Spring - wstęp

Spis treści

Czym jest Framework? . . . 1
Spring . . . 1
Spring nie jest na rynku od wczoraj . . . 1
Po co w ogóle frameworki? . . . 2
Czemu w ogóle mówimy o Springu? . . . 2
Co Spring nam daje? . . . 3

Czym jest Framework?

Zanim przejdziemy do omawiania czegokolwiek, przypomnijmy sobie na początku jaka jest różnica między biblioteką a frameworkiem. Zobaczysz, że zacznie się to coraz bardziej rozjaśniać im bardziej będziemy wchodzić w tematykę głównego frameworka wokół którego będziemy się skupiać - `Springa`.

Czym różni się `library` od `framework`? Klik

- **library** - zestaw funkcji/metod, które można wywołać, które są zorganizowane w postaci klas. Po każdym wywołaniu funkcji z biblioteki to my przejmujemy kontrolę nad dalszym wywołaniem kodu,
- **framework** - jest abstrakcyjnym projektem, który posiada wbudowane zachowania. Aby go używać, wstawiamy swoje własne zachowania/metody w różne miejsca we frameworku np. przez tworzenie własnych klas i wstawianie ich we framework. Następnie kod frameworka wykonuje nasz kod w punktach, które wstawiliśmy.

Takie są definicje, natomiast w praktyce często jest to używane wymiennie ☺.

Spring

Dlaczego przypomnieliśmy sobie czym jest framework? Bo przechodzimy do omówienia jednego z najczęściej używanych frameworków Javy. Obecnie na rynku sytuacja wygląda w ten sposób, że jeżeli mówimy o jakimś projekcie pisanym w Javie to jest bardzo duże prawdopodobieństwo, że projekt ten jest oparty o Springa.

Spring nie jest na rynku od wczoraj

Zgodnie z Wikipedią, Spring powstał w roku 2003. Czyli dawno. Czy oznacza to, że jest stary i nie powinniśmy się go uczyć? Nie. Java też powstała wiele lat temu, a ciągle jest jednym z najbardziej popularnych języków programowania. Ważne jest to żeby technologie, których używamy były cały czas rozwijane i cały czas były do nich dokładane nowe funkcjonalności. Tak samo jest w przypadku corowej Javy jak i w przypadku Springa.

Po co w ogóle frameworki?

Ten fragment będzie dosyć filozoficzny. Zastanów się co jest celem zatrudnienia programisty w firmie?

Firma służy do zarabiania pieniędzy. Rozwój firmy (w uproszczeniu) polega na wzroście przychodów oraz możliwej redukcji kosztów. Czyli firmy chcą sprzedawać te same produkty lub usługi, w coraz większych ilościach przy jednoczesnej redukcji możliwych kosztów. Przykładem może być tutaj infolinia w jakiejś dużej firmie. Chyba nikogo nie muszę przekonywać, że taniej (w długiej perspektywie) dla takiej dużej firmy jest przygotować system, żeby klient mógł się obsługiwać całkowicie samodzielnie zamiast dzwonić na infolinię i rozmawiać z człowiekiem, który i tak będzie za nas wyklikiwał wszystko na komputerze. Dlatego w długiej perspektywie wychodzi taniej, bo komputer nie musi jeść, potrzebuje jedynie prądu. Program napisany raz, działa całą dobę, nie musi chodzić na przerwy itp.

Mam nadzieję, że czujemy już po co powstają systemy informatyczne. Odpowiedzmy sobie zatem na pytanie, czy lepiej jak programista musi codziennie walczyć z problemami technicznymi, czy lepiej jak może w pełni skupić się na rozwijaniu funkcjonalności biznesowych - czyli takich, które faktycznie dają jakiejś firmie przychód. No oczywiście, że to drugie! I to właśnie w tym celu powstają frameworki, czyli gotowe rozwiązania, które mają nam dać możliwość skupienia się na realizacji celów biznesowych, tak żeby nie trzeba było cały czas walczyć z techniczną konfiguracją, albo ustawianiem parametrów, albo z innymi rzeczami, które są powtarzalne i mogą zostać wykonane raz i trzymać się schematu. Zacznie Ci się to klarować jak będziemy wchodzić coraz bardziej w tego typu rozwiązania.

Czemu w ogóle mówimy o Springu?

Dlaczego, skoro uczymy się o Javie, to nagle został wyciągnięty `Spring`? Trzeba spojrzeć na to jak wygląda struktura narzędzi, które możemy wykorzystywać w naszym arsenale umiejętności.

Zanim natomiast opowiemy sobie o tym, dlaczego w ogóle Spring został powołany do życia, należy cofnąć się lekko w historii Javy.

