

# 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. Instrukcja wczytywania danych
6. [Instrukcje w Javie – Instrukcja warunkowa](#)
7. [Instrukcje w Javie – Instrukcje iteracyjne](#)
  - [pętla while](#)
  - [pętla do while](#)
  - Pętla for
  - Pętla foreach
8. Instrukcje w Javie – Instrukcja wyboru
9. [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

The background is a solid dark blue. On the left side, there are several concentric circular patterns. One large circle has a scale around its perimeter with numbers from 140 to 260 in increments of 10. There are also smaller circles and arcs, some with arrows indicating a clockwise direction. The overall design is technical and geometric.

# HISTORIA JAVY

# Java Historia



## Historia Java



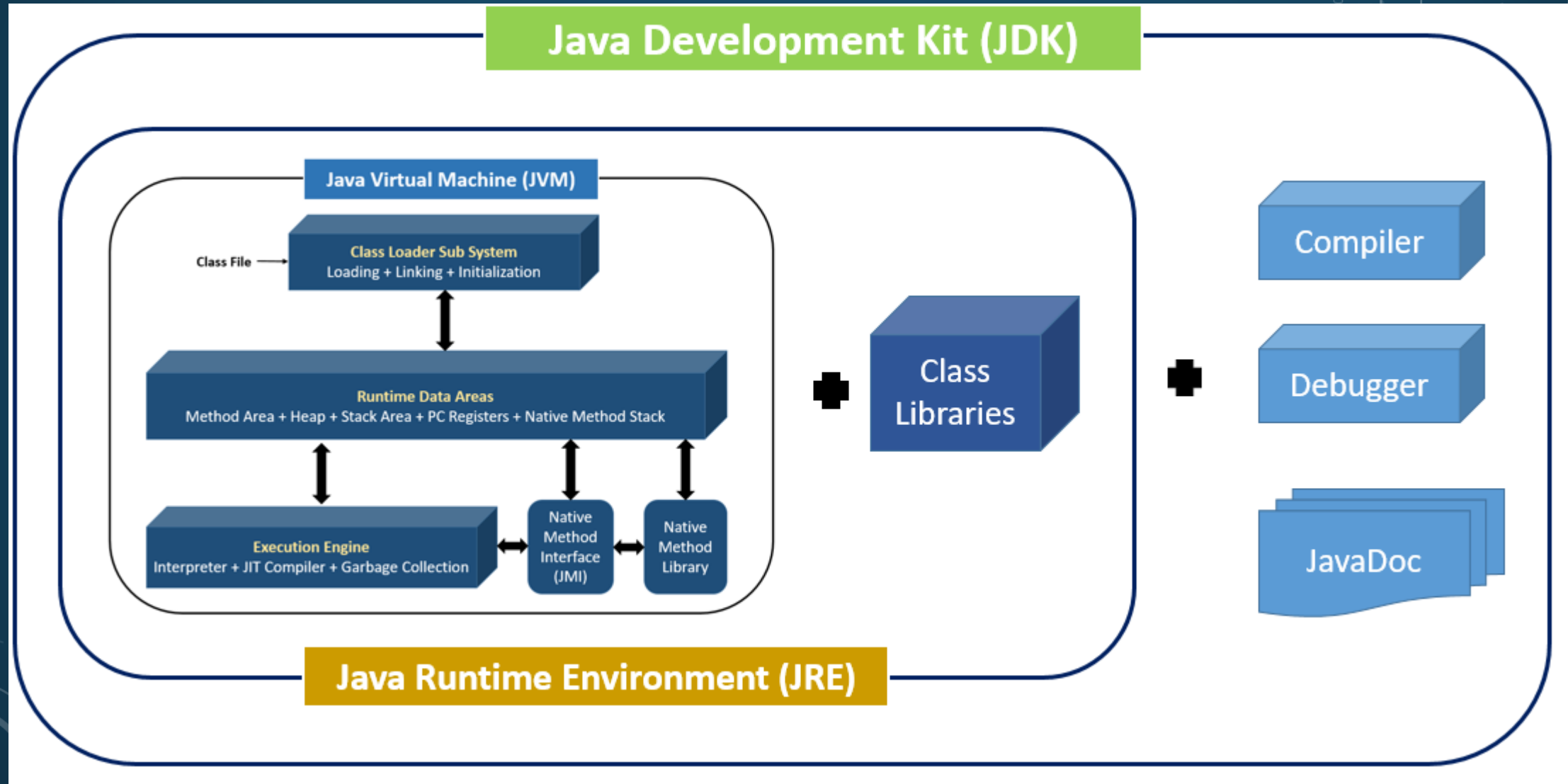
Wersja Javy	Nazwa Kodowa	Liczba class
Java 1.0	OAK	212
Java 1.1	----	504
Java 1.2	Playground	1520
Java 1.3	Kestrel	1842
Java 1.4.0	Merlin	2991
Java 1.5.0	Tiger	3279
Java 1.6.0	Mustang	3793
Java 1.7.0	Dolphin	4024
Java 1.8.0	Spider	4240
Java 1.9.0		6005
Java 1.10.0		6002
Java 1.11.0		4411
Java 1.12.0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

