JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 16 (2020) 219–226
Received: 25 June 2020
Accepted: 15 July 2020

Java and Kotlin code performance in selected web frameworks

Wydajność kodu Java i Kotlin w wybranych szkieletach aplikacji internetowych

Grzegorz Bujnowski*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract

This paper discusses the issue of comparing Java and Kotlin technologies based on the web application framework. The criteria taken into account for testing purposes are: execution time, memory usage, CPU load, database response in set time. Series of tests and their in-depth comparative analysis are carried out. For this case, tests and code analysis were carried out to draw comparative conclusions. The performance in terms of web frameworks, database response speed and tests implementation in different languages - in all these Kotlin proved to be less efficient. There is no significant difference for CPU load. Between individual measurements, the difference does not exceed 2%. Implementation in the Kotlin language has never achieved the best result in any group of measurements.

Keywords: kotlin; jvm; java; benchmark

Streszczenie

W tym artykule omówiono kwestię porównania technologii Java i Kotlin w oparciu o szkielet aplikacji internetowych. Kryteria brane pod uwagę dla celów testowych to: czas wykonania, wykorzystanie pamięci, obciążenie procesora, liczba odpowiedzi z bazy danych w zadanym czasie. Przeprowadzana jest seria testów i ich dogłębna analiza porównawcza. Przeprowadzono testy i analizę kodu. Wydajność pod względem szkieletów aplikacji internetowych, szybkości odpowiedzi bazy danych i szybkości działania testów - we wszystkich Kotlin okazał się mniej wydajny. Nie ma znaczącej różnicy dla obciążenia procesora. Pomiędzy poszczególnymi pomiarami, różnica nie przekracza 2%. Implementacja w języku Kotlin nigdy nie osiągnęła najlepszego wyniku w żadnej grupie pomiarów.

Słowa kluczowe: kotlin; wirtualna maszyna javy; testy wydajnościowe

1. Wstęp

Poniższa praca porównuje technologię Java oraz Kotlin na podstawie szkieletów aplikacji internetowych. Uwzględnione w badaniach kryteria to: wydajność rozwiązywania problemów, zużycie pamięci oraz obciążenie procesora, szybkość działania na bazach danych.

Spośród ogromnej liczby powstałych języków programowania, Java wyróżniła się wyraźnie, stając się przez kilka dziesięcioleci jedną z preferowanych technologii przez programistów. Po ponad 25 latach od powstania Javy, w tle pojawił się nowy konkurent - Kotlin. Nowy oficjalny język programowania promowany przez firmę Google od 2017 roku. Jest kolejnym z wielu języków korzystających z Wirtualnej Maszyny Java. Według indeksu TIOBE [10] w marcu 2020 roku technologia ta zajmowała 30 miejsce, w rankingu popularności.

Od oficjalnego wprowadzenia ćwierć wieku temu obiektowego języka programowania, nadal Java jest najpopularniejszym językiem programowania w 2020 roku. Jest to język programowania początkowo opracowany przez Sun Microsystems, a obecnie będący własnością Oracle Corporation. Jednym z głównych powodów sukcesu Javy jest niezależność platformy – kod jest kompilowany do kodu bajtowego a program wykonywany za pośrednictwem wirtualnej maszyny Java (JVM). Dla wirtualnej maszyny powstało wiele alternatyw dla języka Java, takich jak Groovy, Clojure, Scala i Kotlin. Społeczność programistyczna zwraca szczególną uwagę na Kotlin, ze względu na wsparcie gigantów technologicznych. Kotlin stał się językiem traktowanym priorytetowo przez dominujące na rynku technologicznym.

Zaletą języka Kotlin i Java jest to, że mogą ze sobą współistnieć w zakresie jednego projektu. Kotlin jest kompatybilny z Javą, co oznacza że istnieje możliwość posiadania kodu Kotlin i Java w tym samym projekcie. W związku z tym, że Kotlin jest najszybciej rosnącym w popularności językiem na platformie programistycznej GitHub, jak i to że firma Google oficjalnie zaleca stosowanie tej technologii dla rozwijania aplikacji na telefony Android.

Artykuł próbuje odpowiedzieć na pytania:

1. Czy istnieje różnica w wydajności Java i Kotlin na

*Corresponding author
Email address: gbujnow@gmail.com (G. Bujnowski)

przykładzie szkieletu aplikacji internetowej?

2. W jaki sposób można dokonać porównania wydajności Java i Kotlin na szkielecie aplikacji internetowej?

2. Przegląd publikacji naukowych

Artykuł pod tytułem: "How and Why did developers migrate applications from Java to Kotlin? A study based on code analysis and interviews with developers" (Universite Polytechnique Hauts-de-France) opisuje trzy przypadki:

- gdy programiści zaczynają pisać aplikacje od początku używając języka Kotlin,
- rozwijają kod napisany w Java za pomocą Kotlin,
- migrują całkowicie aplikacje z technologii Java do Kotlin [1].

