Daniel Górski
Korporacyjna Java
Wykład 12: Wybrane biblioteki / framework'i

Kalendarz

- Dzisiaj
  - Wykład
  - Ostatnie wymagania na pracowni
- 22 maja
  - Podsumowanie pracowni
  - Wstępne oddawanie całych projektów
    - Zaczynamy o 8:15 (brak prezentowania przed wykładem)
    - Kilka scenariuszy testowych do przejścia
    - Prezentacja kodu
    - Statyczna prezentacja przepływu sterowania w różnych scenariuszach
    - Zbadanie efektywności testów

Kalendarz...

- 29 maja: zajęcia piątkowe wg kalendarza roku akademickiego
- 5 czerwca
  - Będę dostępny 7:30 – 11:30
  - Możliwość ostatecznej prezentacji projektu i wystawienie ocen
- 12 czerwca: Wykład
  - Kilka osób podłączy się do zdalnej konferencji i opowie o swojej ścieżce edukacyjnej i zawodowej
    - Możliwość dyskusji
    - Będą to osoby z 15 – 20 letnim doświadczeniem w IT

Kalendarz...

- 12 czerwca: Pracownia
  - Będę przeprowadzał testowe rozmowy techniczne
    - Zakres: java, bazy danych, architektura, systemy rozproszone
  - Oceny z pracowni będą już wystawione: na nic to nie będzie rzutować

Wybrane biblioteki / framework'i

Biblioteki w Javie

- Siłą języka jest bardzo duża ilość dostępnych bibliotek / framework'ów
- Część z nich wyprzedza nawet swoją epokę i wprowadza rozwiązania wchodzące później do standardu
- Wiele jest porzuconych / niewspieranych
- Bardzo dużo jest aktywnie rozwijanych
- Wiele błędów jest powszechnia znanych, a prawdopodobnie jeszcze więcej pozostaje w ukryciu :-)

Mockito

- Framework testowy
- Wspiera tworzenie testów jednostkowych dostarczając funkcjonalność tworzenia Mock'ów
- Mock jest obiektem który "udaje" inny obiekt
  - Można dzięki temu testować jedną rzecz bez dostarczania całego potrzebnego kontekstu (repozytoria, serwisy...)
  - Możemy również symulować różnego rodzaju nieoczekiwane sytuacje

Mockito...

- Przykładowy zaawansowany scenariusz użycia:
  - Tworzymy obiekt "Zarządcy" z Mock'iem repozytorium
  - Wywołujemy testowaną metodę tworzenia figury na "Zarządcy"
  - Sprawdzamy czy w Mock'u repozytorium była wywoła metoda dodania figury
  - Sprawdzamy parametry od jakich ta metoda była wywołana

Mockito...

- Mamy bardzo duże możliwości definiowania Mock'ów:
  - Możemy np zdefiniować Mock'a tak aby dla pewnych parametrów rzucał pewien wyjątek
  - Możemy zdefiniować zwracaną wartość z metody:
    - Zawsze ta sama wartość
    - Dla pewnych wzorców parametrów konkretna
    - Wyliczana: np zawsze 1 parametr metody dodać 5
- Daje to Nam możliwości pisania jednostkowych testów zaawansowanych funkcjonalności

