PODSTAWY PROGRAMOWANIA W JĘZYKU **JAVA**

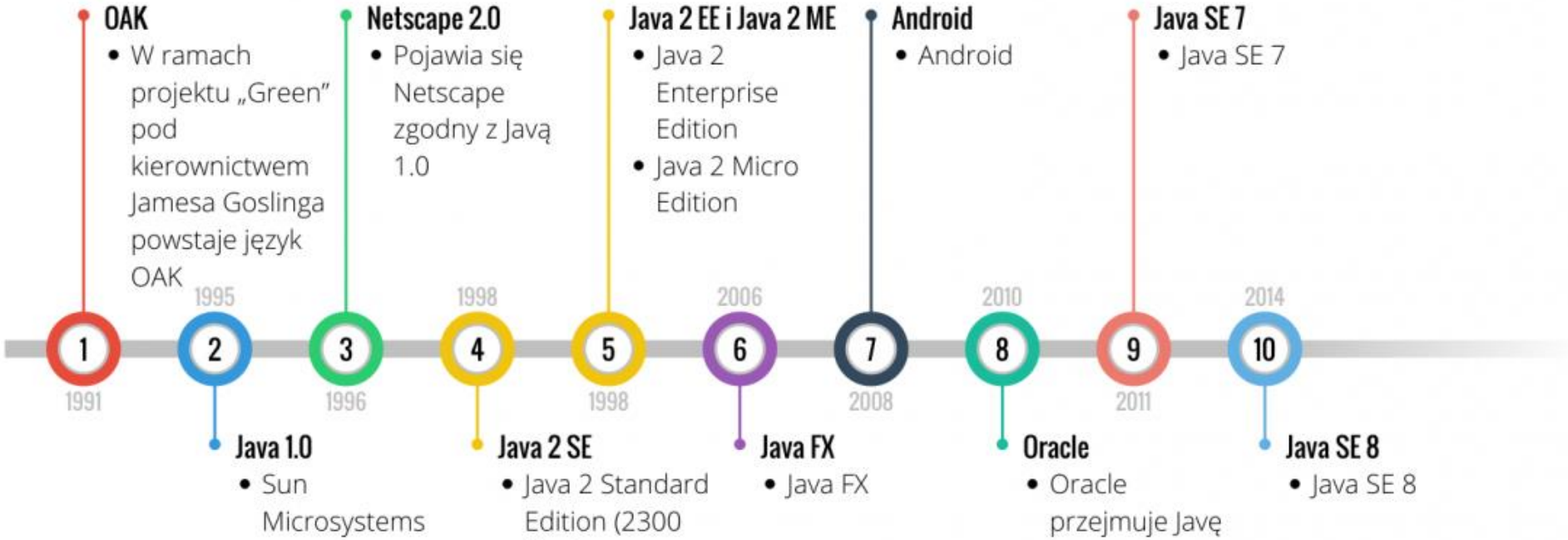
1. Historia Javy
2. Tworzenie programu w języku Java
3. Zmienne, separatory i komentarze
4. Typy proste w Javie
5. Instrukcje w Javie – Instrukcja warunkowa
6. Instrukcje w Javie – Instrukcje iteracyjne
   - pętla while
   - pętla do while
   - Pętla for
   - Pętla foreach
7. Instrukcje w Javie – Instrukcja wyboru
8. Ciekawe skróty

AGENDA

1. Historia Javy
2. Tworzenie programu w języku Java
3. Zmienne, separatory i komentarze
4. Typy proste w Javie
5. Instrukcje w Javie – Instrukcja warunkowa
6. Instrukcje w Javie – Instrukcje iteracyjne
   - pętla while
   - pętla do while
   - Pętla for
   - Pętla foreach
7. Instrukcje w Javie – Instrukcja wyboru
8. Tablice jednowymiarowe
9. Ciekawe skróty