Za pomocą analizy statycznej kodu, pod obserwację wzięto historię rozwijania projektów. Badanie opiera się na studium 374 aplikacji internetowych i 78 wywiadach z programistami odpowiadającymi za użycie nowego języka programowania. Praca rozpoznaje różnicę pomiędzy dwoma językami programowania. Porównuje języki programowania Java i Kotlin. Praca wskazuje, lepszą jakość kodu w projektach napisanych w języku Kotlin. Podane przez programistów powody do zadowolenia z języka Kotlin to: uniknięcie wyjątków typu null pointer (stos błędu pokazywał wyjątek null w 51,96% projektach opisujących ponad 600 projektów w Javie), Kotlin pozawala wyeliminować ten typ problemów z perspektywy kompilatora.

Ankietowani programiści zdecydowali się na migrację z Java, w związku z bardziej defensywnymi technikami programowania. Badania opinii społecznej pokazały, że zmiany w wyborze technologii dotyczących nowych projektów, wynika głównie z wzrastającego trendu narzuconego przez politykę firmy Google. Trend ten wzmacniany jest również badaniami, które pokazują mniejszą złożoność językową programów napisanych w Kotlin.

Drugi artykuł to „Comparative study Java vs Kotlin", praca strukturalnie i semantycznie porównuje oba języki programowania, określa zalety i wady każdego z nich [2]. Artykuł teoretyczny omawiający podstawy składni i zastosowania. Zwraca uwagę na skróceniu składni
i opisuje nowe funkcjonalności.

3. Cel i przedmiot badań

Celem badania jest porównanie języków Java oraz Kotlin na przykładzie szkieletu aplikacji internetowej.
W artykule, dogłębnej analizie zostanie poddana szybkość rozwiązywania problemów, jak i zużycie pamięci oraz obciążenie procesora, szybkość działania na bazie danych.
Do testów zostaną użyte algorytmy porównujące wydajność pomiędzy różnymi językami programowania.

Zostaną stworzone aplikacje internetowe, rozwiązujące typowe problemy w obu językach. Celem będzie porównanie szybkości obsługiwania bazy danych i rozwiązywania problemów. Implementacja algorytmów testowych w Java i Kotlin ma zbadać różnice jakościowe i ilościowe pomiędzy szkieletami aplikacji internetowych (stworzonych w obu językach). Omówiona zostanie wydajność rozwiązań w technologii Java i Kotlin.
Zakres badania to:

- omówienie i porównanie wydajności tworzenia rozwiązań technologicznych na podstawie szkieletów aplikacji internetowych,
- implementacja i opis algorytmów testowych,
- analiza wydajności kodu wykonanych w obu językach programowania,
- testy porównawcze: szybkości wykonywania programów, zużycia pamięci, obciążenia procesora oraz wydajności odpowiedzi z bazy danych, wszystko w zależności od implementacji Kotlin lub Java.

4. Omówienie języków Java i Kotlin

4.1. Java

Projekt Java został zainicjowany w 1991 roku przez Jamesa Goslinga i jego współpracowników. Na początku język nazywał się „Oak" (dąb). Później nazwa projektu została zmieniona na „Java" na odniesienie do kawy Java. Z tego powodu logo tego języka to filiżanka kawy. Firma Sun Microsystems wydała Javę 1.0 w 1996 roku. Następnie nowe wersje wydawane były co 1-3 lata.
Od wersji Java 9 wydanej w 2017 r. Nowe wersje wydawane są co 6 miesięcy.

Java obsługuje wiele paradygmatów programowania, jest językiem imperatywnym opartym na koncepcji obiektowej: prawie każda część programu jest obiektem. Dlatego sam program można uznać za zestaw współdziałających obiektów. Ponadto częściowo obsługuje nowoczesne paradygmaty programowania, takie jak programowanie ogólne, programowanie współbieżne, programowanie funkcyjne i inne.

4.2. Kotlin

Kotlin to statycznie typowany język, w którym możemy pisać kod jak w językach dynamicznych bez utraty wydajności, jak reklamują twórcy - praca zbada ową wydajność [3]. Posiada takie właściwości jak:

- wsparcie dla inicjalizacji obiektu, klasy mogą mieć dane, które mogą być zmienne (var) lub tylko do odczytu (val),
- względnie lekka biblioteka standardowa w porównaniu do Javy,
- niska złożoność technologii oraz podobieństwo do Javy, oznacza dużą łatwość w zdobyciu biegłości w wyżej wspomnianym języku,

- możliwość wywoływania kodu Java bezpośrednio z bazy kodu Kotlin,
- wsparcie dla typów zmiennych non-nullable, zabezpieczenie przed błędem null pointer exception (null safety support),
- ulepszona semantyka w stosunku do Javy,
- brak średników w kodzie.

