



# WSTĘP DO PROGRAMOWANIA URZĄDZEŃ MOBILNYCH KOTLIN, JAVA

## WYKŁAD 1

- PODSTAWOWE INFORMACJE
- TREŚCI PROGRAMOWE
- ZASADY ZALICZENIA

Rafał Lewandków

pokój 075

rafal.lewandkow@uwr.edu.pl

rafal.lewandkow2@uwr.edu.pl

Forma zajęć i liczba godzin:

Wykład 15 godz. /Laboratorium 30 godz.

Literatura obowiązkowa i zalecana:

- Java. Efektywne programowanie. Joshua Bloch
- Atomic Kotlin. Bruce Eckel

Nakład pracy studenta:

Praca własna studenta: 30 godz.

Łączna liczba godzin 75

Liczba punktów ECTS: 3

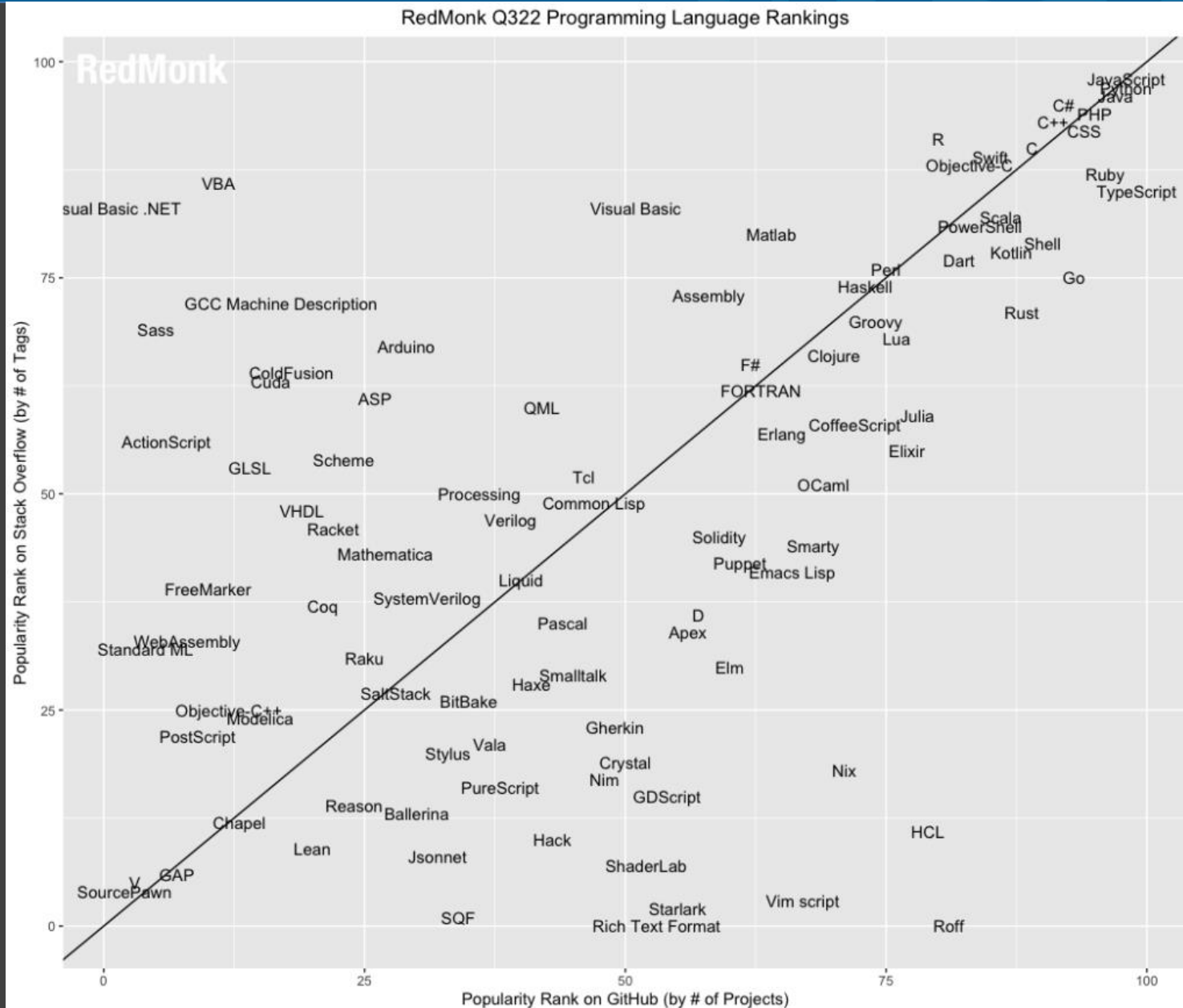
Rafał Lewandków  
pokój 075  
[rafal.lewandkow@uwr.edu.pl](mailto:rafal.lewandkow@uwr.edu.pl)  
[rafal.lewandkow2@uwr.edu.pl](mailto:rafal.lewandkow2@uwr.edu.pl)