WERSJE JAVY

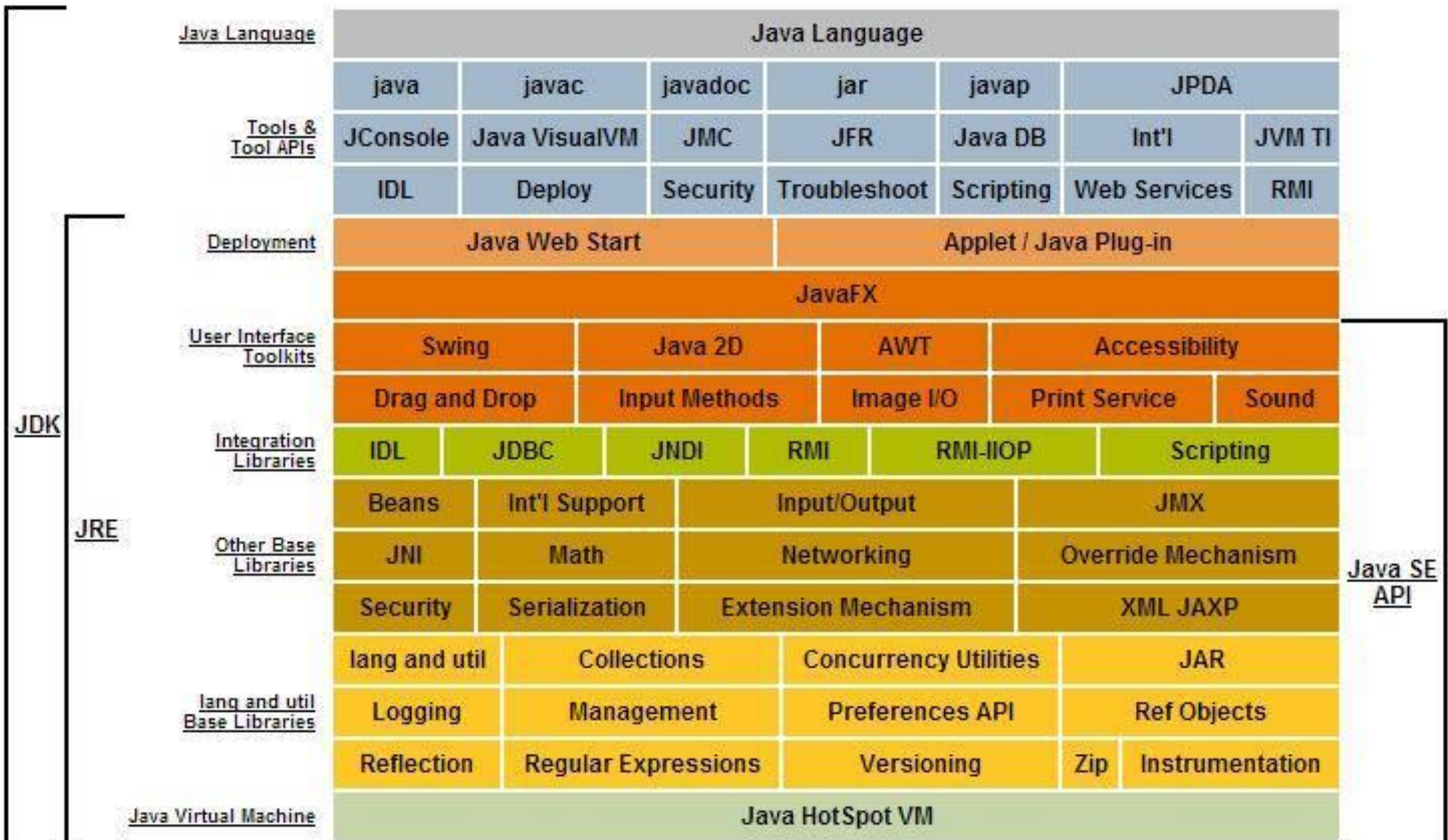
# Przegląd wersji Java

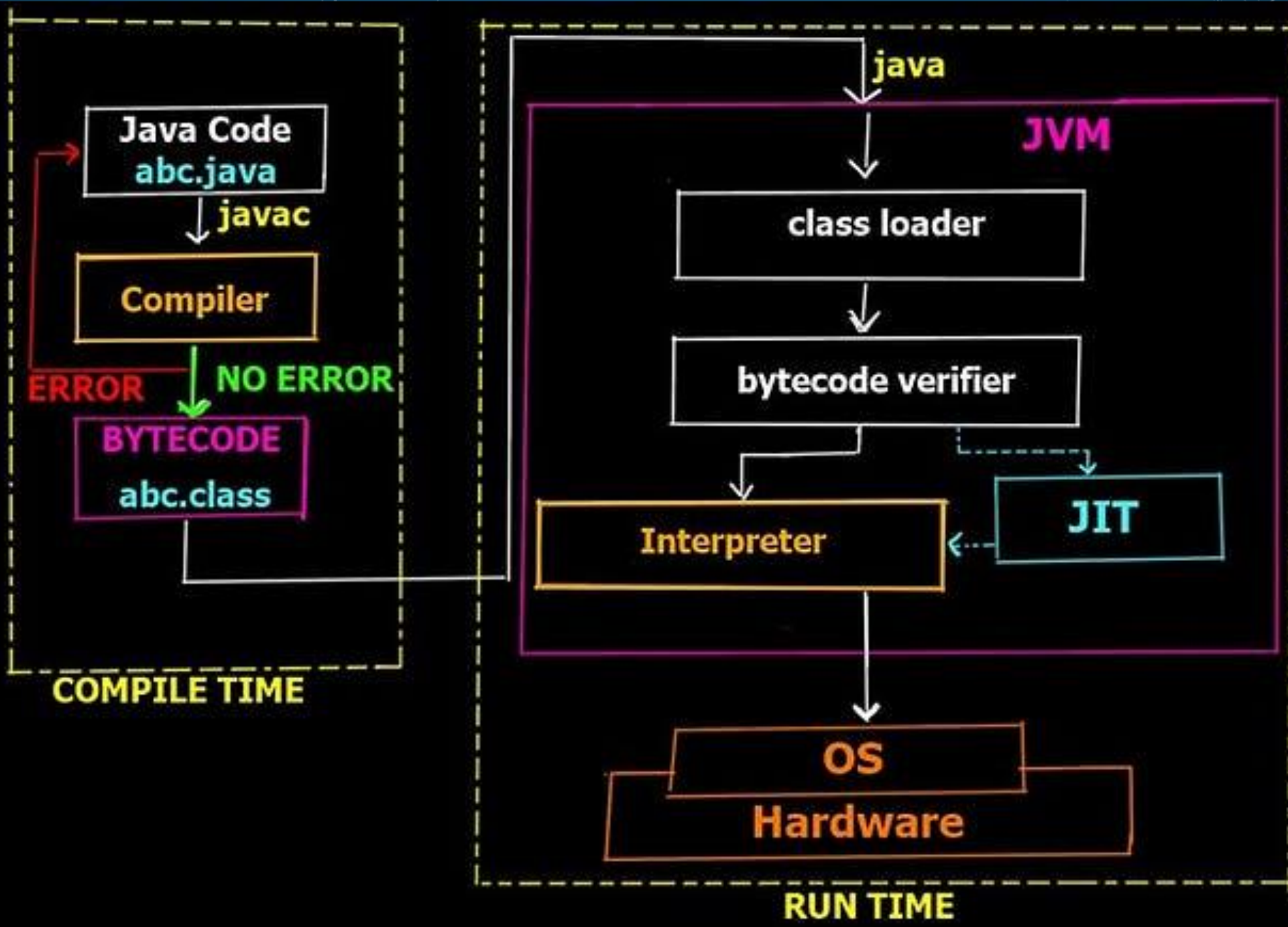


# JDK, JRE | JVM