Firma JetBrains w lipcu 2011 r. zaprezentowała projekt Kotlin, jako nowy język dla platformy JVM. Nazwa technologii pochodzi od wyspy Kotlin, niedaleko Sankt Petersburga w Rosji. Głównym celem tego projektu było zapewnienie bezpieczniejszej i bardziej zwięzłej alternatywy dla języka Java we wszystkich kontekstach, w których Java jest obecnie używana. W 2016 roku została wydana pierwsza oficjalna stabilna wersja (Kotlin v1.0). Społeczność programistów była już zainteresowana korzystaniem z tego języka, zwłaszcza na Androidzie, z powodu walki patentowej firm Google oraz Oracle (2019 rok).

Na konferencji Google I / O 2017 ogłosił wsparcie dla Kotlin na Androida. W tej chwili Kotlin jest uważany za język ogólnego zastosowania dla wielu platform, nie tylko dla Androida. Język ma kilka wydań rocznie. Ponieważ Kotlin jest stosunkowo nowoczesnym językiem, nie posiada starych interfejsów API i przestarzałych koncepcji. Jest nowym pomysłem, względem starych paradygmatów. Najważniejsze nowe funkcje to:

- funkcje rozszerzające (możliwość przyklejania statycznych metod do istniejącego typu)
- leniwe właściwości (lazy properties, odroczenie inicjalizacji obiektów, optymalizacja)
- wsparcie dla typów zmiennych non-nullable, zabezpieczenie przed błędem null pointer exception (null safety support, nie można wywołać metody na obiekcie null)

Podstawowe koncepcje które odróżniają język Kotlin od Javy to posiadanie:

- wyrażeń lambda z funkcjami inline,
- funkcje rozszerzające (extensions functions), możliwość rozszerzania klas bez dziedziczenia po innych klasach,
- zabezpieczenie przed wywołaniem referencji null,
- rozszerzony mechanizm rzutowania (smart cast – operator is, as, as?),
- szablony dla typu String, ewaluacja wyrażeń za pomocą operatora $,
- klasa, może posiadać własności (properties – val, var),
- konstruktor główny i konstruktory dodatkowe,
- przeładowanie operatorów (jak w C++)
- wbudowany typ DTO (obiekty z typem 'data')

5. Metoda prowadzenia badań

Badanie stara się odpowiedzieć na pytanie: Czy istnieją znaczące różnice między wydajnością pomiędzy różnymi implementacjami tego samego testu porównawczego w Kotlin i Javie przy użyciu Wirtualnej Maszyny Java na przykładzie szkieletu aplikacji internetowej?

Jest wiele metod porównywania języków programowania. Jednym z najchętniej cytowanych badań nad różnymi językami programowania jest tzw. „The Computer Langugae Benchmark Game" (CLBG) [4]. Dynamiczne porównanie metryk będzie oparte na idei propagowanej przez Isaac Gouy. W 2000 roku rozpoczął się projekt CLBG, którego celem jest porównanie wszystkich głównych języków programowania, dzięki korzystaniu z obiektywnych testów porównawczych. Autorzy projektu zwracają szczególną uwagę na praktyczność wniosków z badań. Projekt skupia się na stworzeniu metryki, testu, który zależy od kontekstów takich jak:

1. cele badania,
2. konsekwencje możliwych błędów,
3. rodzaje ocenianego systemu komputerowego,
4. środowiska wykonawczego
5. infrastruktury i jakości testów porównawczych [5].

CLBG przedstawia zestaw 10 różnych problemów algorytmicznych. Wszystkie problemy zostały przedstawione i opisane na oficjalnej stronie internetowej [4]. Wprowadzone algorytmy mają ścisłe reguły dotyczące ich implementacji, z czego korzystają ich twórcy.

Biorąc pod uwagę wszystkie algorytmy, istnieją tylko trzy testy porównawcze CLBG, które zostały użyte w końcowym eksperymencie. Reszta testów porównawczych została odrzucona, aby zachować zgodność z założeniem: że kod Java musi być konwertowany na język Kotlin bez potrzeby wprowadzania dużych ilości zmian w kodzie. Dodatkowo, każdy użyty szkielet aplikacji internetowej zostanie przetestowany wydajnościowo na szybkość dostępu do bazy danych.

5.1. Implementacja i opis badania

Istnieją dwie implementacje dla każdego testu porównawczego w tym eksperymencie. Badanie zawiera algorytmy napisane w języku Java i Kotlin. Przypadek testu bazy danych zawiera przekonwertowany kod Java na Kotlin.

Wszystkie algorytmy zostały wykonane i zmierzone przez zewnętrzny skrypt Pythona (według zaleceń CLBG). Żadna z implementacji nie ma nieistotnych części kodu poza szkieletem aplikacji internetowej. Nadmiarowy kod, mogłyby zakłócić ostateczny wynik.

Pamięć podręczna i przestrzeń wymiany systemu plików są czyszczone przed wykonaniem pomiarów (skrypt Phyton CLBG) - każdy program ma podobny kontekst.