Konsultacje:

# ZASADY ZALICZENIA

Warunkiem zaliczenia laboratorium jest uzyskanie pozytywnej oceny z wykonywanych zadań umieszczanych na listach

- Na zajęcia przewidzianych jest 8 list zadań
- Z każdej listy wystawiana jest osobna ocena
- Nie jest konieczne zaliczenie wszystkich list aby otrzymać ocenę pozytywną z laboratorium
- Każde zadanie ma przydzieloną liczbę punktów
- Każda lista posiada informację o liczbie punktów wymaganych na konkretną ocenę
- Każda lista posiada termin zwrotu
- Za każdy tydzień opóźnienia otrzymana ocena jest obniżana o 0,5
- Każdą listę można poprawić w ciągu 4 tygodni od terminu zwrotu listy - za wyjątkiem końca semestru
- Poprawa list jest możliwa pod warunkiem uzyskania co najmniej 50% punktów
- Ocena końcowa jest średnią arytmetyczną ze wszystkich ocen z list.
- Na ocenę 3,0 wymagana jest średnia co najmniej 3,0



1. Porównanie języków Java i Kotlin
2. Typy danych
3. Wyrażenia, instrukcje, pętle
4. Funkcje
5. Klasy, obiekty
6. Interfejsy
7. Wielowątkowość
8. Wzorce projektowe