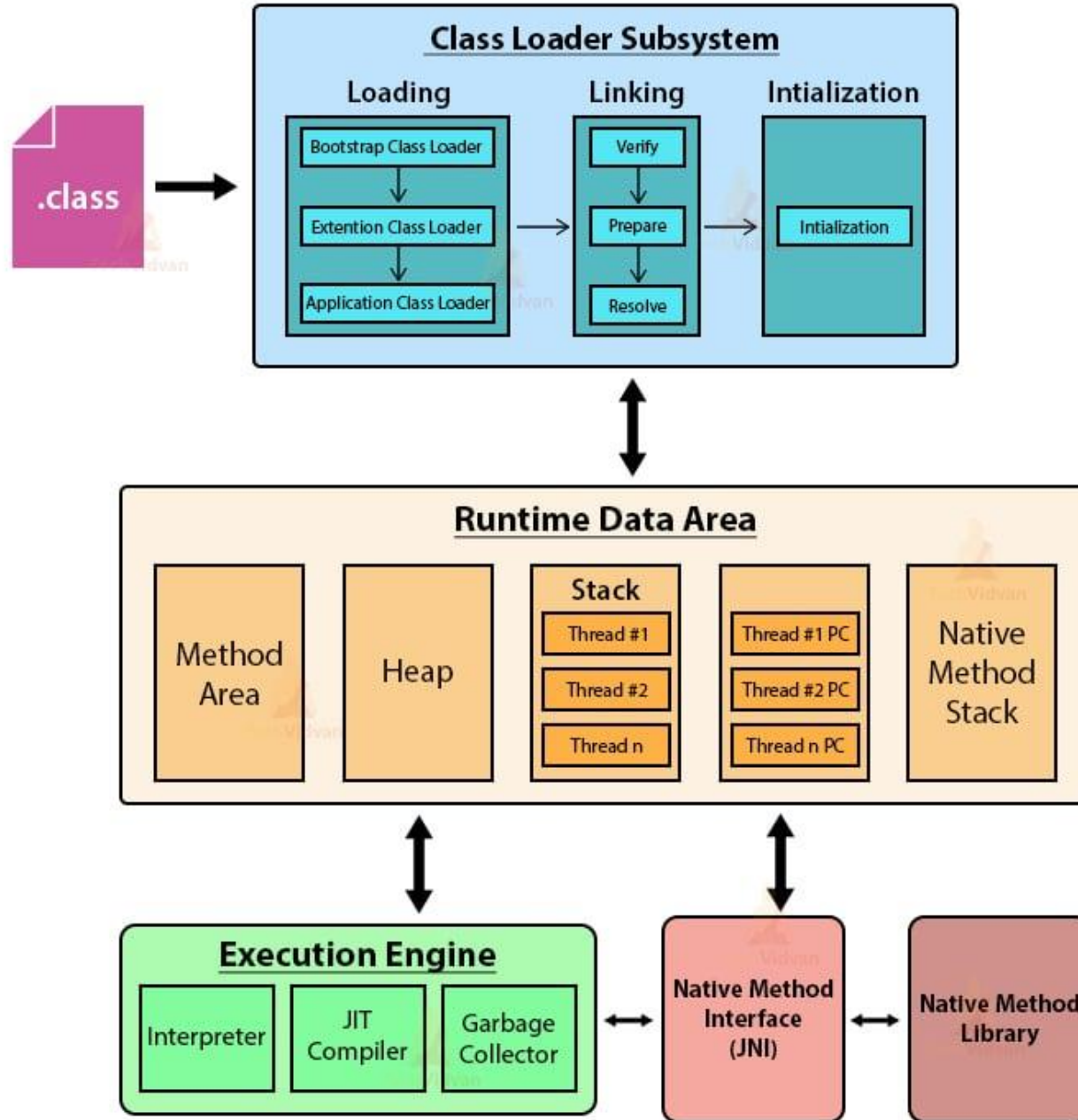








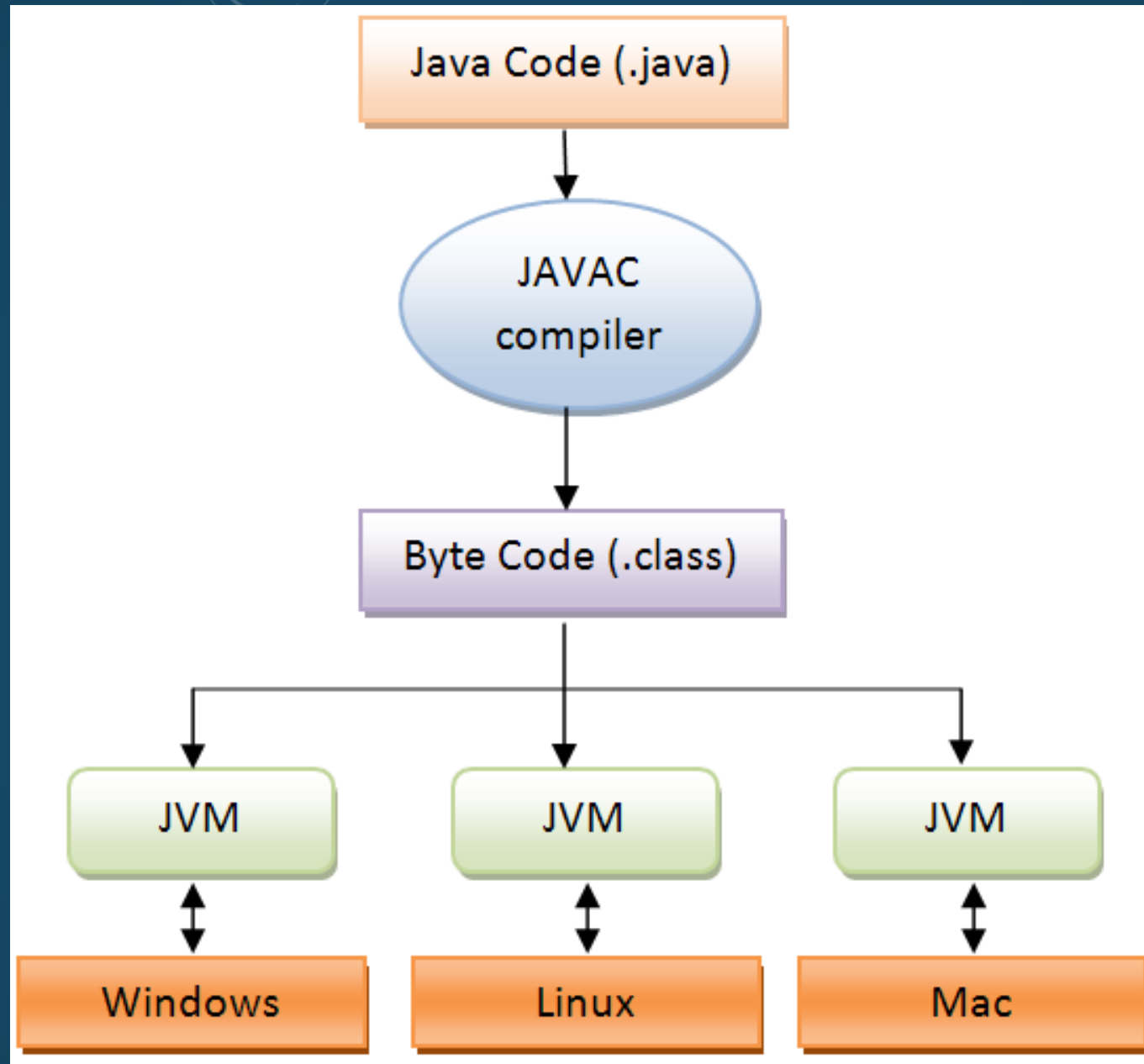
# JVM Model



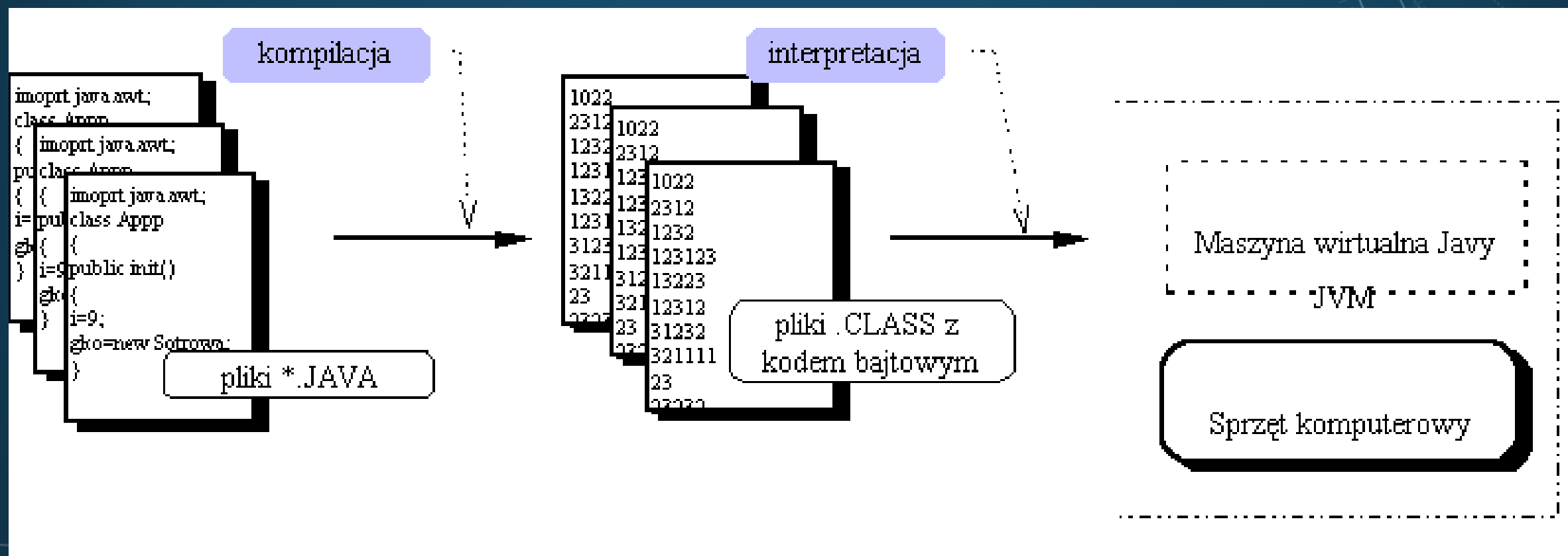


The background is a dark blue gradient. On the left side, there is a large, semi-circular scale with tick marks and numbers ranging from 140 to 260. Several concentric circles and arcs are scattered across the image, some with arrows indicating a clockwise or counter-clockwise direction. The overall aesthetic is technical and modern.