Wszystkie kody Java zostały pobrane bezpośrednio z repozytorium „The Computer Language Benchmark Game", bez dokonywania w nich zmian. Kody algorytmów w Kotlin, zostały stworzone na podstawie kodu Java z użyciem nowoczesnych wyrażeń z rozszerzenia języka. Kody użyte w eksperymencie to te, które osiągnęły najlepsze wyniki, zgodnie z tabelą wyników dostępnych

na stronie CLBG. Testy były uruchamiane w ramach aplikacji stworzonej w szkieletach aplikacji internetowych takich jak: Spring, Micronaut, Ktor.

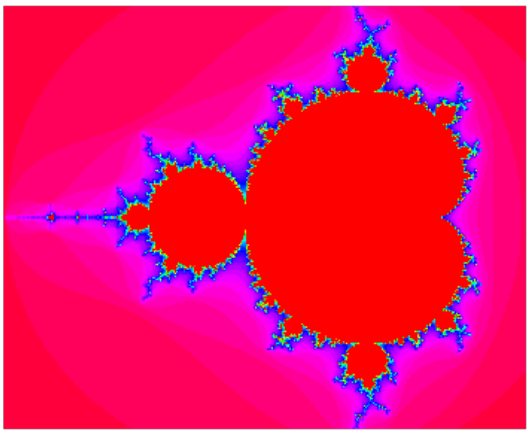
5.2. Opis użytych algorytmów

5.2.1. Mandelbrot

Termin zbioru Mandelbrota stosuje się zarówno w odniesieniu do ogólnej klasy zbiorów fraktalnych, jak i do konkretnego wystąpienia takiego zbioru. Zestaw Mandelbrota oznacza zbiór punktów na płaszczyźnie zespolonej, tak że odpowiadający mu zbiór Julii (dwa komplementarne tzn. będące swoimi dopełnieniami zbiory zdefiniowane przez odwzorowanie będące funkcją wymierną) jest przestrzenią spójną niemożliwą do obliczenia. „Mandelbrot" jest zbiorem uzyskanym z kwadratowego równania rekurencyjnego [6].

$$\boldsymbol{z}_{(n+1)} = z^2{}_i + C$$

$z_0 = C$ gdzie punkty C w płaszczyźnie zespolonej, dla których orbita z_n nie ma tendencji do nieskończoności, w zbiorze. Ustawienie równego każdemu punktowi w zestawie, który nie jest punktem okresowym, daje ten sam wynik. Zestaw Mandelbrota został pierwotnie nazwany przez Mandelbrota cząsteczką mu. Jest to zazwyczaj fraktal przedstawiany za pomocą różnych kolorów, aby można zaobserwować różnicę w równaniach (Rysunek 1). Użyty w badaniu algorytm wykreśla zbiór Mandelbrota dla zbioru [-1,5-i, 0,5 + i] na mapie bitowej N-na-N punktów. Algorytm zapisuje bajt po bajcie w formacie bitmapy. Wyjście programu jest ustawione dla N=6000.



Rysunek 1: Jedna z repreznetacji zbioru Mandelbrot

5.2.2. FASTA

FASTA jest to w skrócie format zapisu sekwencji kwasów nukleinowych oraz białek.

W bioinformatyce i biochemii format FASTA jest formatem tekstowym do reprezentowania sekwencji (DNA, RNA) lub sekwencji aminokwasowych (białkowych), w których nukleotydy lub aminokwasy są reprezentowane przy użyciu kodów jednoliterowych. Format pozwala na nazywanie i komentowanie sekwencji [9]. Format pochodzi z pakietu oprogramowania o tożsamej nazwie FASTA, obecnie stał się uniwersalnym standardem w dziedzinie bioinformatyki.

Linia z opisem rozpoczyna się od znaku "większe niż" ("¿"). Pierwsze słowo po tym znaku służy jako identyfikator sekwencji. Dalej w tej samej linii umieszczany jest opis. W kolejnych liniach znajduje się ciąg znaków składający się na sekwencję.

Przykładowa sekwencje w formacie FASTA wygląda następująco:
¿MCHU - Calmodulin - Human, rabbit ADQLTEEQIAEFKEAFSLFDKDGDGTITTKELGTVMRSLGQNPTEAELQDMINEVDADGNGTIDFPE*

Algorytm zastosowany w eksperymencie generuje sekwencje DNA, kopiując z dane z innej sekwencji. Następnie generuje sekwencje DNA na podstawie losowej selekcji z 2 tablic alfabetów.

5.2.3. Drzewo binarne

Algorytm do testowania wydajnościowego to uproszczona adaptacja GCBench - Hansa Boehm'a, która została zaadaptowana na podstawie testu porównawczego przez Johna Ellisa i Pete Kovaca.

Program polega na stworzeniu idealnych drzew binarnych - zanim jakiekolwiek węzły drzew zostaną poddane odśmiecaniu pamięci. Węzły liścia są takie same jak węzły wewnętrzne, pamięć jest przydzielana w ten sam sposób.