 **IntelliJ IDEA**

Capable and Ergonomic

Download

IDE

Java

Groovy

Scala

Kotlin

Android

## Why IntelliJ IDEA

## Kotlin (Introduced 2011, Version 1.0: 2016)

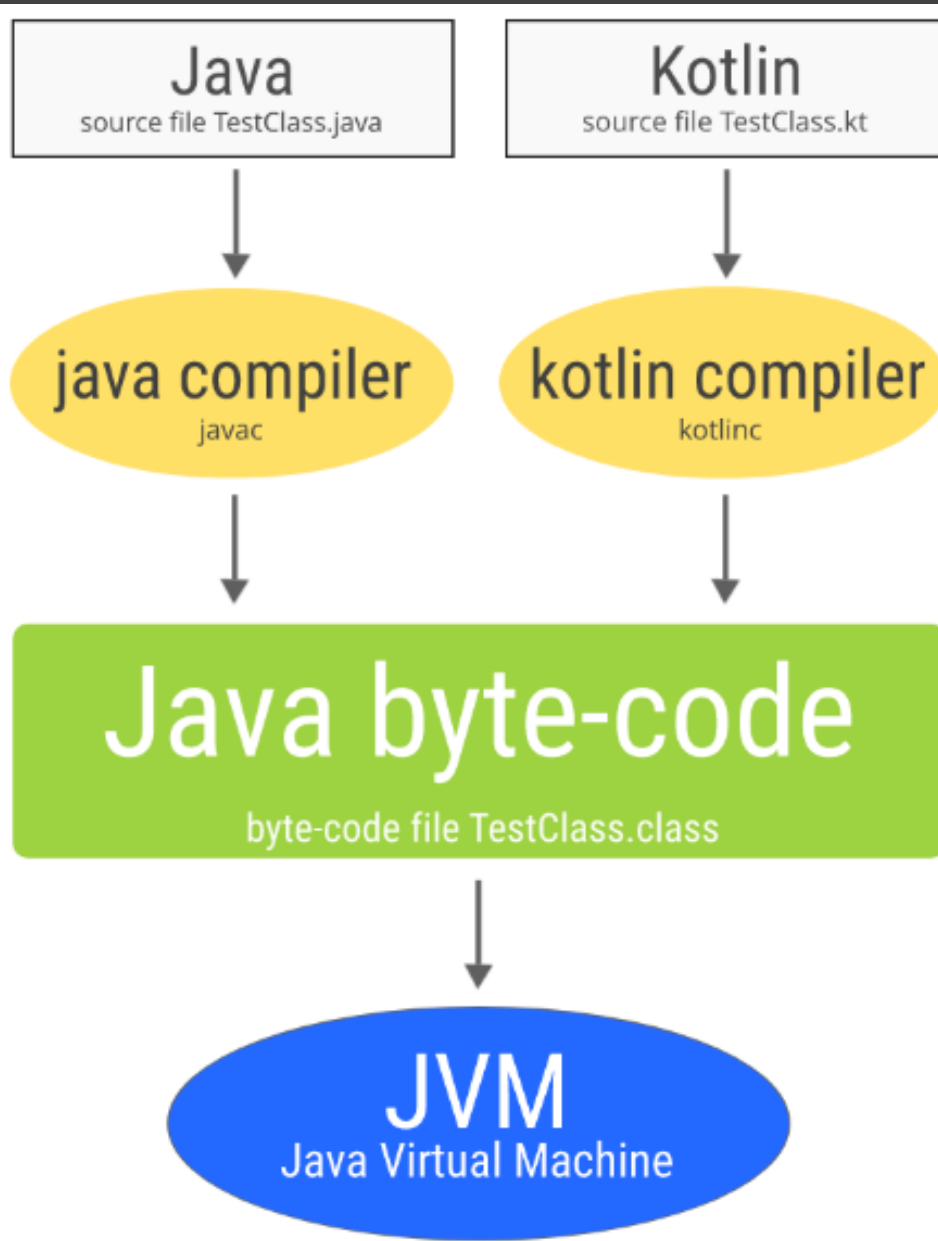
- **FORTTRAN: FORMula TRANslation (1957)**
- **LISP: LISt Processor (1958)**
- **ALGOL: ALGORithmic Language (1958)**
- **COBOL: COMmon Business-Oriented Language (1959)**
- **BASIC: Beginners' All-purpose Symbolic Instruction Code (1964)**
- **Simula 67, the Original Object-Oriented Language (1967)**
- **Pascal (1970)**
- **C (1972)**
- **Smalltalk (1972)**
- **C++: A Better C with Objects (1983)**
- **Python: (1990)**
- **Haskell: Pure Functional Programming (1990)**
- **Java: Virtual Machines and Garbage Collection (1995)**
- **JavaScript: (1995)**
- **C#: (2000)**
- **Scala: (2003)**
- **Groovy: (2007)**



**JDK (Java Development Kit)** - Pakiet Programisty Javy. JDK zawiera Środowisko Uruchomieniowe Javy (tzn. JRE) oraz zestaw narzędzi niezbędnych do wytwarzania oraz kompilowania oprogramowania tworzonego w języku JAVA.