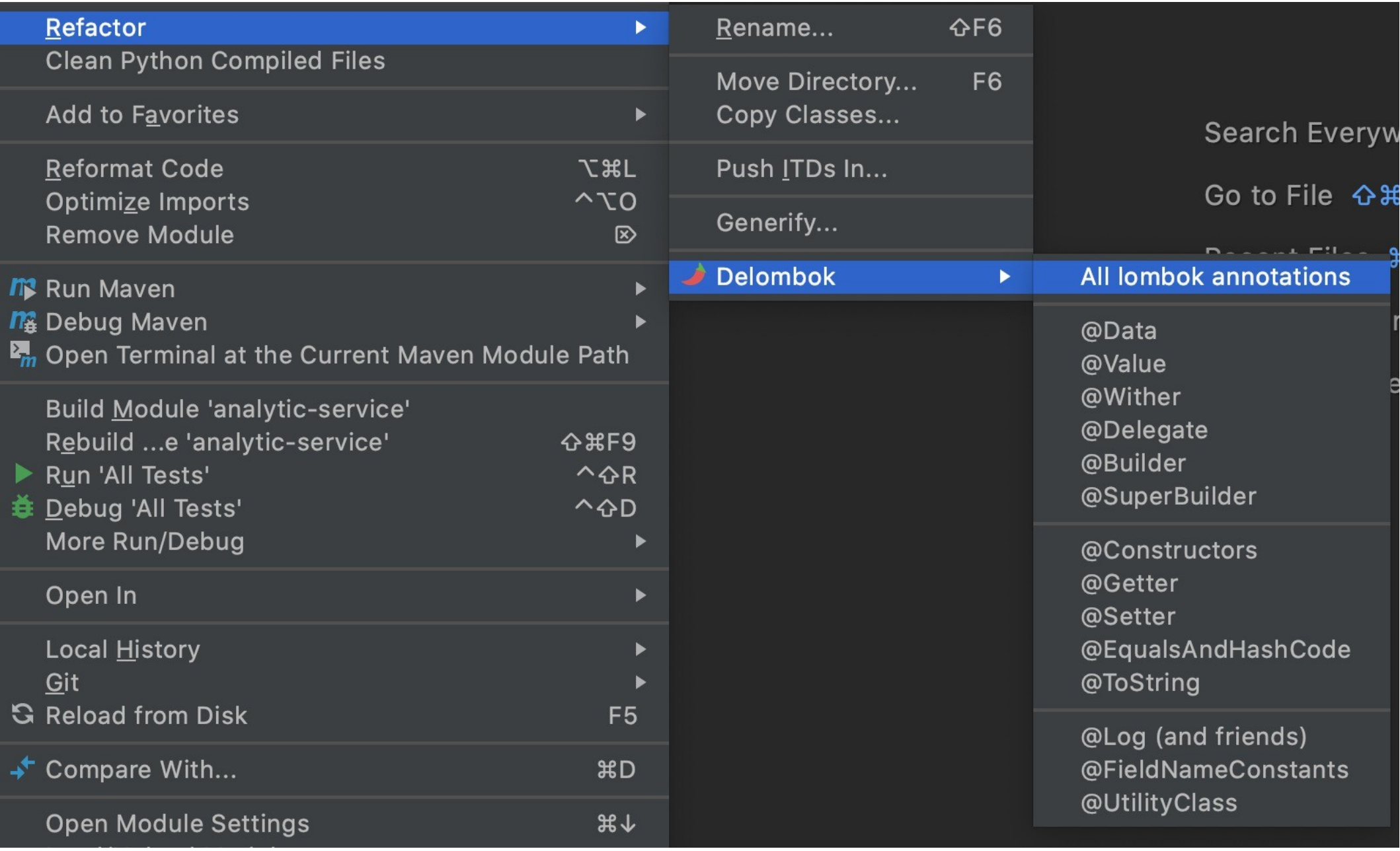
Lombok

- Nie da się ukryć, że w Javie tworzy się wiele powtarzalnego kodu
- Z pomocą przychodzą IDE, które potrafią na życzenie wygenerować wiele standardowych rzeczy
- Niektóre usprawnienia wchodzą do języka, jak np rekordy
- Lombok jest bardzo ciekawym usprawnieniem pracy

Lombok...

- @Getter
- @Setter
- @ToString
- @NoArgsConstructor
- @EqualsAndHashCode
- @ToString
- @Builder
- @RequiredArgsConstructor
- @AllArgsConstructor
- @Data
- @Value
- @With
- A co jeśli chcemy dużo kodu wygenerować, a tylko jedną rzecz zmienić?

Delombok


Refactor
Clean Python Compiled Files
Add to Favorites
Reformat Code ⌥⌘L
Optimize Imports ^⌥O
Remove Module
Run Maven
Debug Maven
Open Terminal at the Current Maven Module Path
Build Module 'analytic-service'
Rebuild ...e 'analytic-service' ⇧⌘F9
Run 'All Tests' ^⇧R
Debug 'All Tests' ^⇧D
More Run/Debug
Open In
Local History
Git
Reload from Disk F5
Compare With... ⌘D
Open Module Settings ⌘↓
Rename... ⇧F6
Move Directory... F6
Copy Classes...
Push ITDs In...
Generify...
Delombok
All lombok annotations
@Data
@Value
@Wither
@Delegate
@Builder
@SuperBuilder
@Constructors
@Getter
@Setter
@EqualsAndHashCode
@ToString
@Log (and friends)
@FieldNameConstants
@UtilityClass
Search Everyw
Go to File ⇧⌘

Jenetics

- Projekt istnieje od około 10 lat
- Wciąż jest rozwijany
- API do algorytmu genetycznego, dla problemu optymalizacyjnego definiujemy:
  - Chromosom z genami
  - Funkcję oceny
  - Parametryzację wyszukiwania
- Uzyskujemy chromosom z dosyć dobrym rozwiązaniem