# TWORZENIE PROGRAMU W JĘZYKU JAVA



# KOMPILACJA I WYKONANIE PROGRAMU NAPISANEGO W JAVIE



# KOMPILACJA I WYKONANIE PROGRAMU NAPISANEGO W C++.





The background is a dark blue gradient. On the left side, there is a large, semi-circular scale with tick marks and numbers ranging from 140 to 260. Several concentric circles and arcs are scattered across the image, some with arrows indicating a clockwise or counter-clockwise direction. The overall aesthetic is technical and scientific.

# 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:

- DEKLARACJA 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*



The background is a solid dark blue. On the left side, there are several concentric circular patterns. One prominent circle has a scale around its perimeter with numbers ranging from 140 to 260 in increments of 10. Other circles are partially visible, some with dashed lines and arrows indicating a clockwise direction. The overall aesthetic is technical and modern.

# 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

The background is a solid dark blue. It features several abstract, light blue circular elements. On the left side, there is a large, semi-circular scale with tick marks and numbers ranging from 140 to 260. Overlaid on this and other parts of the background are various concentric circles, some solid and some dashed, with small arrowheads indicating a clockwise direction of movement. The overall aesthetic is technical and modern.

# INSTRUKCJE JĘZYKA JAVA

The background is a solid dark blue. It features several faint, light blue circular and semi-circular patterns. On the left side, there is a large circular scale with tick marks and numbers ranging from 140 to 260. Other smaller circular patterns with arrows are scattered across the left and top portions of the image.

# INSTRUKCJA WCZYTYWANIA DANYCH



Jednym ze sposobów wczytania danych do programu w Javie jest wykorzystanie obiektów klasy Scanner.

Procedura postępowania:

1. Utworzyć obiekt klasy Scanner

```
Scanner c = new Scanner(System.in);
```

2. Zaimportować klasę Scanner na początku pliku

```
import java.util.Scanner;
```

3. Utworzyć zmienną odpowiedniego typu do wczytania danych:

Klasa `scanner` oferuje szereg metod, które pozwalają na odczyt danych różnych typów.

- `next()` – odczytuje pojedynczy wyraz
- `nextLine()` – odczytuje całe zdanie
- `nextInt()` - odczytuje kolejną liczbę całkowitą
- `nextDouble()` - odczytuje kolejną liczbę zmiennoprzecinkową,
- `nextBoolean()` - odczytuje kolejną wartość typu boolean,
- `nextBigInteger()` – odczytuje duże liczby całkowite

Oraz analogiczne metody dla typów `byte`, `short`, `long`, `float`, `BigDecimal`.

**Nie istnieje metoda do wczytywania pojedynczego znaku typu `char`. !!!**

Więcej informacji o klasie `Scanner` [tutaj](#).

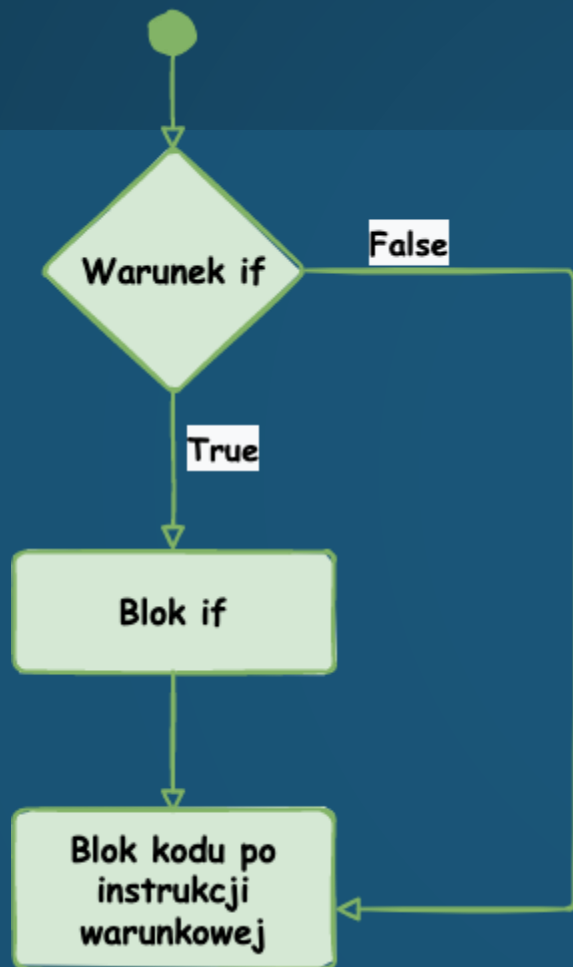
The background is a solid dark blue. It features several faint, light blue circular and semi-circular patterns. On the left side, there is a large circular scale with tick marks and numbers ranging from 140 to 260. Other smaller circular patterns with arrows are scattered across the left and top areas. The text 'INSTRUKCJA WARUNKOWA' is centered in the lower half of the image.