HISTORIA JAVY

Java Historia



Historia Java

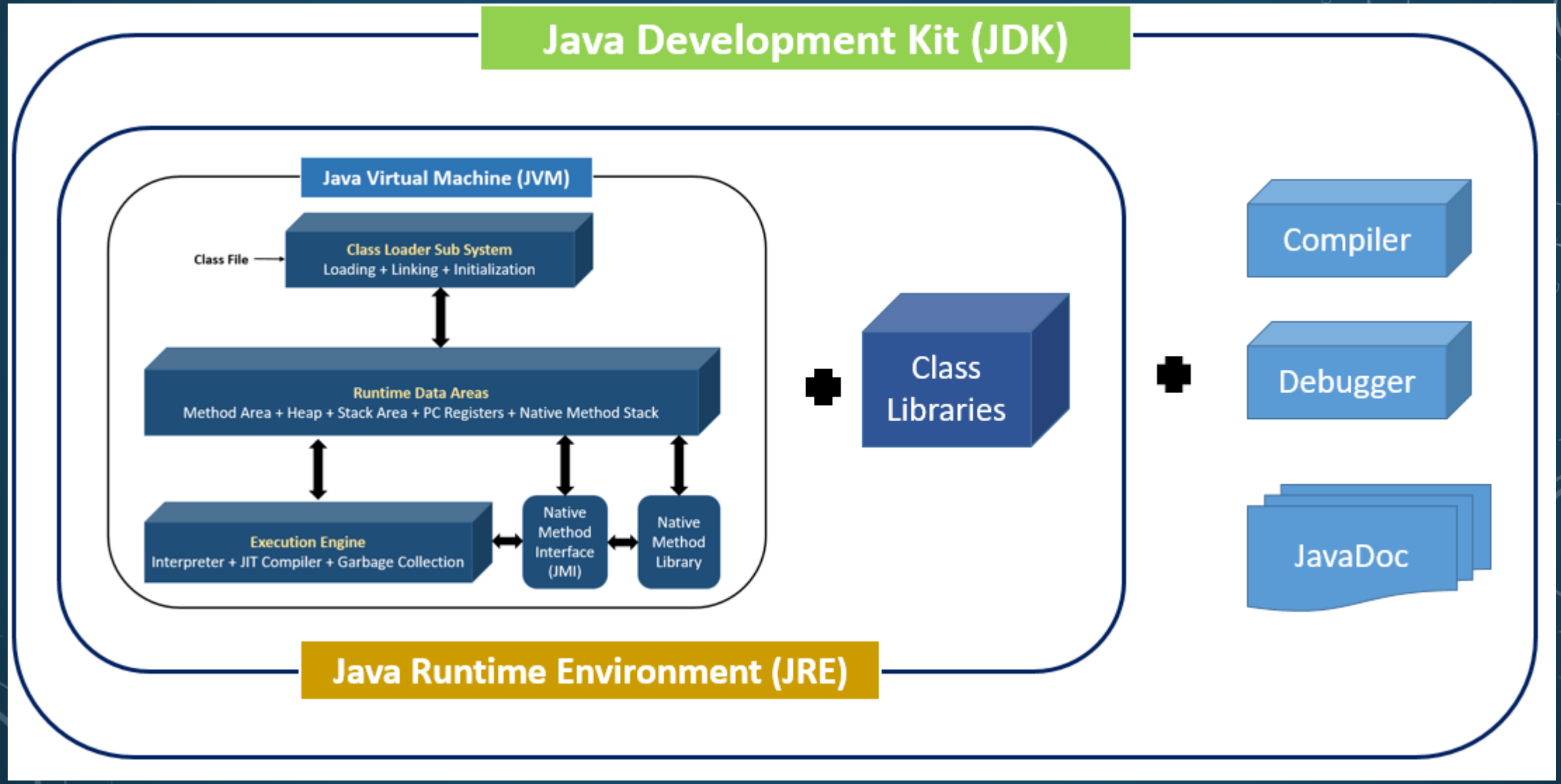
STORM IT

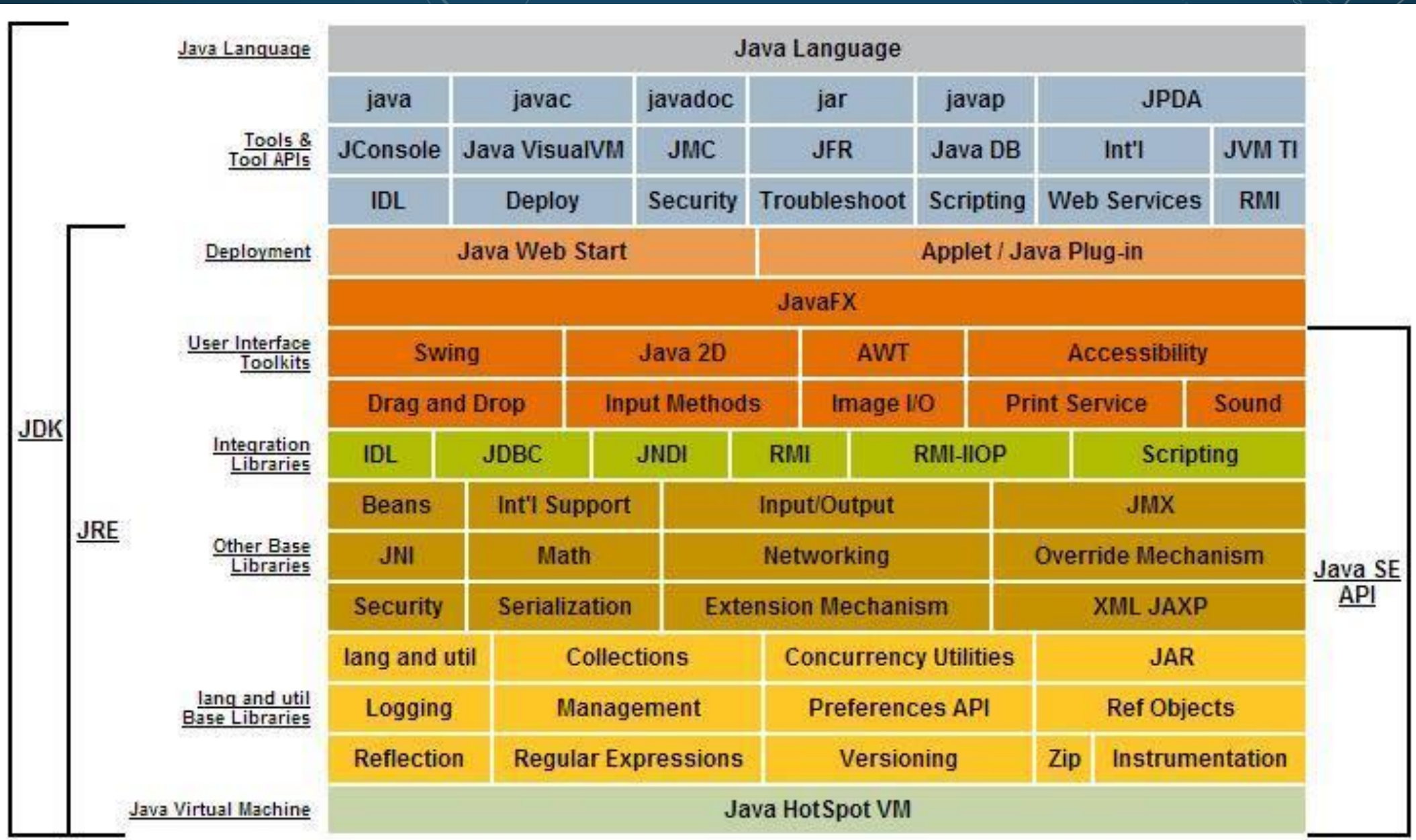
Wersja Javy	Nazwa Kodowa	Liczba class
Java 1.0	OAK	212
Java 1.1	----	504
Java 1.2	Playground	1520
Java .1.3	Kestrel	1842
Java 1.4.0	Merlin	2991
Java 1.5.0	Tiger	3279
Java 1.6.0	Mustang	3793
Java 1.7.0	Dolphin	4024
Java 1.8.0	Spider	4240
Java 1.9.0		6005
Java 1.10.0		6002
Java 1.11.0		4411
Java 1.12.0		4433
Java 1.13.0		4545
Java 1.14.0		4569
Java 1.15.0		