JTS (Java Topology Suite)

- Rozwijana od przeszło 20 lat
- Biblioteka z geometrią analityczną / obliczeniową
- Wykorzystywana w wielu innych projektach:
  - PostGIS (typy i zapytania przestrzenne w bazie PostgreSQL)
  - QGIS i inne aplikacje wspierające pracę w tym obszarze
  - Google Earth
- GIS: Geographic Information System

Selenium IDE

- Rozwijany od około 20 lat
- Framework do testowania aplikacji w przeglądarce
- Bazowa wersja to nagrywanie scenariuszy testowych i ich odpalanie
- Rozszerzona to testowanie aplikacji w przeglądarce za pomocą programu w Javie komunikującego się z Selenium IDE
  - Możliwość tworzenia bardzo kontekstowych przypadków testowych

Spring

- Wszechstronny framework do tworzenia aplikacji
- Początki w 2002, wersja 1.0 w 2004
- Obecnie najnowsza wersja to 6.1.4

IoC i DI

- IoC: Inversion of Control
  - Bardziej paradygmat niż wzorzec projektowy
  - Sterowanie sprawuje framework, wywołuje on kod programu, ale kontroluje go i może przerwać jego pracę
- DI: Dependency Injection
  - Jedna z możliwych implementacji elementów IoC
  - Programista może "wstrzyknąć" do swojego kodu np inne serwisy, np poprzez adnotacje
    - Mogą być leniwie inicjalizowane w momencie gdy są potrzebne

Spring Data

- Framework dostępu do danych w bazie ORM
- Wewnętrznie korzysta z implementacji JPA przez Hibernate
- Wprowadza swój obiektowy język zapytań: JPQL zbliżony do HQL
- Po co Nam w takim razie kolejna warstwa abstrakcji w dostępie do danych?
  - Prostota obsługi 90% przypadków użycia w dużych projektach biznesowych

Spring Data...

- Tworzymy:
  - interface UserRepository extends CrudRepository<User, Long>
- Od razu mamy metody:
  - User save(User entity);
  - Optional<User> findById(Long primaryKey);
  - Iterable<User> findAll();
  - long count();
  - void delete(User entity);
  - void deleteAll()
  - boolean existsById(Long primaryKey);
  - ...

Spring Data...

- Możemy dodać do interfejsu swoje definicje metod z zapytaniem w adnotacji
- Dodajemy jedynie definicje metod np:
  - List<Person> findByLastname(String lastname);
  - List<Person> findByNameAndLastname(String name, String lastname);
  - List<Person> findByNameAndAgeGreaterThan(String name, Long age);
  - long countByName(String name);
  - long deleteByAgeLessThanEqual(Long age);
- Nic nie implementujemy, Spring Data dostarczy Nam implementację poprzez DI

Spring AOP

- AOP: Aspect Oriented Programming
- Dodatkowa warstwa abstrakcji (proxy) na kod Naszego programu
- Możemy stworzyć coś na wzór programowego breakpoint'a
  - Możemy podpiąć się np pod metody zawierające w nazwie wzorzec, w zadanym pakiecie, w klasach implementujących zadany interfejs
  - Podpinamy Naszą metodę i mamy pełną kontrolę: widzimy od jakich parametrów została wywołana zadana metoda, możemy je podmienić, możemy rzucić wyjątek, możemy po prostu wywołać metodę, możemy zmienić wynik zwracany...

Spring AOP...

- Może zwiększyć to bardzo mocno złożość projektu
- Używane z wyczuciem może jednak uporządkować / uprościć część operacji
  - Klasyczny przykład to sprawdzanie uprawnień użytkownika do wywoływania krytycznych metod
  - Może być użyte w celu zrobienia zaawansowanego logowania kontekstowego
  - Można w ten sposób pracować nad starymi / niemodyfikowalnymi / zewnętrznymi bibliotekami aby w locie zmieniać ich zachowanie

Architektura mikroserwisów RESTowych

- Obecnie bardzo popularną architekturą jest architektura mikroserwisów RESTowych
  - W pewnym stopniu pojawiają się przejścia z REST na GRPC: podyktowane jest to lepszą wydajnością GRPC
- Innym rodzajem architektur są architektury klasy: event-driven