Program definiuję klasę i metody węzła drzewa. Przydziela drzewo binarne do pamięci, sprawdza poprawność i jego istnienie. Następnie dealokuje obiekt drzewa z pamięci. Po utworzeniu drzewa binarnego implementacja koncentruje się na obliczeniu długowiecznego drzewa lub poddrzewa z parametrami maxDepth i stretchDepth. Algorytm sprawdza drzewo i jego poddrzewa, aby znaleźć drzewo o najdłuższym okresie życia.
Aby je znaleźć, algorytm przydziela drzewo binarne.
Następnie je sprawdzana i cofa się po jego węzłach.
Po znalezieniu najdłużej żyjącego drzewa usuwa je z pamięci [4].

5.3. Metryki

Każdy test jest uruchamiany i mierzony przy najmniejszej wartości wejściowej, dane wyjściowe programu są przekierowywane do pliku i porównywane z oczekiwanymi danymi wyjściowymi ze strony CLBG. Tak długo, jak dane wyjściowe odpowiadają oczekiwanym wynikom. Program jest uruchamiany ponownie i testowany przy następnej większej wartości wejściowej, aż do wykonania wszystkich zadeklarowanych pomiarów.

Jeśli program zwróci prawidłowe dane w czasie krótszym niż 120 sekund, wtedy algorytm jest uruchamiany pięć razy ponownie. Jeśli program nie da oczekiwanego wyniku w czasie godziny, eksperyment jest przerywany.

Otrzymane dane z testów zostały skrócone o wyniki z najmniejszym i największym zużyciem pamięci dla

pomiarów. Testy trwające dłużej niż 2 godzin zostały wykonane tylko raz.

5.4. Sposób mierzenia czasu

Podjęta została decyzja o zastosowaniu oficjalnych technik mierzenia i metryk z benchmarku CLBG. Skrypty użyte do eksperymentu, są opisywane w innych pracach naukowych. Każdy algorytm jest uruchamiany jako proces dziecko ze skryptu Python, za pomocą funkcji Popen. Czas jest mierzony przed uruchomieniem algorytmu do do momentu jego zamknięcia.

5.5. Obciążenie procesora

Obciążenie procesora jest mierzone za pomocą funkcji QueryInformation JobObject(hJob,JobObject BasicAccountingInformation) Metoda zwraca całkowity czas wykonywania aktywnych procesów powiązanych z zadaniem od ostatniego wywołania.

5.6. Obciążenie pamięci

Funkcja użyta z API Windows nazywa się: QueryInformation JobObject(hJob,JobObject ExtendedLimitInformation)

5.7. Obciążenie dostępu do bazy danych

Test jest wykonany z pomocą 3 frameworków (Spring, Micronaut, Ktor), dla każdego stosując język Java i Kotlin.
Techniki wydajnościowe, użyte w badaniu to test aktualizacji danych w bazie. Techniki wydajnościowe, użyte w badaniu szybkości działania bazy danych na poszczególnym szkielecie to test aktualizacji danych w bazie. Test wydajnościowy sprawdza: - trwałość obiektów ORM, - wydajność sterownika bazy danych przy uruchamianiu instrukcji UPDATE. Test polega na wykonywaniu dużej liczby operacji na bazie danych w stylu odczytu i zapisu.

5.8. Środowisko eksperymentu

Użyte w badaniu środowisko do programowania Java (java -version):

- java version "11.0.7" 2020-04-14 LTS
- Java(TM) SE Runtime Environment 18.9 (build 11.0.7+8-LTS)
- Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.7+8-LTS, mixed mode)

Użyte w badaniu środowisko do programowania Kotlin (kotlinc -version):

- info: kotlinc-native 1.3.72-eap-463 (JRE 11.0.7+8-LTS)
- Kotlin/Native: 1.3.72

Środowsiko programistyczne, na którym zostały uruchomione testy przedstawia tabela 1 opisująca parametry dysku, tabela 2 pamięć RAM oraz tabela 3 CPU.

Tabela 1: Dysk

Model	Seagate ST1000LM024 HN-M101MBB
Wielkość	1 TB
Szybkość	5400 RPM
Wersja SATA	SATA 3.0 3Gb/s

Tabela 2: Pamięć RAM

Model	Seagate ST1000LM024 HN-M101MBB
Wielkość	8 GB
Taktowanie	800 MHz

Tabela 3: Procesor:

Architektura	x86_64
CPUs	4
Wątki na procesor	2
Nazwa modelu	Intel Core i3-3120M
Technologia procesora	22nm
Taktowanie CPU	2500 MHz

6. Testy

Skrypt w języku Python organizuje wykonanie eksperymentu. Język zarządzający środowiskiem dla tego typu badania nie ma wpływu na wyniki.
Został on wybrany na podstawie łatwości zaimplementowania i zintegrowania ze skryptem Python
z repozytorium „The Computer Language Benchmarks Game". Dane dotyczące wydajności, obciążenia procesora i pamięci zostały dostarczone przez system w postaci plików CSV. Reszta danych razem z wynikiem testów bazy danych, została uporządkowana manualnie. W celu ułatwienia pracy, badanie zostało zautomatyzowane.

