiedzi na zapytanie skierowane poprzez internet, odczytują parametry zapytania, wykonują operacje wewnętrzne (np.: na bazie danych) i wysyłają odpowiedź (np.: dynamicznie wygenerowana strona HTML).

Elementy składowe Java API

- **Podstawowe mechanizmy języka** – obiekty, typy wbudowane, tablice, klasa String, klasy realizujące wejście/wyjście, wątki, dostęp do funkcji systemu, itd.
- **Aplety** – klasy umożliwiające tworzenie i uruchamianie apletów
- **Komunikacja sieciowa** – URL, implementacje protokołów TCP, UDP, IP
- **Wsparcie dla internacjonalizacji** – obsługa *locale*, automatyczne ładowanie zasobów (tekstów)
- **Bezpieczeństwo** – podpisy elektroniczne, zarządzanie kluczami publicznymi i prywatnymi, certyfikaty
- **Oprogramowanie komponentowe** – możliwe jest tworzenie i wykorzystywanie komponentów JavaBeans™ . Beans są samodzielnymi komponentami wielokrotnego użytku, które mogą zostać wykorzystane w aplikacjach, apletach, serwletach lub bardziej złożonych komponentach. Mogą być one wizualnie konfigurowane za pomocą odpowiednich narzędzi.
- **Dostęp do baz danych** (Java Database Connectivity – JDBC™) – zapewnia ujednolicony interfejs dostępu do większości relacyjnych baz danych.

Java jako język ma następujące cechy:

- Prostota
  - Obiektowość
  - Dopuszcza rozproszoną architekturę aplikacji
  - Jest językiem interpretowanym
  - Wydajność
  - Bezpieczeństwo
  - Niezależność od architektury sprzętowej
  - Przenośność
  - Wielowątkowość

Prostota

Język został zaprojektowany, aby ułatwić pisanie kodu wolnego od błędów.

- Zrezygnowano z konstrukcji języka C/C++ promujących powstanie błędów (mieszanie typów `bool` i `int`, przesłanianie, `goto`, wskaźniki, brak automatycznej inicjalizacji, itp.)
- Znaczny procent błędów języków C/C++ popełniany jest przy alokacji i zwalnianiu pamięci. W języku Java proces przydzielania pamięci i jej zwalniania odbywa się automatycznie pod kontrolą Java VM. W szczególności obiekty nie są programowo usuwane.

Obiektowość

Język Java implementuje typowe paradygmaty programowania obiektowego:

- Program składa się wyłącznie z klas i obiektów.

Brak jest samodzielnie istniejących funkcji. Wszystkie funkcje muszą być zaimplementowane jako metody obiektów.

Obiekty składają się z *metod* i *atrybutów* (pól składowych).

- Program jest siecią obiektów, które nawzajem przesyłają do siebie komunikaty.
- Każdy obiekt ma własną pamięć, na którą składają się inne obiekty.

Aby stworzyć nowy rodzaj obiektu możemy zgrupować obiekty niższego typu (także zmienne proste).

- Każdy obiekt ma typ (należy do pewnej klasy)
- Wszystkie obiekty danego typu mogą przyjmować te same komunikaty. (Mają ten sam interfejs.). Możliwe jest dziedziczenie, a także polimorficzne wywołanie metod.
- Możliwa jest kontrola dostępu do elementów składowych obiektów (`public`, `protected`, `private`)

Bezpieczeństwo

- W języku Java brak jest wskaźników, za pomocą których możemy sięgnąć do dowolnego obszaru pamięci. Dostęp do pamięci odbywa się pod kontrolą bezpiecznej maszyny wirtualnej. Kontrolowany jest między innymi dostęp do elementów tablic.
- Interpreter ma także wbudowaną kontrolę dostępu do zasobów systemowych (plików, sprzętu, drukarek, możliwości otworzenia połączeń sieciowych).
- Język ma wbudowaną mocną kontrolę typów. Wbudowana informacja RTTI (Run Time Type Identification) nie pozwala na nieuprawnione rzutowanie.