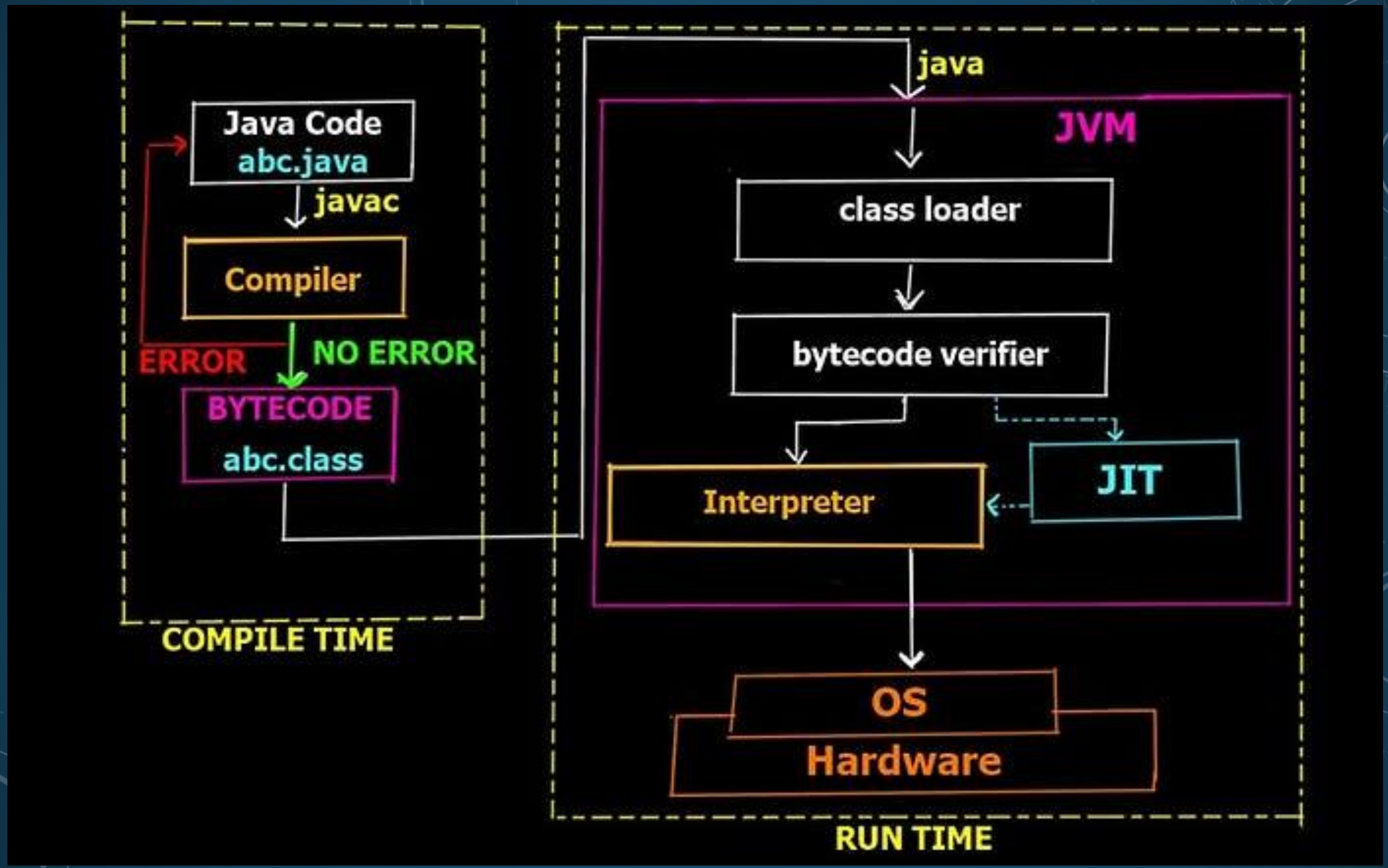
WERSJE JAVY

Przegląd wersji Java

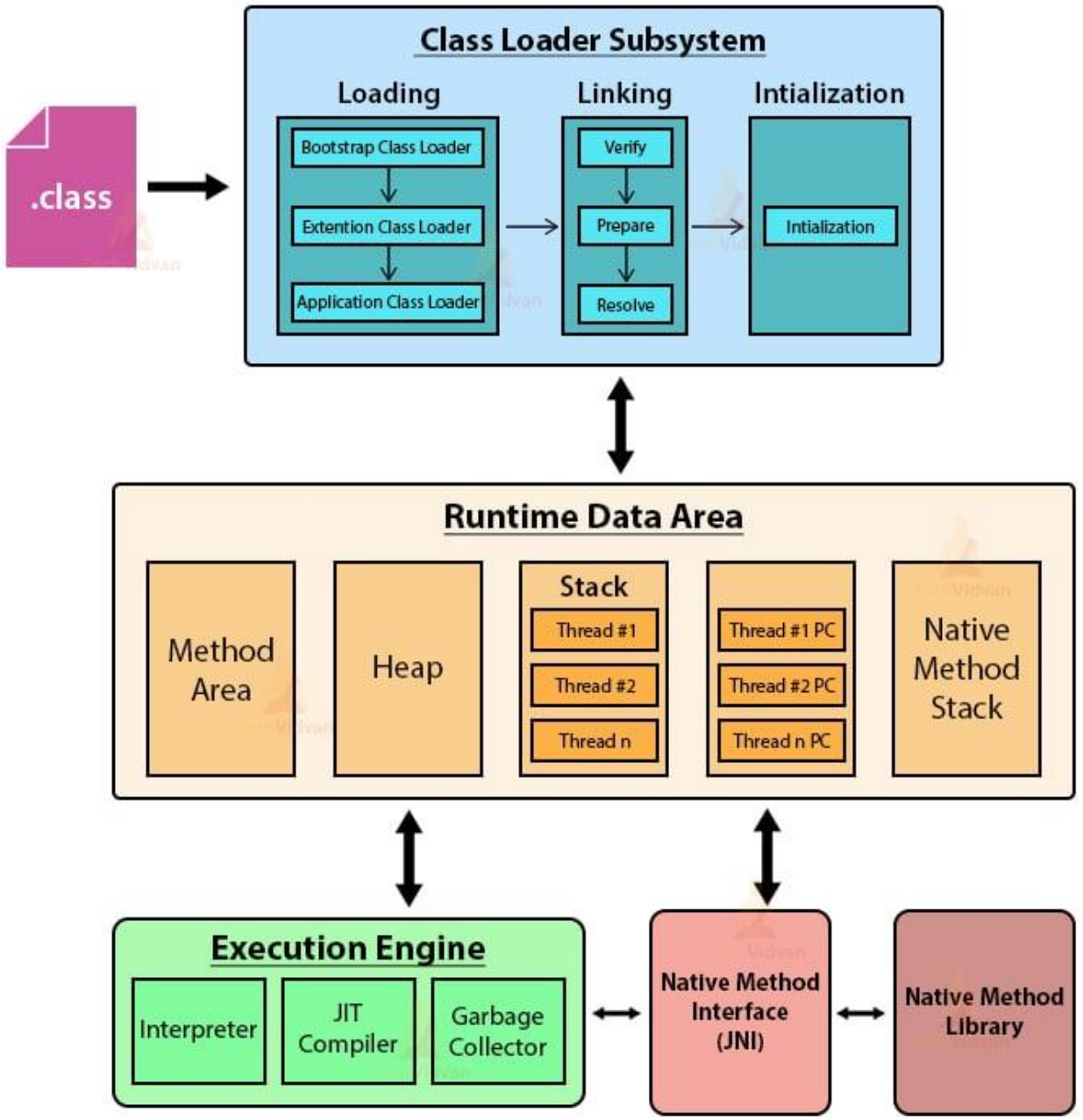
JDK, JRE I JVM


Java Development Kit (JDK)
Java Virtual Machine (JVM)
Class File
Class Loader Sub System
Loading + Linking + Initialization
Runtime Data Areas
Method Area + Heap + Stack Area + PC Registers + Native Method Stack
Execution Engine
Interpreter + JIT Compiler + Garbage Collection
Native Method Interface (JMI)
Native Method Library
Java Runtime Environment (JRE)
Class Libraries
Compiler
Debugger
JavaDoc