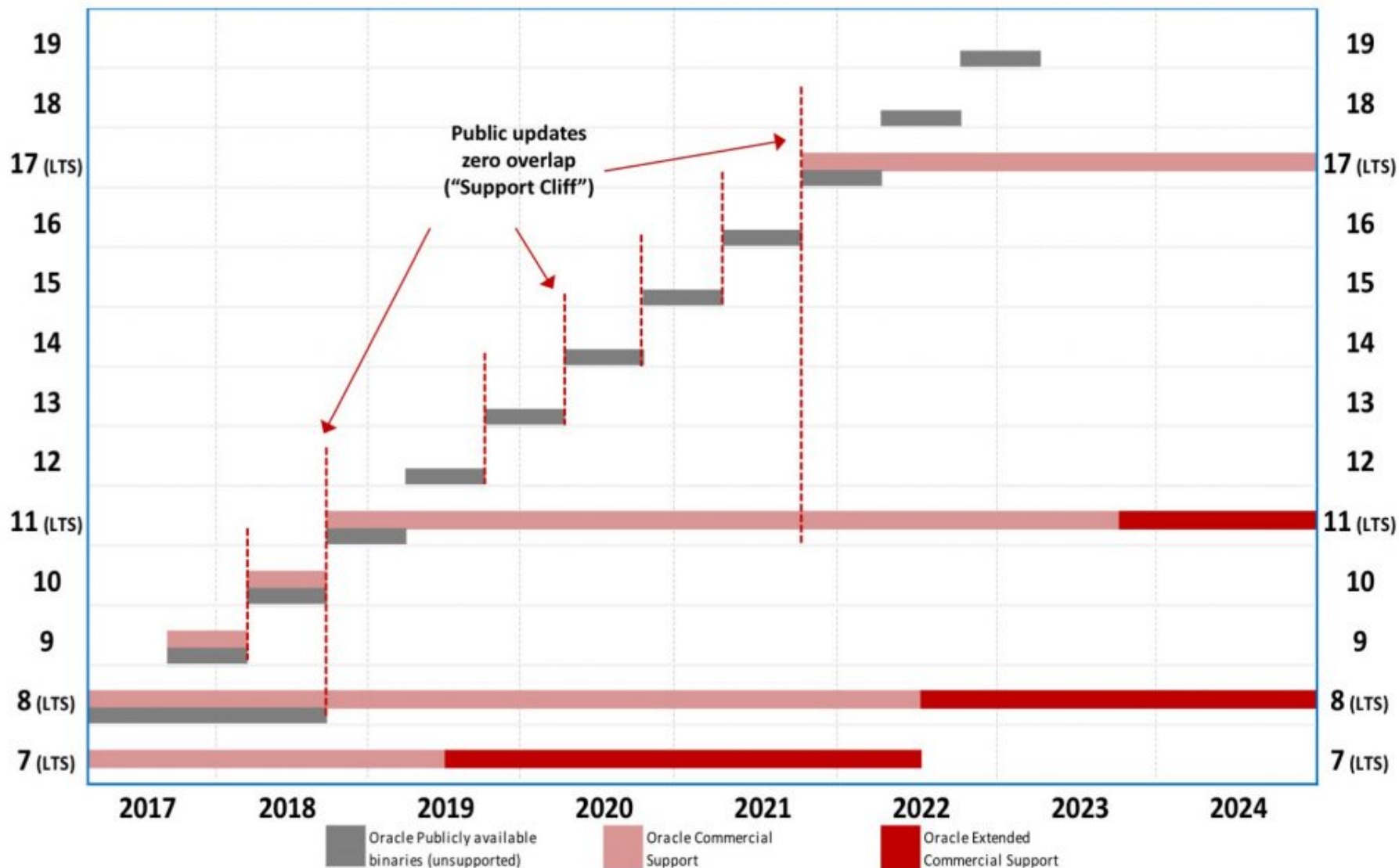
**JRE (Java Runtime Environment)** - Środowisko Uruchomieniowe Javy. W skład JRE wchodzi Wirtualna Maszyna Javy (JVM) + zbiór klas oraz narzędzi wymaganych do uruchomienia aplikacji wytworzonych w języku JAVA.