Co to jest ten REST

- **Re**presentational **S**tate **T**ransfer
- Jest to styl architektury (nie ma oficjalnej standaryzacji REST)
- Jest to architektura klient – serwer z wyraźnym rozgraniczeniem:
  - Dane są przechowywane po stronie serwera
  - Klient komunikuje się z serwerem poprzez zdefiniowane HTTP API
    - Przede wszystkim metody: GET / PUT / POST / DELETE
  - Jest to komunikacja bezstanowa (stateless)

API RESTowe

- Komunikacja klient – serwer jest bezstanowa
  - Nie są utrzymywane sesje
  - Każde zapytanie do serwera musi zawierać komplet potrzebnych informacji
    - Wywoływana usługa
    - Dane uwierzytelniające (jeśli są potrzebne)
    - Parametryzacja
  - W odpowiedzi klient otrzymuje:
    - Kod odpowiedzi (może to być kod błędu)
      - 4xx – błędy z winy klienta: niepoprawne zapytanie
      - 5xx – błędy serwera
    - Zwykle otrzymuje dane

Główne metody serwisów REST

- GET: pobieranie danych
- PUT: edycja
- DELETE – usunięcie
- POST – tworzenie
- GET, PUT i DELETE są metodami idempotentnymi
  - Z perspektywy stanu serwera nie ma różnicy czy zostaną zaaplikowane raz czy wiele razy

Główne zalety architektury REST

- Lekkość: bazuje na standardzie HTTP
- Niezależność od technologii: komunikacja może być prowadzona w różnych formatach danych
  - Przede wszystkim JSON, XML, HTML...
- Skalowalność
  - Zapytania bezstanowe są zazwyczaj dobrze skalowalne
  - Zapytania idempotentne są dobrze cache'owalne

Serwisy REST w Spring'u

- Spring Boot daje Nam możliwość uruchomienia serwisu z minimalną ilością dedykowanej konfiguracji
- Są generatory projektów jak:
  - https://start.spring.io/
- Możemy szybko wygenerować np plik *.pom dla projektu

H2

- Jest to baza SQL w pamięci
- Często jest to dobra rzecz na początek projektu
  - Nie wymaga konfiguracji
  - Są minimalne różnice w różnych przypadkach brzegowych względem innych baz SQL, ale... różne bazy SQL mają różnice pomiędzy sobą

Stworzenie projektu Spring Boot z API Webowym i bazą w 15 minut

- Stworzyć plik pom dla projektu przez: https://start.spring.io/ wraz z zależnościami
  - H2 Database
  - Spring Web
  - Spring Data JPA
- Zaimportować projekt do IDE
- Uruchomić

Przykładowy serwis

- @RestController
- @GetMapping("/version")
  - public String getVersion()
- @GetMapping("/square/{number}")
  - public Long getSquare(@PathVariable("number") Long number)

Przykładowa encja

- @Entity
- @Table
- @Id
  - @GeneratedValue(strategy = GenerationType.AUTO)
- @Column

Dalsze kroki...

- Dodać persystentną bazę
- Wykonać deployment w chmurze
- Testować i implementować funkcjonalności
- Stworzyć frontend

Pracownia: co wygląda dobrze

- Stabilizacja architektury fabryk dla figur
  - Dodawanie nowej figury wymaga tylko dodania kodu związanego z tą figurą
  - S**O**LID: łatwa rozszerzalność kosztem trudnej modyfikalności: jeśli chcemy mocno przebudować sposób działania to zbieramy wymagania i piszemy nowy kod

Pracownia

- Program ma umożliwić rozwiązywanie podstawowych zadań geometrycznych dla sześiokąta foremnego
- Program ma umożliwić zmianę wersji językowej na polską lub angielską
  - W przypadku sterowania poleceniami ma to być konfigurowalna jednojęzyczność, a nie dwujęzyczność
  - Json do pliku nie musi być tłumaczony
- Program ma uniemożliwić dodawanie dubli
  - We wszystkich sposobach tworzenia figur
  - Wyświetlamy komunikat, że dodanie figury się nie powiodło, bo taka już istnieje wraz z datą utworzenia

Pracownia

- Dlaczego reprezentacja tego samego bytu na różne sposoby jest tak duzym problemem?
  - Powoduje to różne sposoby przetwarzania / prezentacji tego samego bytu
  - W dużych, długoterminowych projektach informatycznych wiele rzeczy może się zdarzyć
  - Jest duże prawdopodobieństwo rozsynchronizowania się tego kiedyś

Sześiokąt foremny

- Charakterystyka figury
  - Długość boku
  - Pole powierzchni
  - Obwód
- Możliwe wejście:
  - Długość boku
  - Pole powierzchni
  - Obwód