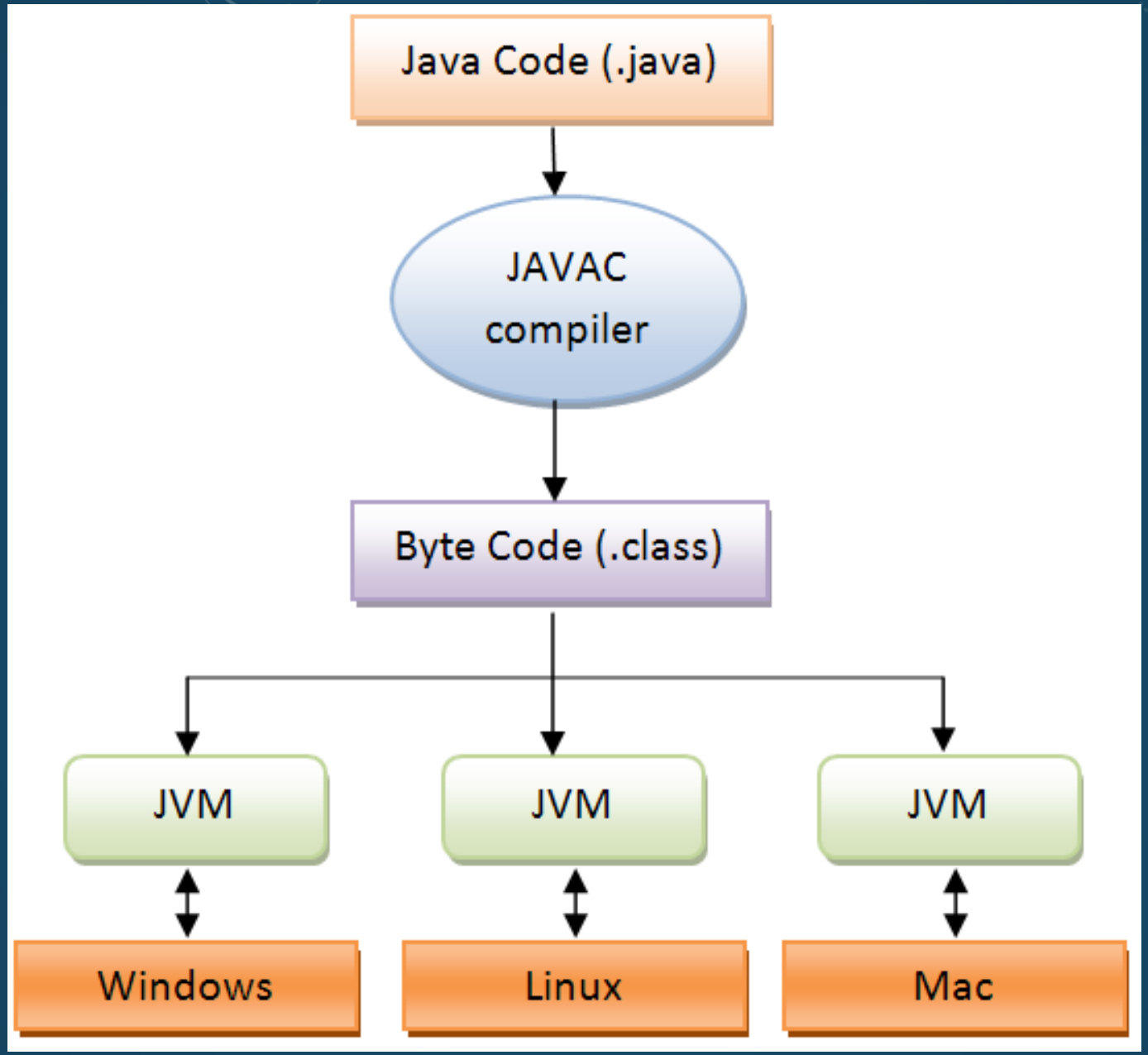

JDK
JRE
Java SE API
Java Language
Java Language
java
javac
javadoc
jar
javap
JPDA
Tools & Tool APIs
JConsole
Java VisualVM
JMC
JFR
Java DB
Int'l
JVM TI
IDL
Deploy
Security
Troubleshoot
Scripting
Web Services
RMI
Deployment
Java Web Start
Applet / Java Plug-in
JavaFX
User Interface Toolkits
Swing
Java 2D
AWT
Accessibility
Drag and Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC
JNDI
RMI
RMI-IIOP
Scripting
Other Base Libraries
Beans
Int'l Support
Input/Output
JMX
JNI
Math
Networking
Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
lang and util Base Libraries
lang and util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences API
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Instrumentation
Java Virtual Machine
Java HotSpot VM


Java Code
abc.java
javac
Compiler
ERROR
NO ERROR
BYTECODE
abc.class
COMPILE TIME
java
JVM
class loader
bytecode verifier
Interpreter
JIT
OS
Hardware
RUN TIME

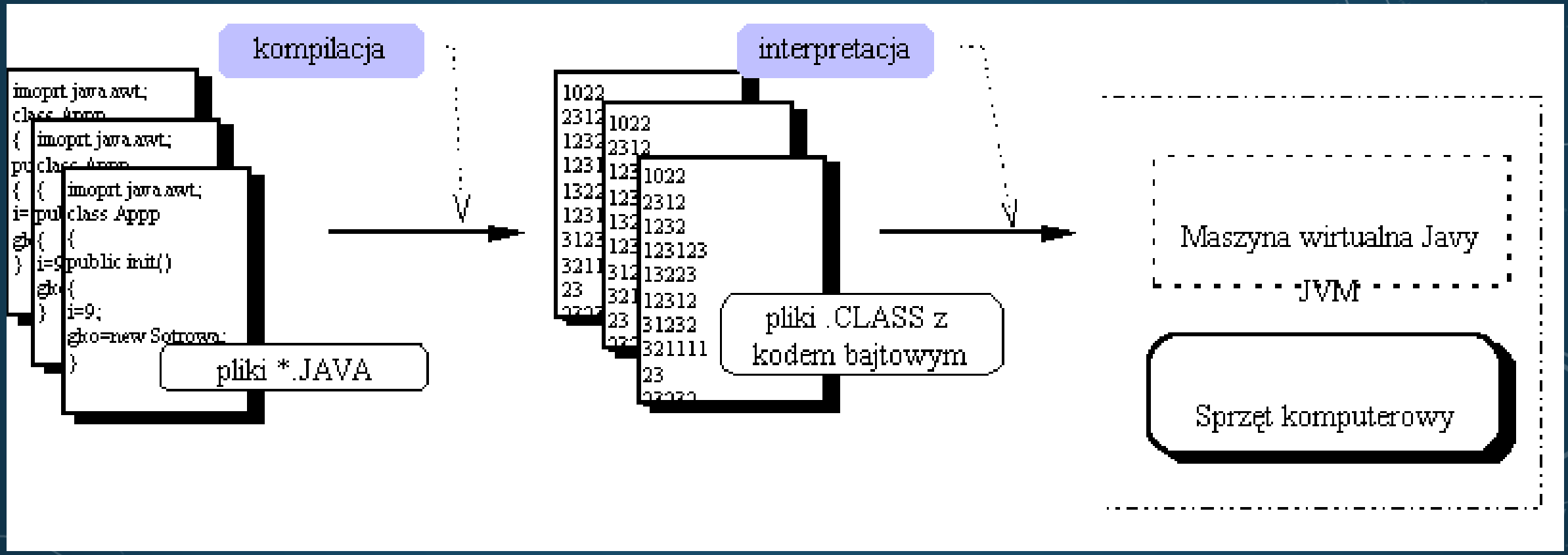
JVM Model


.class
Class Loader Subsystem
Loading
Linking
Intialization
Bootstrap Class Loader
Extention Class Loader
Application Class Loader
Verify
Prepare
Resolve
Intialization
Runtime Data Area
Method Area
Heap
Stack
Thread #1
Thread #2
Thread n
Thread #1 PC
Thread #2 PC
Thread n PC
Native Method Stack
Execution Engine
Interpreter
JIT Compiler
Garbage Collector
Native Method Interface (JNI)
Native Method Library