Przypomnijmy sobie, że na Javę składa się kilka jej edycji. Java SE, ME, FX oraz Java EE (podział w uproszczeniu). W roku 1999 do życia została powołana Java J2EE 1.2. Nie będziemy się tutaj zagłębiać w specyfikę nazewnictwa, bo jak to w Javie bywa, na przestrzeni lat zmieniało się to nieustannie. Dlatego dla uproszczenia nazwiemy to sobie Java EE. Java EE (Java Enterprise Edition), powstała jako platforma do tworzenia aplikacji biznesowych (czyli tych co realizują jakiś biznes, żeby zarabiać pieniądze), która miała na celu upraszczać rozwiązywanie problemów powtarzających się podczas wytwarzania oprogramowania. Java Enterprise składała się z zestawu specyfikacji, które miały nam pomagać w komunikacji z bazami danych, w komunikacji z innymi serwerami, w tworzeniu stron HTML itp.

Założenie Javy EE było bardzo słuszne. Programista powinien skupiać się na realizowaniu funkcjonalności biznesowych, takich, które faktycznie służą do zarabiania pieniędzy albo zmniejszania kosztów przedsiębiorstwa dla którego pracuje. Jeżeli ktoś "techniczny" spędza godziny, a nawet dni na konfigurowaniu serwerów i walce z technologią to jest to w pewnym sensie "marnotrawstwo" bo walka taka generuje koszty, a nie przychód dla firmy. Nie są w tym czasie wytwarzane żadne funkcjonalności biznesowe, które pomagają zarabiać pieniądze albo zmniejszyć koszty. Celem Javy EE było również ujednolicenie interfejsów, które były następnie wykorzystywane przez dostawców serwerów aplikacyjnych (taki program, który uruchamia nasz program w Internecie). Czyli chciano tak podejść do tematu, żeby programista spędzał jak najwięcej czasu na realizacji celów biznesowych, a nie "kopał się" z konfiguracją. Ale coś nie pykło …

Java EE nie była zbyt lubiana przez programistów. Miała dosyć słabą dokumentację i dosyć mało przydatnych przykładów (tutoriali). Serwery aplikacyjne były "ciężkie" (w uproszczeniu można powiedzieć, że zawierały bardzo dużo kodu, który i tak często był nieużywany). Serwery aplikacyjne były często niestabilne, więc jednak programiści musieli "kopać się" z konfiguracją zamiast pisać "ficzery" biznesowe. Oczywiście Java EE dawała bardzo dużo możliwości, ale w ogólnym rozrachunku nie cieszyła się "ciepłym" odbiorem.

Wtedy właśnie (w roku 2002) pojawił się Spring framework, który został zapoczątkowany przez jedną osobę - Rod Johnson. Spring zaczął powstawać w dosyć ciekawym momencie - gdy programiści byli już zmęczeni Javą EE i oczekiwali czegoś nowego. Sytuacja była o tyle ciekawa, że duże korporacje forsowały używanie Javy EE, natomiast programiści jako ludzie, którzy faktycznie używali tych narzędzi, zaczęli się skłaniać w stronę Springa. Dlaczego? Spring zaczynał mieć coraz lepszą dokumentację, nie wymagał stosowania "ciężkich" serwerów aplikacyjnych jak Java EE, był mniej błędogenny i zwyczajnie o wiele przyjemniej się go używało. Wokół Springa zaczęła rozrastać się ogromna społeczność i można powiedzieć, że wyszedł on zwycięsko z tego starcia 😉.