Programy języka Java są dynamiczne linkowane

Kod źródłowy umieszczany jest w plikach o rozszerzeniu `.java`. Definicje klas zapisuje się w odrębnych plikach o nazwach identycznych z nazwą klasy (istotne jest rozróżnienie między małymi i dużymi literami)

Podczas kompilacji tworzone są pliki zawierające definicje, tablice symboli i kod metod klas w postaci rozkazów bytecode. Pliki te mają rozszerzenie `.class`. Zazwyczaj w wyniku kompilacji kodu pojedynczej klasy powstaje jeden plik (chyba że klasa ma klasy zagnieżdżone).

Podczas kompilacji kompilator przeszukuje bieżący katalog oraz katalog wskazany przez zmienną `CLASSPATH`, aby znaleźć wszystkie klasy, do których nazw istnieją odwołania w kodzie źródłowym.

Ładowanie klas

Podczas wykonania programu interpreter ładuje kod klas w momencie, kiedy tworzone są po raz obiekty danego typu.

- Pliki `class` zawierające rozkazy bytecode są ładowane z pliku lokalnego lub z sieci przez moduł *class loader*.
- Następnie pliki te są sprawdzane przez moduł *class verifier*, aby upewnić się, czy mają one poprawny format. Dalej sprawdzane są własności typu: użycie stosu, zgodność argumentów funkcji, poprawność symboli stałych, inicjalizacja atrybutów, itd. Ma to zapewnić poprawność wykonania programu a w szczególności uniknąć błędów dostępu do pamięci, które mogłyby zagrozić spójności systemu operacyjnego.
- Po dokonaniu weryfikacji następuje właściwe wykonanie kodu przez *interpreter*.

Garbage collection

Programy w języku Java nie zajmują się alokacją i zwalnianiem pamięci jawnie.

Programowo można jedynie stworzyć obiekt za pomocą operatora `new`. W tym momencie maszyna wirtualna Javy wczyta kod klasy, jeśli nie był on wcześniej załadowany, przydzieli pamięć oraz przeprowadzi niezbędną inicjalizację pól składowych, następnie wywoła konstruktor.

Obiekty nigdy nie są jawnie usuwane. Za usuwanie obiektów i zwalnianie pamięci odpowiada mechanizm *garabage collection* – automatycznego usuwania nieużywanych obiektów.

Podstawowe różnice pomiędzy C++ i Java

- Brak zmiennych globalnych
- Brak funkcji o zasięgu globalnym
- Brak struktur, unii, pól bitowych
- Brak możliwości bezpośredniej definicji zmiennych typów obiektowych. Obiekty muszą być tworzone za pomocą operatora `new`.
- Brak wskaźników
- Brak `const`
- Brak preprocesora
- Brak dziedziczenia wielobazowego
- Brak możliwości przeciążania operatorów

Fazy tworzenia programu

Tworzenie kodu źródłowego

Za pomocą edytora należy utworzyć kod klasy i zapisać na dysku w pliku z rozszerzeniem `java`.

Nazwa pliku powinna dokładnie odpowiadać nazwie klasy (z uwzględnieniem małych i dużych liter).

Przykład 1

HelloWorld.java

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

Przykład 2

HelloWorldApplet.java

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

public class HelloWorldApplet
extends Applet
{
  public void paint(Graphics g)
  {
    g.drawString("Hello World", 20, 20);
    System.out.println("paint invoked");
  }
}
```

Kompilacja

- Pakiet JDK może zostać pobrany z witryny Oracle http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html. Można też od razu pobrać IDE Netbeans.

- Po instalacji wszystkie narzędzia umieszczone są w katalogu typu: c:**Program Files\Java**\jdk1.8.xxx\bin

  Zakładamy, że katalog ten jest do dany do zmiennej środowiskowej PATH. W przeciwnym przypadku konieczne jest jawne podanie ścieżki.

  - Uruchamiamy kompilację podając komendę:

  ```
  >javac HelloWorld.java
  ```

  lub

  ```
  >javac HelloWorldApplet.java
  ```

W jej wyniku powstaną pliki w formacie `class` `HelloWorld.class` lub `HelloWorldApplet.class` zawierające rozkazy bytecode dla wirtualnej maszyny Java.

Wykonanie

W fazie wykonania wirtualna maszyna Java ładuje pliki class z dysku, przeprowadza ich weryfikację i dynamiczne linkowanie, a następnie uruchamia interpreter.

Aplikacje

Aby uruchomić aplikację `HelloWorld` należy wydać komendę:

```
>java HelloWorld
```

lub jeżeli uruchamiamy z innego katalogu

```
>{ścieżka}java –classpath {katalog z plikiem class} HelloWorld
```

np.:

```
„c:\Program Files\Java\jdkXXXXX\bin\java”
-classpath . HelloWorld
```

Interpreter załaduje kod z lokalnego pliku. Program rozpocznie wykonanie od funkcji

```
public static void main(String args[]).
```

Rezultat:

```
HelloWorld
```

Aplety

- Uruchomienie apletu wymaga dodatkowego kroku. Konieczne jest stworzenie pliku HTML zawierającego odwołanie do apletu.

Przykład

Page.html

```
<HTML>
    <HEAD>
        <TITLE>HelloWorldApplet</TITLE>
    </HEAD>
    <BODY>
        <APPLET
            CODE="HelloWorldApplet.class"
            WIDTH="300" HEIGHT="300">
            </APPLET>
    </BODY>
</HTML>
```

appletviewer

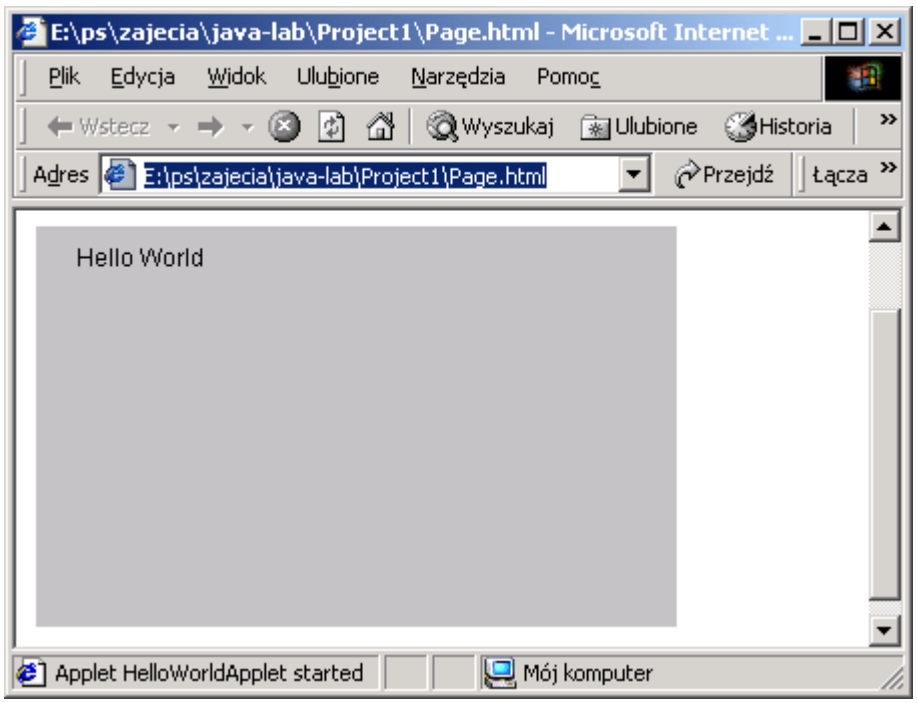
Aby uruchomić aplet należy wydać polecenie