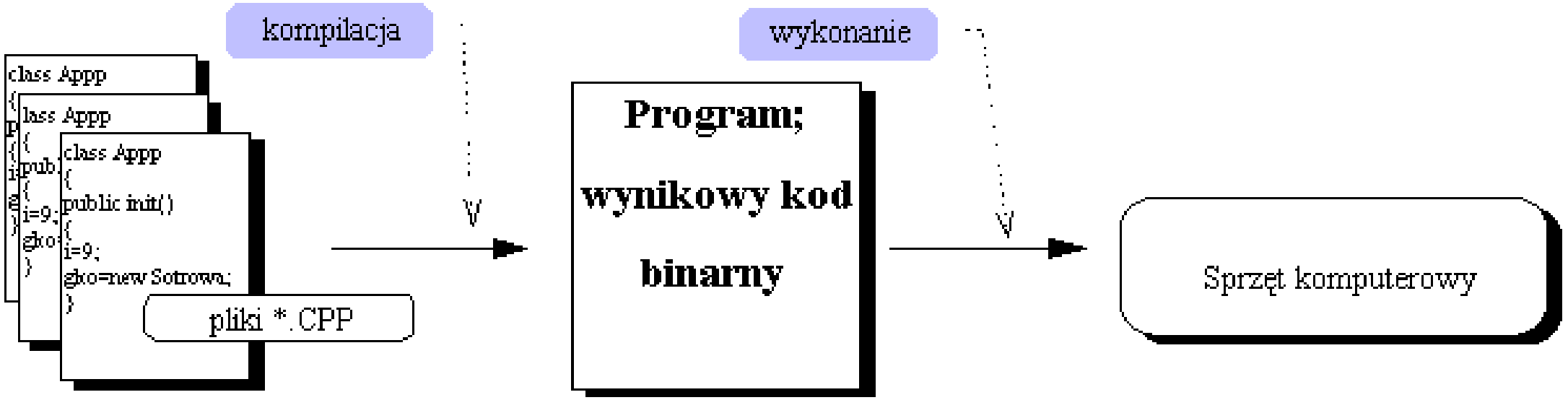
TWORZENIE PROGRAMU W JĘZYKU JAVA


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

KOMPILACJA I WYKONANIE PROGRAMU NAPISANEGO W JAVIE



KOMPILACJA I WYKONANIE PROGRAMU NAPISANEGO W C++.



ZMIENNE, SEPARATORY I KOMENTARZE

POJĘCIE ZMIENNEJ I STAŁEJ

Zmienne, czyli specjalny element języka, który pozwala przechowywać wartości, a następnie odwoływać się do nich po nazwach.

Proces tworzenia zmiennej możemy podzielić na dwa etapy:

- **deklaracja** - tutaj określamy typ i nazwę zmiennej,
- **inicjalizacja (nazywana też często inicjacją)** - nadanie wartości zmiennej.

Zmienne możesz poprzedzić słowem kluczowym **final**. Zadeklarowana w ten sposób zmienna może być tylko raz zainicjowana i będzie się nazywała *zmienną finalną (stałą)*. Próba przypisania do niej wartości po raz drugi zakończy się błędem i program nawet nie przejdzie kompilacji.

- **I sposób:**

DEKLARACJA:

- float area ;
- char grade ;
- String sentence ;
- int payRate;

INICJALIZACJA:

- area = 120.45f;
- grade = ’g’;
- sentence = ” Hello World”;
- payRate = 130;

- **II sposób:**

DEKLARAJA I INICJALIZACJA

- float area = 120.45f;
- char grade = ’g’;
- String sentence = ” Hello World”;
- int payRate = 130;

NOTACJE DLA NAZW ZMIENNYCH

- Notacja Camel Case:
  Przykład: myVariableName, calculateTotalAmount, getUserInfo

- Notacja Pascal Case:
  Przykład: MyClass, CalculateTotalAmount, GetUserInfo

- Notacja Snake Case:
  Przykład: my_variable_name, calculate_total_amount, get_user_info

- Notacja Kebab Case (czyli również Dash Case):
  Przykład: my-variable-name, calculate-total-amount, get-user-info

- Notacja Wielkich Liter (ALL CAPS):

  Przykład: CONSTANT_VALUE, MAX_SIZE, LOGGER

- Notacja Weguglowa (Google Case lub Upper Camel Case):

  Przykład: MyVariableName, CalculateTotalAmount, GetUserInfo

- Notacja Małych Liter (Lower Case):

  Przykład: myvariablename, calculatetotalamount, getuserinfo

- Notacja Węgierska:

Przykład: strName (oznacza zmienną tekstową), iCount (oznacza liczbę całkowitą)

SEPARATORY

Symbol	Nazwa separatora	Zastosowanie
( )	Nawiasy okrągłe	Do tworzenia listy parametrów metod, określenie kolejności działań, zawiera wyrażenia w instrukcjach sterujących, otaczanie typów przy rzutowaniu
{ }	Nawiasy klamrowe	Definiowanie bloku instrukcji sterującej, klasy lub metody, do otaczania wartości inicjalizujących tablicę
[ ]	Nawiasy kwadratowe	Do deklarowania tablic
;	Średnik	Do rozdzielania instrukcji, do pętli for
,	Przecinek	Do deklaracji wielu zmiennych jednakowego typu
.	Kropka	Do odwoływania się do metod i zmiennych obiektów danej klasy, rozdzielanie nazw pakietów, podpakietów i klas
::	Dwukropek	Do tworzenia odwołań do metod i konstruktorów
@	At	Rozpoczyna adnotację

KOMENTARZE W JAVIE

W języku Java wyróżniamy 2 rodzaje komentarzy:

1) Komentarz jednowierszowy: // (skrót: CTRL + /)
2) Komentarz wielowierszowy /* … */ (skrót: CTRL + SHIFT + /)

Formatowanie kodu w Intellij IDEA to: CTRL + ALT + L

TYPY PROSTE W JAVIE

LICZBY CAŁKOWITE

Wyróżniamy cztery typy danych dla liczb całkowitych:

- ***byte*** - 1 bajt - zakres od -128 do 127
- ***short*** - 2 bajty - zakres od -32 768 do 32 767
- ***int*** - 4 bajty - zakres od -2 147 483 648 do 2 147 483 647
- ***long*** - 8 bajtów - zakres od -2^63 do (2^63)-1 (posiadają przyrostek **L**, lub **l**)

LICZBY RZECZYWISTE (LICZBY ZMIENNOPRZECINKOWE)

Wyróżniamy dwa typy danych dla liczb rzeczywistych:

- ***float*** - 4 bajty - max ok 6-7 liczb po przecinku (posiadają przyrostek **F**, lub **f**)
- ***double*** - 8 bajtów - max ok 15 cyfr po przecinku (posiadają przyrostek **D**, lub **d**)

TYP ZNAKOWY

Kolejnym typem jest ***char***, czyli znak. Służy on do reprezentacji pojedynczych znaków. Wartości tego typu będziemy zapisywali pomiędzy apostrofami. Możemy się posługiwać także wartością w postaci liczby szesnastkowej, lub dziesiętnej, odpowiadającej kodowi danego znaku z tabeli Unikodu.