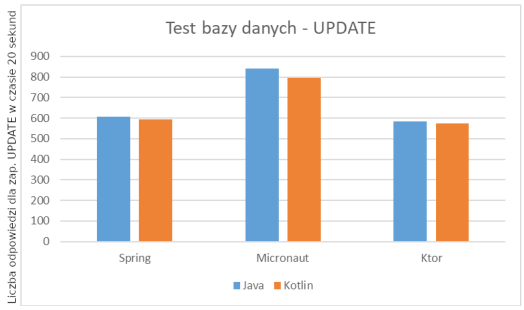
W tym celu skrypt wykonuje następujące kroki:

1. Przygotowywane jest środowisko testowe.
2. Przy pomocy narzędzi automatyzujących budowanie projektów (Maven), przygotowane zostają szkielety aplikacji internetowych. Każda ze zbudowanych aplikacji, posiada zaimplementowane testy wydajnościowe, które można uruchomić metodą http.
3. W zależności od badanego języka programowania, uruchamiany zostaje zadany szkielet aplikacji.
4. Zostają wywołane metody http uruchamiające poszczególne testy wydajnościowe. Testy są wywołane w kolejności: Fasta, Mandelbrot, tworzenie drzew binarnych, test bazy danych.
5. Po ukończeniu zadanych testów, wyniki zapisywane są do plików CSV.

7. Analiza wykonanych testów

7.1. Test odpowiedzi bazy danych

Porównanie wyników testów na bazie danych (liczba odpowiedzi na 20 sekund zapytań) pokazuje (Rysunek 2), że najbardziej optymalnym pod względem odpowiedzi jest szkielet Micronaut. Spośród trzech badanych platform, średnia dla Kotlina jest za każdym razem gorsza



Rysunek 2: Testy na bazie danych dla języków: Kotlin oraz Java (Spring, Micronaut, Ktor)

niż dla Javy. Eksperyment sprawdzający wydajność instrukcji UPDATE, pozawala zauważyć, że różnica w ilości odpowiedzi, może wynikać z różnicy szybkości działania Wirtualnej Maszyny Java.
Kotlin w każdym teście z bazami danych wypada gorzej niż Java.

7.2. Test obciążenia procesora

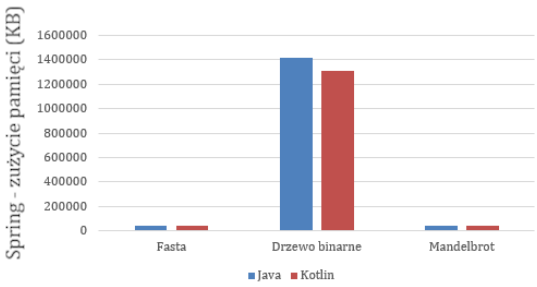
Obciążenie procesora jest podobne do siebie względem algorytmów i badanych języków programowania, niezależnie od szkieletu aplikacji – na której został uruchomiony test. Architektura procesora, wirtualna maszyna java, kod bajtowy – to wszystko ma znaczenie w tym eksperymencie.

Oba języki obciążają procesor w podobny sposób. Nie ma zauważalnego spadku obciążenia w żadnym z tych języków lub implementacji.

7.3. Zużycie pamięci

Znacząca zmienność wyników pomiaru zużycia pamięci jest widoczna w teście porównawczym „Drzewa binarne". Jak wspomniano wcześniej, test ten, głównie manipuluje liczbami całkowitymi. Istnieje różnica w jaki sposób, Java i Kotlin zarządzają liczbami całkowitymi.

7.3.1. Spring

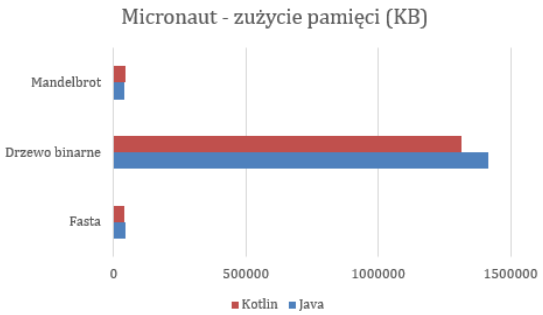


Rysunek 3: Test zużycia pamięci dla szkieletu Spring

W Javie liczby są w większości zapisywane jako typy wbudowane (prymitywne), w przeciwieństwie do Kotlina, gdzie wszystkie liczby są spakowane jako obiekty. Różnica ta jest widoczna na rysynku nr 3.