# 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'.*

Wyróżniamy 2 typy instrukcji if:



The background is a solid dark blue. It features several abstract, light blue circular and semi-circular patterns. On the left side, there is a large, curved scale with numerical markings ranging from 140 to 260 in increments of 10. The numbers are oriented along the curve of the scale. Various circular arrows, some solid and some dashed, are scattered across the image, suggesting a sense of rotation or a cyclical process. The overall aesthetic is clean, modern, and technical.

# 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.**

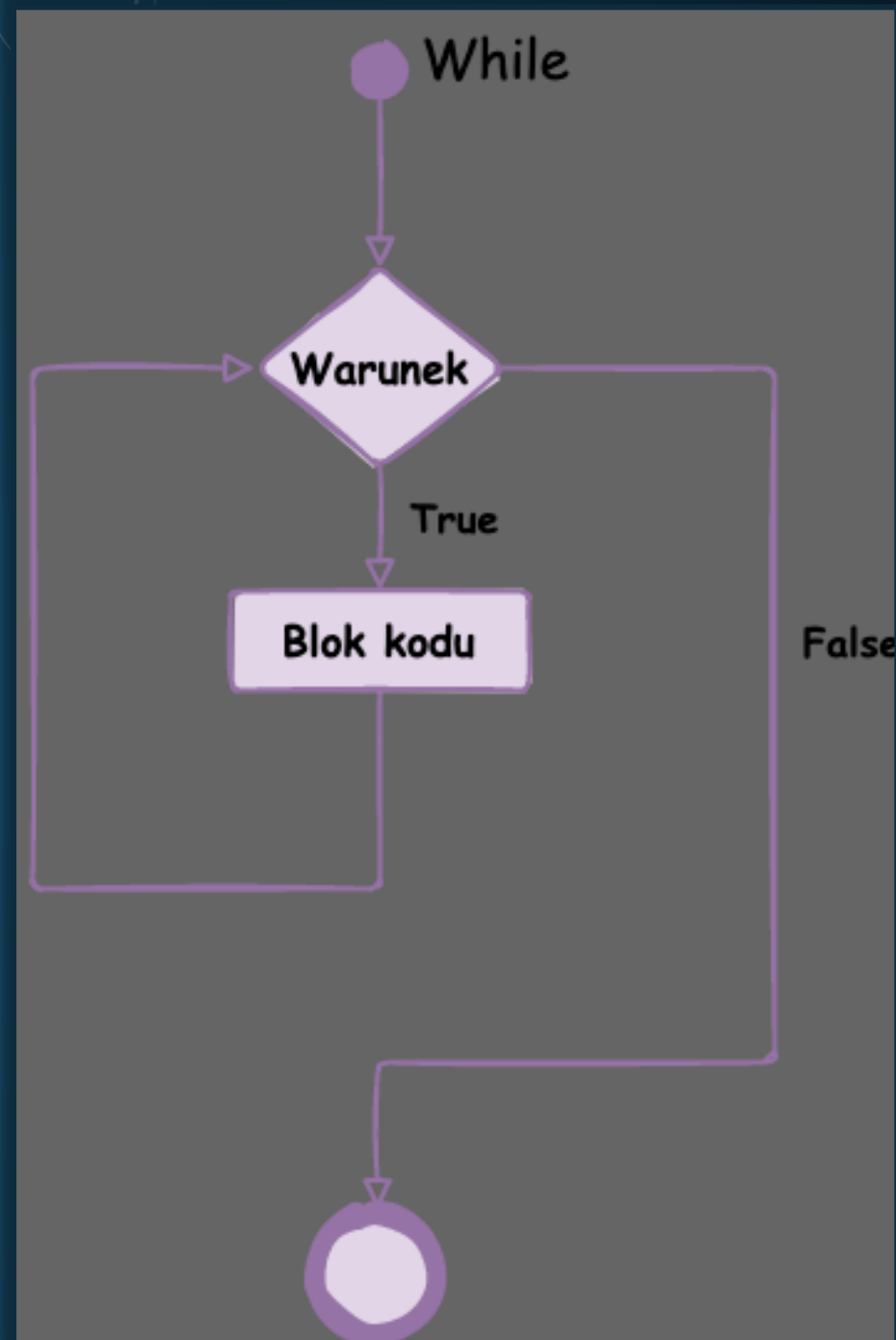
Funkcjonalność wszystkich jest wymienna, a decyzja, którą należy użyć w danym momencie, zależy głównie od kontekstu, w jakim ma być wykorzystana oraz od preferencji programisty.