Znaki specjalne
Istnieją również znaki specjalne, które muszą być poprzedzone znakiem backslash **** :

- **\t** - tab
- **\n** - nowa linia
- **\r** - powrót karetki
- **\"** - cudzysłów
- **\'** - apostrof
- **** - backslash

TYP LOGICZNY

Ostatnim typem prostym jest ***boolean***. Reprezentuje on tylko dwie wartości:

- **true** - prawda
- **false** - fałsz

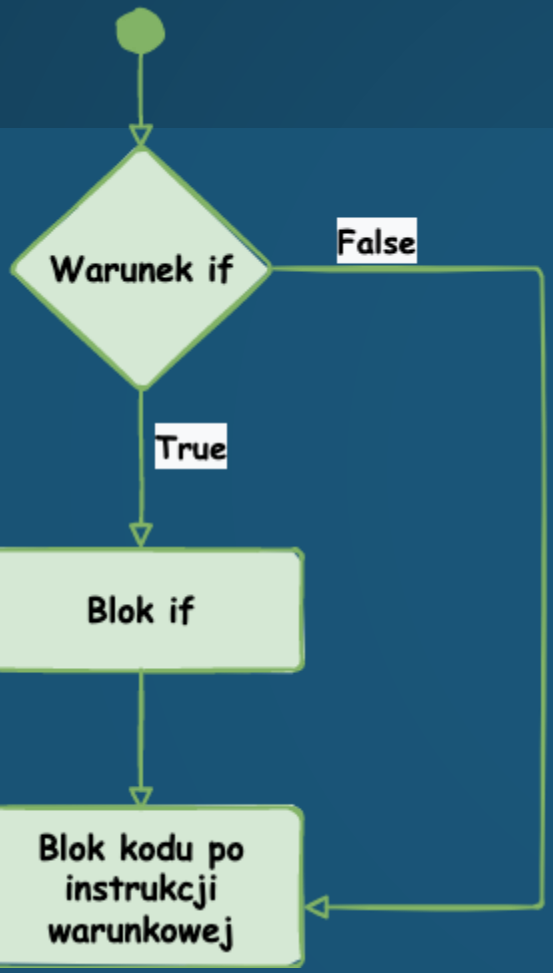
INSTRUKCJE JĘZYKA JAVA

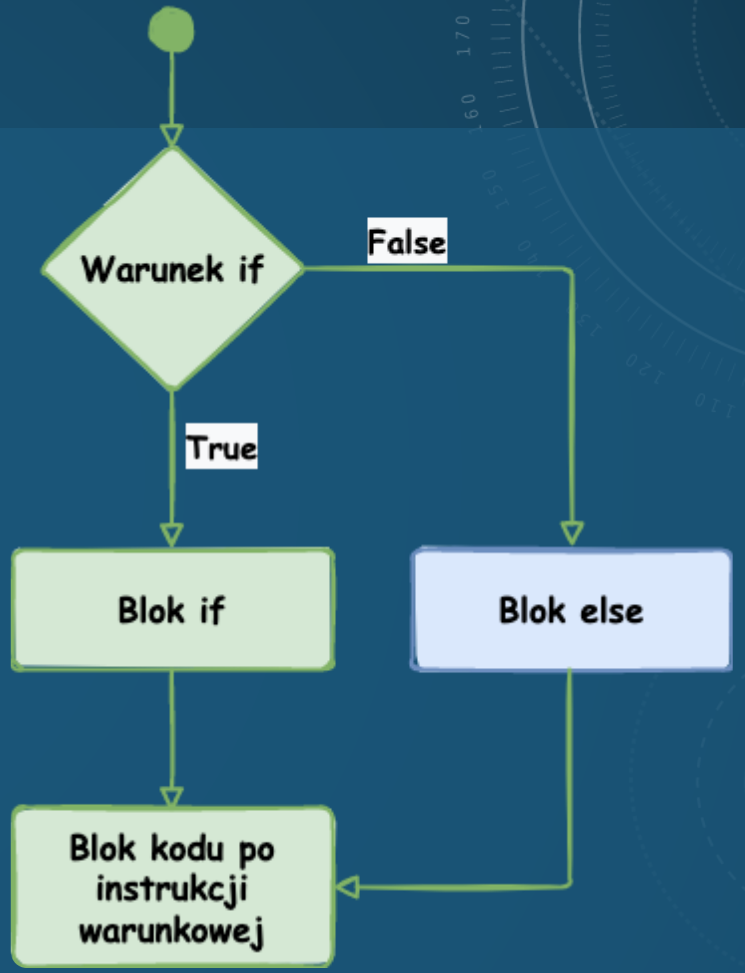
INSTRUKCJA WARUNKOWA

Instrukcje warunkowe **IF ELSE** to konstrukcja języka, dzięki której można rozwidlić ścieżkę wykonywania programu. Przy jej pomocy możemy określić warunki, jakie mają zajść, żeby dany fragment kodu został wykonany.

W wolnym tłumaczeniu można przeczytać ją jako:
jeżeli zajdzie pewien warunek, to zrób 'to', w przeciwnym wypadku zrób 'tamto'.

Wyrózniamy 2 typy instrukcji if:


Warunek if
False
True
Blok if
Blok kodu po instrukcji warunkowej


Warunek if
False
True
Blok if
Blok else
Blok kodu po instrukcji warunkowej

INSTRUKCJE ITERACYJNE

Pętle są jednym z podstawowych narzędzi wykorzystywanych przez programistę. Dzięki nim można wywołać określoną funkcjonalność podaną ilość razy, zamiast za każdym razem wywoływać ją ręcznie.

W Javie mamy do dyspozycji kilka rodzajów pętli:

- **while,**
- **do while,**
- **for,**
- **foreach**.