```
>appletviewer page.html
```

- Aplikacja appletviewer załaduje wskazany plik HTML, odnajdzie znacznik <APPLET>...</APPLET> i załaduje, a następnie uruchomi kod klasy wskazany w parametrze `CODE="HelloWorldApplet.class"`

Przeglądarka

Plik HTML może bezpośrednio zostać załadowany do przeglądarki. Kod apletu może zostać załadowany i uruchomiony przez wirtualną maszynę wbudowaną w przeglądarkę.

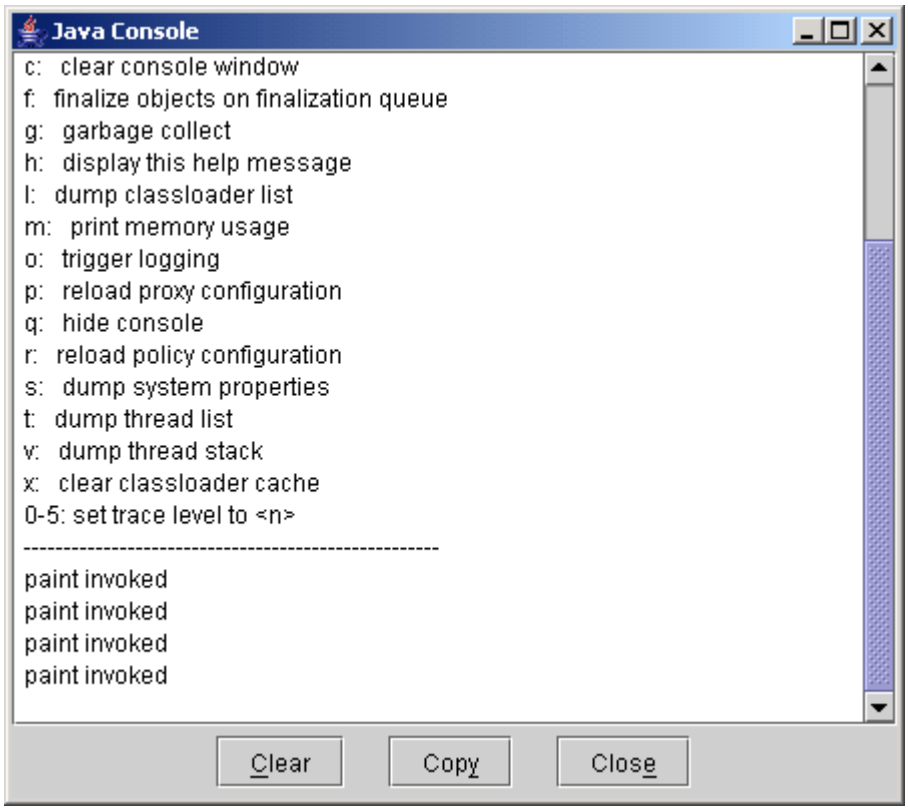


Konsola Java

W metodzie `paint()` apletu znalazła się instrukcja wydruku na konsoli. `System.out.println("…")`

```
public void paint(Graphics g)
  {
    g.drawString("Hello World", 20, 20);
    System.out.println("paint invoked");
  }
```

Konsola Java jest dodatkowym narzędziem wspierającym uruchamianie apletów. Okno konsoli otworzyć zarówno z aplikacji *appletviewer*, jak i przeglądarki. Poza wypisywaniem tekstów pozwala ona na wydawanie komend analizujących stan wirtualnej maszyny (wątki, użycie pamięci), a także manualne uruchomienie mechanizmu *garbage collection*.



Popularne IDE

Każda zintegrowana platforma zapewnia:

- edytor z podświetlaniem składni
- code completion (kontekstowe sugestie kodu podczas pisania)
- narzędzia refaktoryzacji
- debugger
- często profiler
- różne użyteczne wtyczki

Najpopularniejsze:

- NetBeans (promowane przez Oracle, także platforma dla UI)
- Eclipse (dzięki mechanizmowi wtyczek i wielu gotowym rozwiązaniom najpopularniejsza platforma dla pisania programów z interfejsem graficznym)
- InteliJ (najczęściej spotykane na laptopach studentów)
- AndroidStudio (dedykowane dla systemu Android, oferuje emulator)