# JAVA – PĘTLA WHILE (ANG. WHILE LOOP)

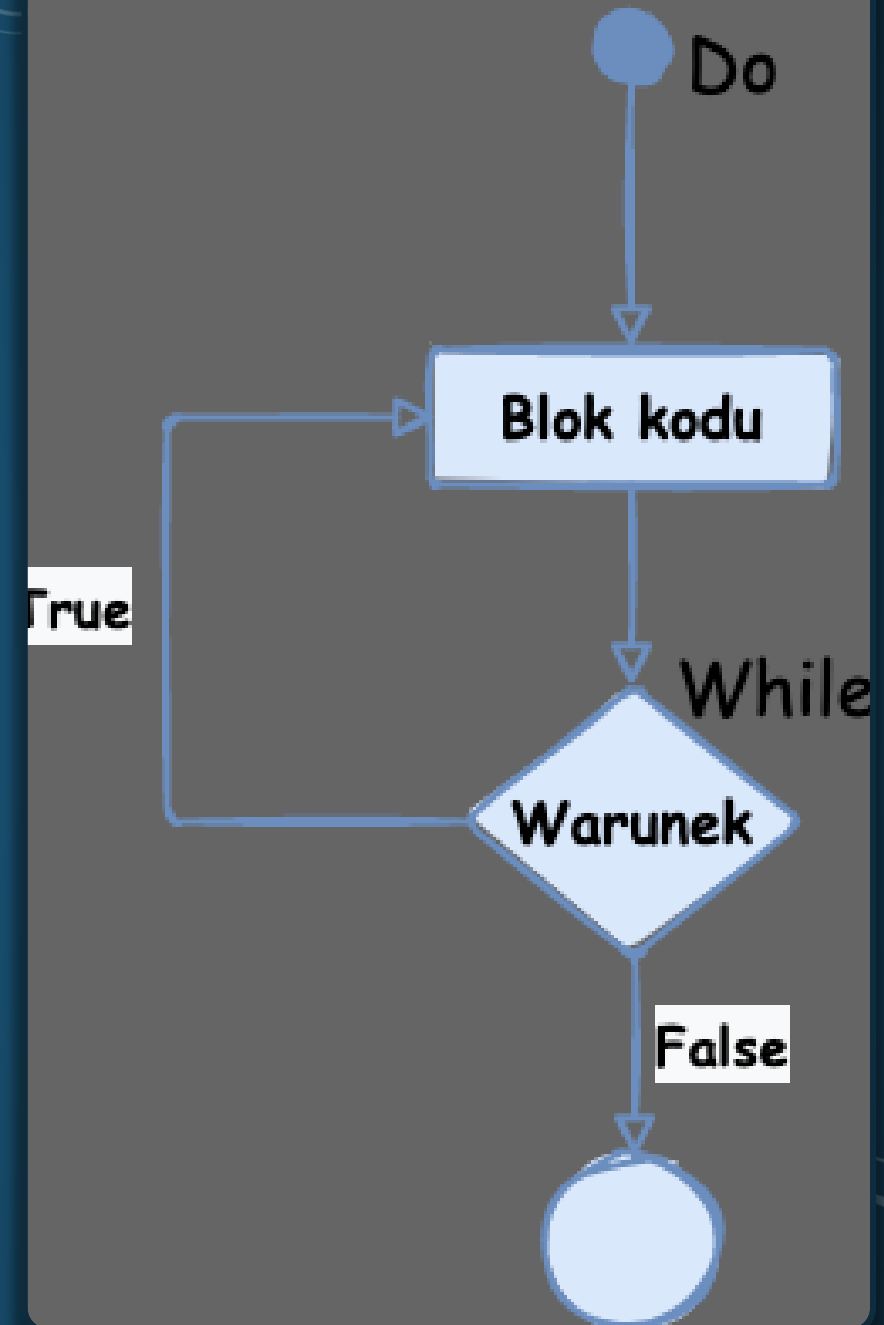
**Pętlę while** najczęściej wykorzystuję się, gdy nie można określić konkretnej ilości powtórzeń do wykonania. Znane są nam jednak warunki określające, jak długo dana sytuacja będzie zachodziła. Przykładowo: „dopóki licznik nie przekroczy zakresu tablicy, wypisz jej kolejny element”.

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



# JAVA – PĘTLA DO WHILE (ANG. *DO WHILE LOOP*)

**Pętla do while** jest to modyfikacja pętli *while*. Główna 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.



# 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