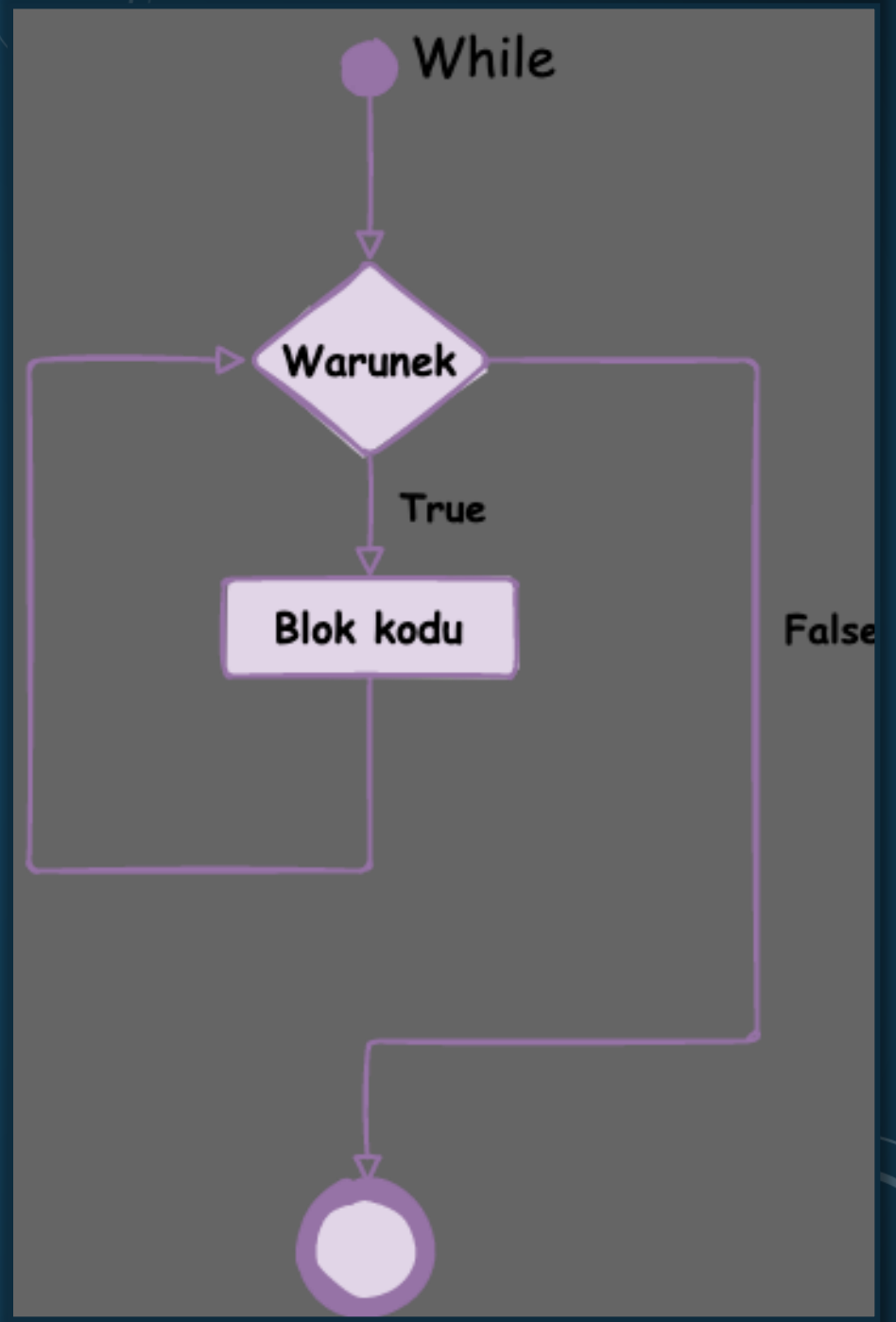
Funkcjonalność wszystkich jest wymienna, a decyzja, którą należy użyć w danym momencie, zależy głównie od kontekstu, w jakim ma być wykorzystana oraz od preferencji programisty.

-

PĘTLA WHILE (ANG. *WHILE LOOP*)

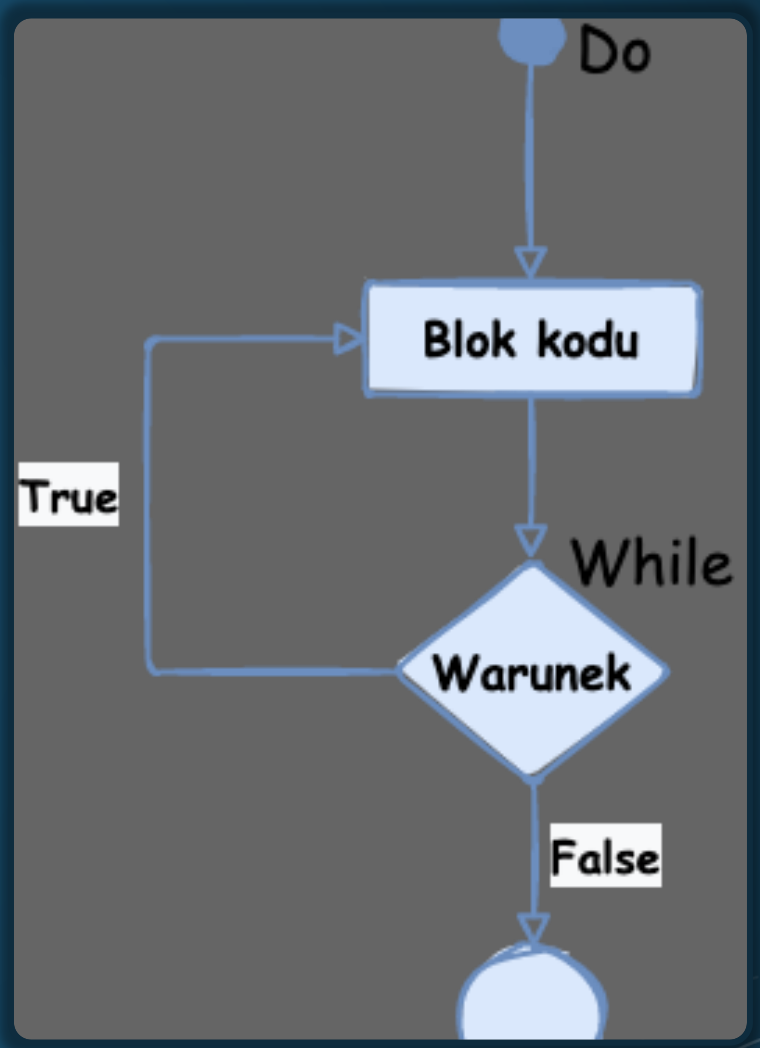
Pętlę while najczęściej wykorzystuję się, gdy nie można określić konkretnej ilości powtórzeń do wykonania. Znane są nam jednak warunki określające, jak długo dana sytuacja będzie zachodziła. Przykładowo: „dopóki licznik nie przekroczy zakresu tablicy, wypisz jej kolejny element”.

Pętla jest wykonywana tak długo, póki warunek jest spełniony. Jeżeli warunek od początku nie będzie spełniony, taka pętla nie zostanie wywołana ani razu.



PĘTLA DO WHILE (ANG. *DO WHILE LOOP*)

Pętla do while jest to modyfikacja pętli *while*. Głowna różnica polega na tym, że ciało pętli zostanie wykonane przynajmniej raz, nawet jeżeli warunek zawsze jest niespełniony. Najpierw wykonywane są instrukcje zdefiniowane wewnątrz pętli, a dopiero potem sprawdzany jest jej warunek.