7.3.2. Micronaut

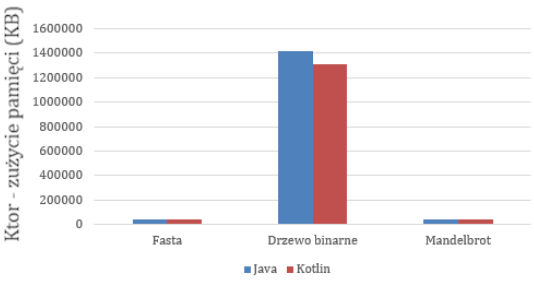
Micronaut to nowoczesny szkielet, oparty na JVM. Mikrousługa z pełnym wsparciem, zaprojektowany do budowania i testowania modułowych projektów.
Został opracowany przez twórców Grails i czerpie inspirację z tworzenia rzeczywistych aplikacji od monolitów po mikrousługi przy użyciu Spring, Spring Boot i Grails. Historycznie szkielet aplikacji, takie jak Spring i Grails, nie były zaprojektowane do działania w scenariuszach takich jak funkcje bez serwera (serverless - model usług w chmurze), aplikacje na Androida lub mikrousługi o niskim zużyciu pamięci. Micronaut został zaprojektowany odpowiednio dla tych wszystkich scenariuszy. Już



Rysunek 4: Test zużycia pamięci dla szkieletu Micronaut

rysunki 3 i 4 pozwalają stwierdzić, że użyty język i szkielet aplikacji nie koreluje, pod względem zużycia pamięci.

7.3.3. Ktor



Rysunek 5: Test zużycia pamięci dla szkieletu Ktor

Zużycia pamięci dla testu tworzenia drzew binarnych jest mniejszcze dla języka Kotlin niż Java również na szkielecie aplikacji Ktor, co widzać na rysunku 5.

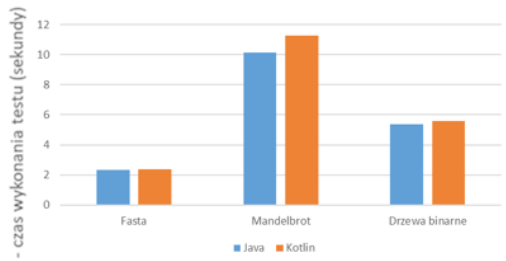
7.4. Analiza czasu wykonania algorytmów

Analiza czasu wykonania algorytmów oparta jest na technikach mierzenia i metryk z benchmarku CLBG. Każdy algorytm jest uruchamiany jako proces dziecko ze skryptu Python, za pomocą funkcji Popen. Czas jest mierzony przed uruchomieniem algorytmu do momentu jego zamknięcia.

7.4.1. Spring

Z wykresu rysunku 6 można dostrzec, że w przypadku testu na tworzenie fraktali, kod Java wykonuje się znacząco szybciej. Z danych, można zauważyć, że w każdym

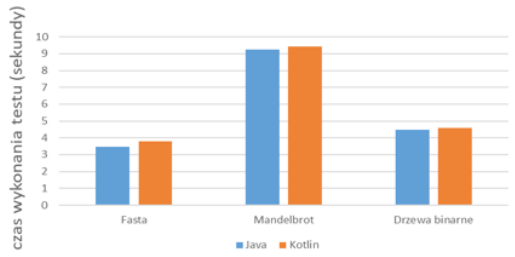
przypadku – test napisany w kodzie Java wykonuje się szybciej.



Rysunek 6: Test szybkości wykonania kodu dla szkieletu Spring

7.4.2. Micronaut

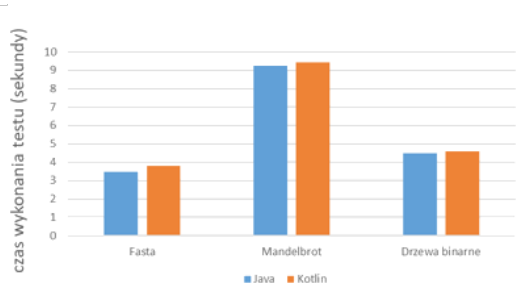
Podobnie jak we wcześniejszych testach wykres z rysunku 7, nie odbiega wynikami od innych szkieletów aplikacji internetowych. Wydajność jest niekorzystna dla technilogii Kotlin.



Rysunek 7: Test szybkości wykonania kodu dla szkieletu Micronaut

7.4.3. Ktor

Rysunek 8, przedstawia wyniki testu dla szkieletu Ktor. Pomiędzy szkieletami aplikacji internetowymi, można dostrzec niewielką różnicę w czasie wykonywania zadań. Wynika ona, prawdopodobnie z niedeterministycznego środowiska, na którym odbywały się testy.



Rysunek 8: Test szybkości wykonania kodu dla szkieletu Ktor

8. Wnioski

Aby zrozumieć koncepcję porównywania języków na przykładzie szkieletów aplikacji internetowej, w pracy przygotowano, krótki wstęp w tematykę semantyczną Java i Kotlin. Opisano przebieg generowania kodu wykonywalnego, wraz z użyciem Wirtualnej Maszyny Java.