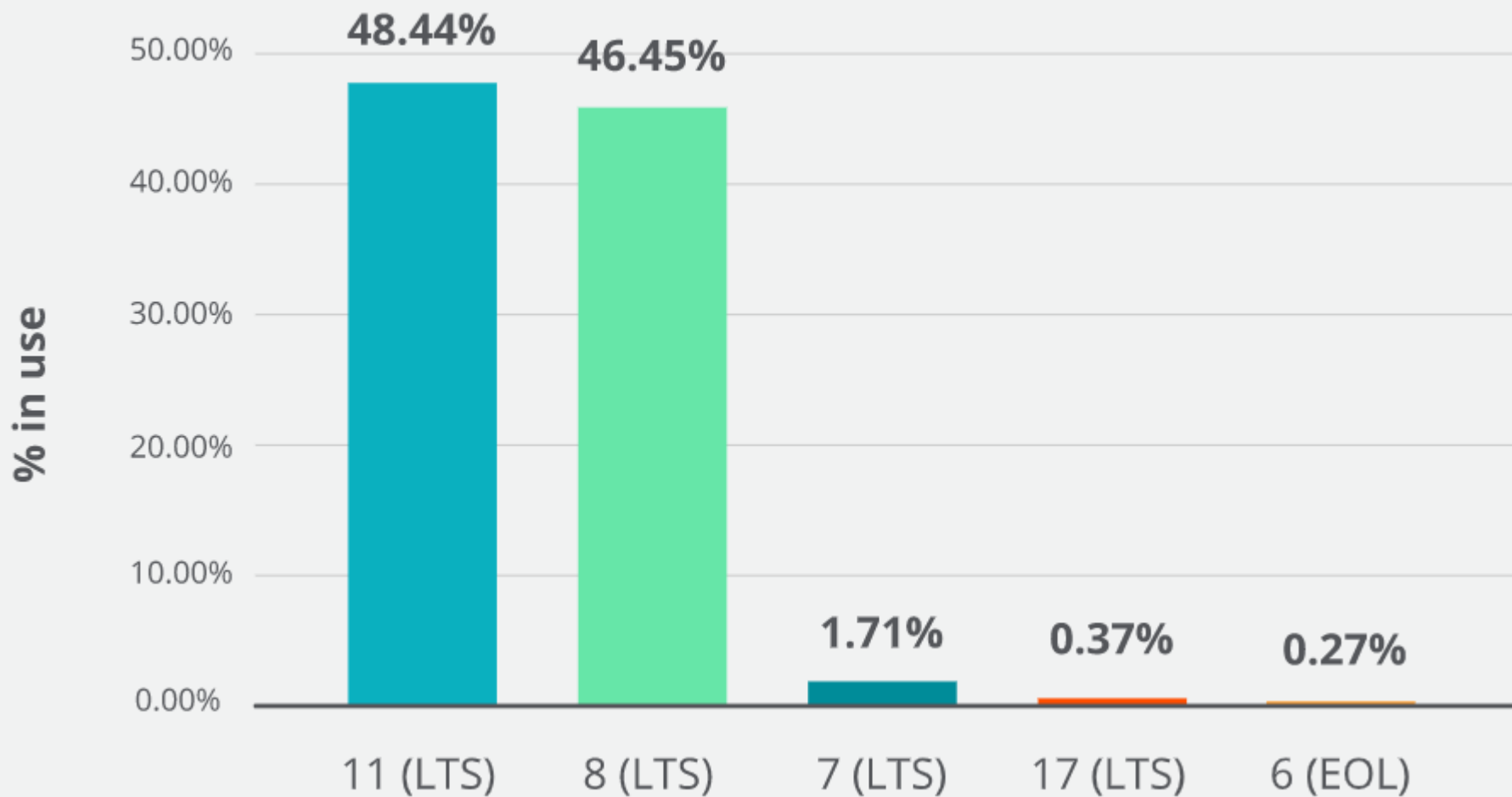
**JVM (Java Virtual Machine)** - Wirtualna Maszyna Javy. Środowisko zdolne do wykonywania skompilowanego kodu aplikacji (kod bajtowy Javy).