PĘTLA FOR (ANG. *FOOR LOOP*)

Pętla for zazwyczaj wykorzystywana jest, gdy z góry znamy ilość wykonania kolejnych iteracji pętli. Jej schematyczny zapis wygląda w ten sposób:

```
for([inicjalizacja]; [warunek]; [modyfikacja]){
    [ciało pętli – instrukcje do wykonania] }
```

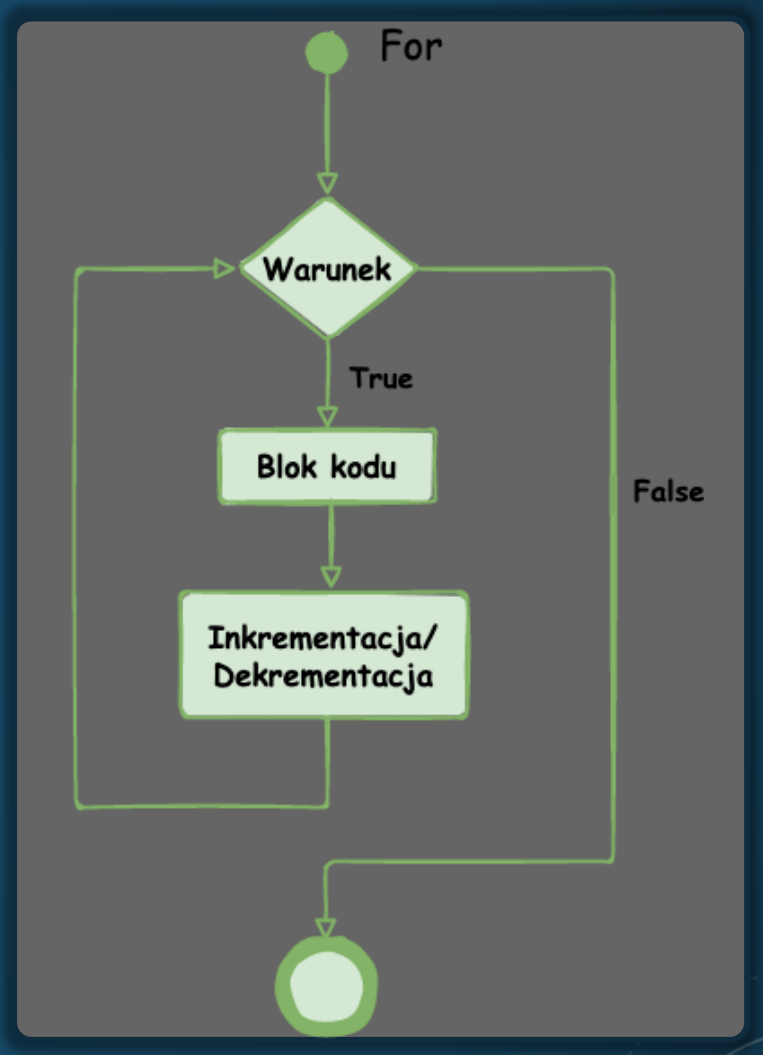
Wszystkie cztery elementy pętli **są opcjonalne**, jednak dla wygody i czytelności kodu warto z nich korzystać.

1.inicjalizacja – służy do zainicjowania zmiennych początkowymi wartościami, np. utworzenie i zainicjowanie licznika;

2.warunek – w warunku należy sprawdzić, czy ciało pętli ma być wykonane;

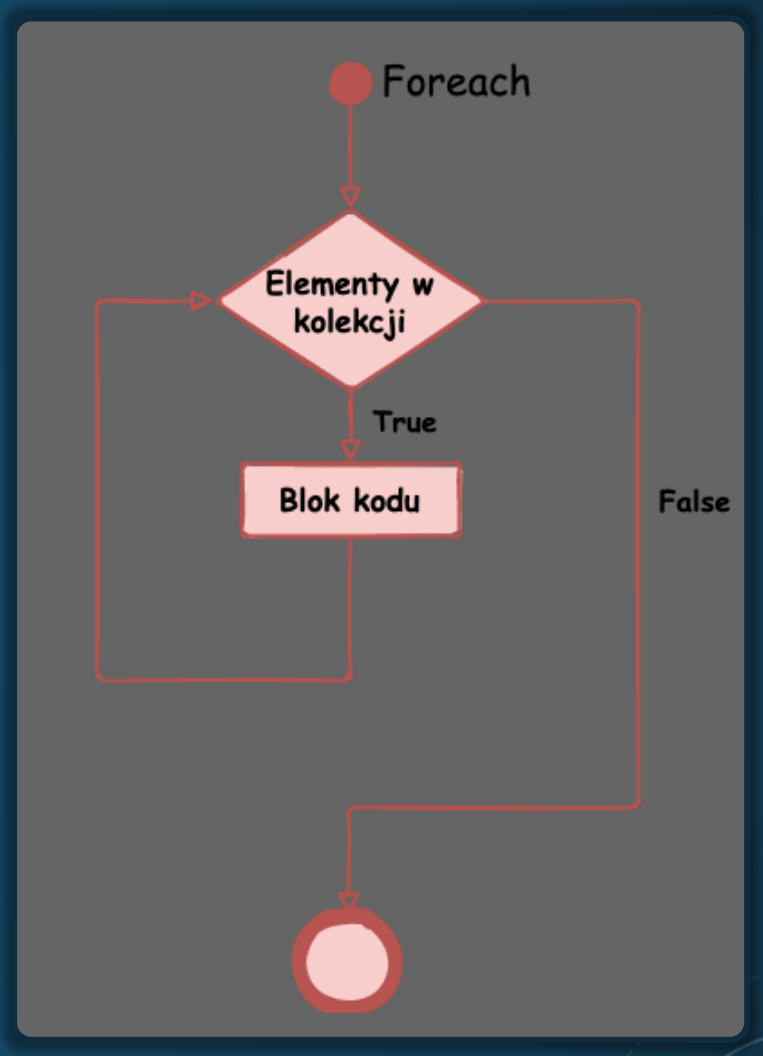
3.modyfikacja – ten element pętli jest wykonywany po ciele funkcji, można tu np. zmodyfikować licznik pętli;

4.ciało pętli – czyli wszystkie instrukcje, które chcemy, żeby były wykonane określoną ilość razy



PĘTLA FOREACH (ANG. *FOREACH LOOP*)

Pętla for each to konstrukcja, która pozwala na sekwencyjne przeglądanie różnych zbiorów danych. Mogą nimi być tablice, a także dynamiczne struktury jak na przykład listy.



PRZYKŁAD

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars)
  System.out.println(i);
```

CIEKAWE SKRÓTY

- CTRL + ALT + L - formatowanie kodu
- CTRL + ALT + T – tworzenie instrukcji
- CTRL + ALT + F7 – pokazuje użycie danego elementu
- CTRL + ALT + SHIFT + T – refractoring (zmiany nazw)

- CTRL + SHIFT + V – otwarcie schowka kopiowania
- CTRL + SHIFT + ENTER – uzupełnianie instrukcji
- CTRL + SHIFT + SPACJA – otwiera klasę danego obiektu

- CTRL + SHIFT + Strzałka w dół lub górę – przenoszenie wiersza w górę lub w dół