Następnie na podstawie przeglądu artykułów naukowych wybrano i omówiono sposób porównywania. Trzy algorytmy testujące języki programowania z „The Computer Langugae Benchmark Game" (CLBG) zostały przedstawione jako metryka dla badań. Każdy z algorytmów testowych został szczegółowo opisany. Dodatkowo przedstawiono i przetestowano wydajność zapytań do bazy danych na różnych szkieletach aplikacji internetowej z użyciem Java i Kotlin.

Został uruchomiony skrypt zarządzający testami. Zmierzona została ilość odpowiedzi z bazy danych w czasie 20 sekund wysyłania zapytań z poszczególnego szkieletu aplikacji internetowej do bazy danych H2. Następnie skrypt Python z repozytorium kodu „The Computer Langugae Benchmark Game" uruchomił na każdym z szkieletów tj. Spring, Micronaut, Ktor zestaw testów:

- badanie obciążenia procesora,
- badanie czasu wykonywania programu,
- zużycie pamięci.

Kolejnym krokiem była analiza na podstawie wyników. Analizowano współczynniki takie jak: ilość odpowiedzi z serwera w zadanym czasie, procentowe obciążenie CPU, szybkość zwracania wyników w sekundach oraz zużycie pamięci w KB. Wnioski z pracy są następujące: Implementacja w języku Kotlin nigdy nie osiągnęła najlepszego wyniku w żadnej grupie pomiarów. Zróżnicowanie pod względem szkieletów aplikacji internetowych, szybkości odpowiedzi bazy danych i implementacji testów w różnych językach – we wszystkich testach wychodzi na niekorzyść języka Kotlin. We wszystkich przypadkach, wydajność była gorsza o średnio 5,6%. Pomiędzy szkieletami Spring i Ktor, nie dostrzega się dużej różnicy. Na tle obu wyróżnia się Micronaut z najlepszymi wynikami w badaniu. Szkielet Micronaut nie korzysta z refleksji, co może wpływać na lepszą wydajność w stosunku do konkurencyjnych rozwiązań.

Nie ma znaczącej różnicy pomiędzy obciążeniem procesora pomiędzy poszczególnymi pomiarami, różnica nie przekracza 2%. Takie różnice mogą wynikać ze względu na środowisko badawcze tj. sprzęt komputerowy i oprogramowanie systemu, W dziedzinie testów CLBG, porównując parametr szybkości wykonania algorytmu, najszybciej wykonał się test Mandelbrot dla szkieletu Spring. W każdym z testów szybciej wykonują się programy zaimplementowane w języku Java na każdym z szkieletów internetowych.

Nie ma dużej różnicy pomiędzy szkieletami aplikacji. Największe znaczenia ma język implementacji. Kod Java osiąga najlepszy czas wykonania, we wszystkich testach. Wynika, to z optymalizacji Wirtualnej Maszyna Java dla tego języka.

Porównanie zużycia pamięci okazało się najniższe w implementacji Java, na wszystkich aplikacjach internetowych i uruchomionych testach, z jednym wyjątkiem. Kotlin zużywa mniej pamięci w implementacji te-

stu tworzenia drzew binarnych. Jest to 8% różnica na korzyść Kotlin.
Szkielety aplikacji internetowych nie wykazują większego wpływu na zróżnicowanie badania.

Głębsza analiza danych pozwoliłaby na wyciągniecie jeszcze większej ilości wniosków. Wyniki z badań zawartych w tym artykule skłaniają do kolejnych badań i porównań, skupionych na kodzie bajtowym i Wirtualnej Maszynie Java.

Literatura

[1] M. Martinez, B. Gois. How and Why did developers migrate Android Applications from Java to Kotlin? A study based on code analysis and interviews with developers, arXiv preprint arXiv:2003.12730 (2020).

[2] S. Bose, A comparative study: java vs kotlin programming in android application development, International Journal of Advanced Research in Computer Science (9) (2018) 41-45.

[3] T. Kalibera, R. Jones, Rigorous benchmarking in reasonable time, in Proceedings of the 2013 international symposium on memory management (2013) 63-74.

[4] I. Gouy, The Computer Language Benchmarks Game. Web.(https://benchmarksgame-team.pages.debian.net/benchmarksgame/).

[5] A. Prokopec, Oracle Labs: On Evaluating the Renaissance Benchmarking Suite: Variety, Performance, and Complexity, arXiv preprint arXiv:1903.10267 (2019).

[6] P. Alfeld, The Mandelbrot Set (https://www.math.utah.edu/~alfeld/math/mandelbrot/mandelbrot.html)

[7] D. Stepanov, M. Akhin, M. Belyaev, How We Stopped Worrying About Bugs in Kotlin Compiler, in 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE (2019) 317-326.

[8] B. G. Mateus, M. Martinez, An empirical study on quality of Android applications written in Kotlin language, Empirical Software Engineering (2019) 3356-3393.

[9] D. J. Lipman, W. R. Pearson, Rapid and sensitive protein similarity searches, Science 227 (4693) (1985) 1435–1441.

[10] TIOBE index (https://www.tiobe.com/tiobe-index/)