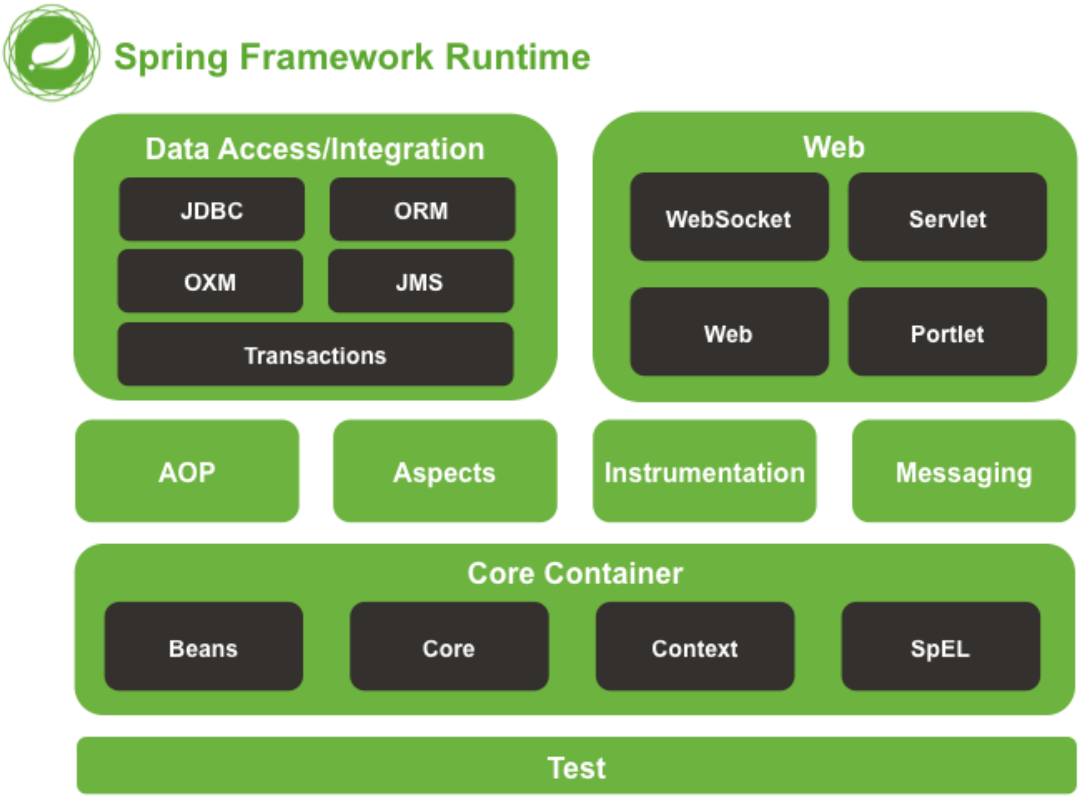
W realiach czasów dzisiejszych, nie znać Springa to jak nie wiedzieć, gdzie ma się toaletę w domu (możesz sobie zapisać ten cytat razem z autorem 😄). Dlatego właśnie przejdziemy do omówienia jego bardziej popularnych elementów. Sam Spring jest wielkim projektem i ja osobiście nie znam osoby, która zna jego wszystkie możliwe zakamarki 😄.

Co Spring nam daje?

Potrzeba istnienia frameworków wynikała z tego, że bez nich każdy nowy projekt w Javie wymagałby konfiguracji od zera. Konfiguracja taka jest często bardzo zbliżona. Wydaje się zatem idealnym rozwiązaniem, żeby móc mieć zapewnioną konfigurację projektu, która odpowiada jakimś wypracowanym schematom jako bazę i ewentualnie tę konfigurację modyfikować. Spring został stworzony zgodnie z zasadą "Convention over configuration". Oznacza to, że jeżeli trzymamy się ogólnie przyjętej konwencji, to zmniejszamy ilość kodu, jaki programista musi napisać, ponieważ może wykorzystać elementy kodu, które dostarcza nam już wcześniej framework. Jeżeli natomiast potrzebujemy jakiejś "customizacji" - nadal jest ona możliwa.

`Spring` jest projektem open-source, oznacza to, że wielu programistów przyczynia się do jego rozwoju, natomiast ostateczna decyzja co do wprowadzania zmian w samym kodzie frameworka należy do `Spring Development Team`. Cały kod frameworka jest dostępny publicznie na GitHub i każdy może proponować zmiany w jego działaniu.

Spring opiera się o ideę `DI` oraz `IoC`. Jeżel spojrzymy na program Java "z góry" to zauważymy, że każdy program to zbiór komunikujących się wzajemnie obiektów. Używanie Springa jest bardzo wygodne ze względu na to, że pozwala on w bardzo prosty sposób łączyć oraz rozłączać rozmawiające ze sobą obiekty. Możemy to porównać przez analogię do klocków Lego. Możemy zestawiać wiele klocków ze sobą, wymieniać je, łączyć oraz rozdzielać. Sam Spring to gigant składający się z wielu projektów, z kórych można dowolnie korzystać. Tak samo jak w przypadku Spring Framework, który składa się z wielu modułów, które możesz zobaczyć na poniższej grafice.



Obraz 1. Źródło: Dokumentacja Spring

Dzięki tej elastyczności w łączeniu ze sobą obiektów, również moduły mogą być przez nas dołączane i odłączane od całego projektu. Właśnie dzięki takiemu podejściu projekt ten zyskał tak ogromną popularność - **bierzemy tylko to co jest nam potrzebne i tylko z tego korzystamy**.

A sam Spring jest oczywiście napisany w Java, dlatego oglądanie jego kodu źródłowego powinno być dla nas w miarę czytelne ☺.