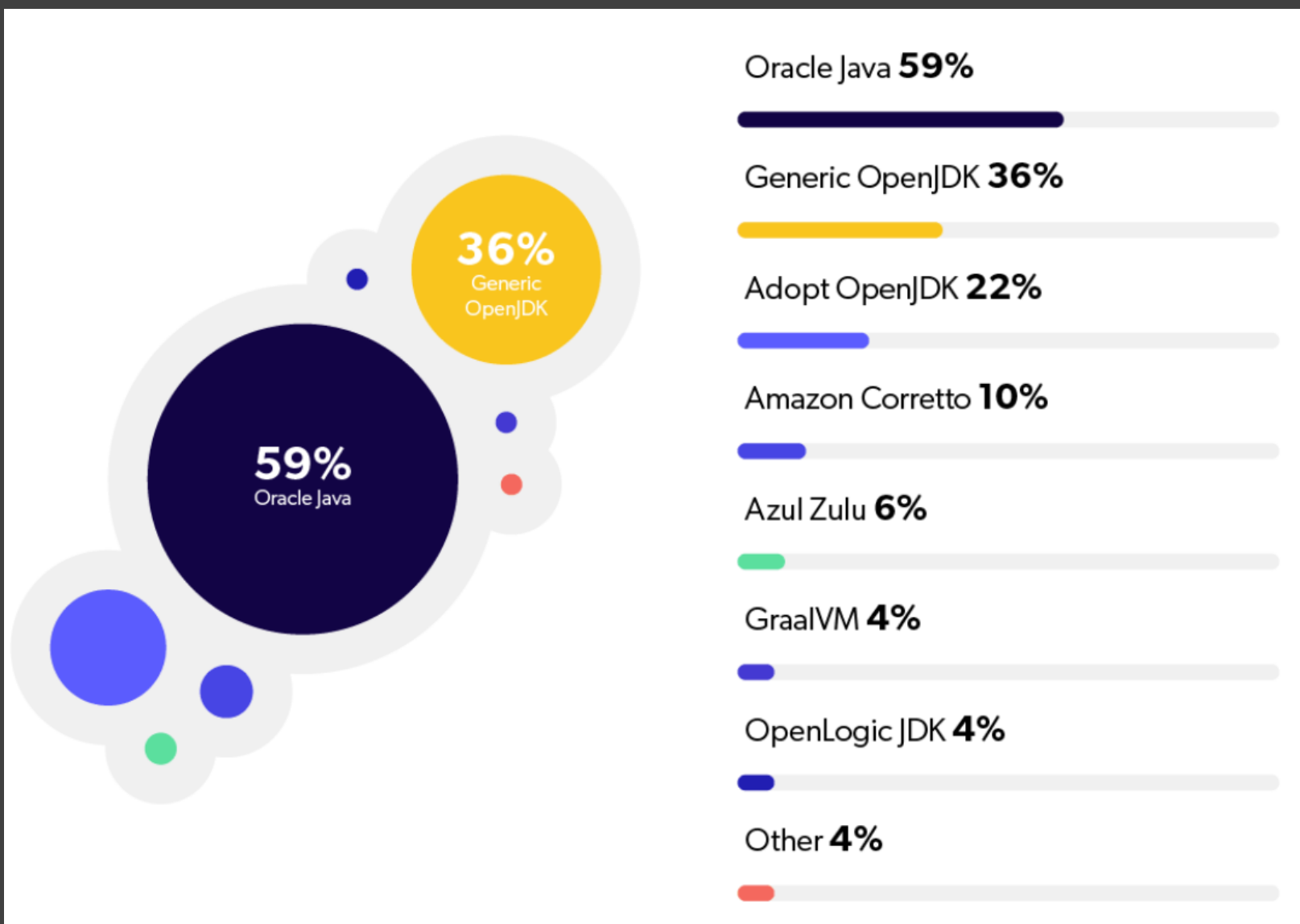
## Java SE Lifecycle – 5+ Year Timeline

Java SE Version





# Java – JRE/JDK Distributions



- Niektóre podobieństwa do C/C++
  - Styl pisania komentarzy
  - Wiele słów kluczowych jest identycznych
  - Typy danych
  - Niemal identyczna struktura kodu
- Niektóre różnice
  - Java wprowadza wiele słów kluczowych nieznanych z C/C++
  - Występują operatory nieznane w C/C++
  - Nie występuje przeciążanie operatorów
  - Brak niektórych cech języka np. wskaźniki
  - Niektóre cechy języka są zmodyfikowane w porównaniu do C/C++ - pętle muszą być kontrolowane przez wyrażenia logiczne

- Cechy języka
  - Statyczne typowanie danych – na etapie kompilacji typy wyrażeń są znane a kompilator sprawdza czy istnieją pola i metody w obiektach do których odwołujemy się w kodzie
  - Domniemanie typów – typ danych jest określany przez kompilator na podstawie kontekstu
  - Typy zerowalne
  - Typy funkcyjne
  - Klasy danych
  - Można łączyć programowanie obiektowe i funkcyjne

- Null Safety

`val number: Int? = null`

- Extension Functions
- Coroutines Support
- Data Classes
- Functional Programming: Higher-Order Functions
- Primitive types
- Wildcard Types

`ArrayList<?>`



```
plugins {  
    id 'java'  
}  
  
group 'org.example'  
version '1.0-SNAPSHOT'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.2'  
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.7.2'  
}  
  
test {  
    useJUnitPlatform()  
}
```



× **m** pom.xml

```
<modelVersion>4.0.0</modelVersion>
<groupId>groupId</groupId>
<artifactId>JavaMavenDemo</artifactId>
<version>1.0-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifestFile>src/main/resources/META-INF/MANIFEST.MF</manifestFile>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```