**Pytania rekrutacyjne**

# Java

## 1.Co to jest JVM?

maszyna wirtualna oraz środowisko zdolne do wykonywania [kodu bajtowego Javy](https://pl.wikipedia.org/wiki/Kod_bajtowy_Javy).

Wirtualna maszyna Javy to zestaw aplikacji napisanych na tradycyjne urządzenia i [systemy operacyjne](https://pl.wikipedia.org/wiki/System_operacyjny). Dostarcza środowiska, w którym może się wykonywać program skompilowany do postaci [kodu bajtowego Javy](https://pl.wikipedia.org/wiki/Kod_bajtowy_Javy), zapewniając takie usługi, jak [odśmiecanie pamięci](https://pl.wikipedia.org/wiki/Od%C5%9Bmiecanie_pami%C4%99ci" \o "Odśmiecanie pamięci) czy obsługę [wyjątków](https://pl.wikipedia.org/wiki/Wyj%C4%85tek) oraz [bibliotekę standardową](https://pl.wikipedia.org/wiki/Biblioteka_programistyczna). W zależności od potrzeb i liczby dostępnych narzędzi, wyróżniane są dwie główne dystrybucje: JRE i JDK

Elementy składowe:

W skład maszyny wirtualnej Java wchodzą następujące elementy:

* [**Interpreter**](https://pl.wikipedia.org/wiki/Interpreter_(program_komputerowy)) – wykonuje krok po kroku instrukcje programu zapisane w postaci kodu bajtowego,
* [**kompilator**](https://pl.wikipedia.org/wiki/Kompilator)[**JIT**](https://pl.wikipedia.org/wiki/JIT_(informatyka)) – opcjonalny komponent wchodzący w skład części implementacji, który kompiluje najczęściej wykonywane fragmenty kodu do postaci [kodu maszynowego](https://pl.wikipedia.org/wiki/J%C4%99zyk_maszynowy), dzięki czemu mogą być one wykonywane bezpośrednio przez procesor komputera. Pozwala na zwiększenie wydajności,
* **Zarządca pamięci** – zarządza [stertą](https://pl.wikipedia.org/wiki/Sterta_(informatyka)), na której przechowywane wszystkie [obiekty](https://pl.wikipedia.org/wiki/Obiekt_(programowanie_obiektowe)) wykonywanej aplikacji oraz zapewnia automatyczne zwalnianie nieużywanej pamięci,
* **Weryfikator kodu bajtowego** – kluczowym dla bezpieczeństwa aspektem jest weryfikacja kodu bajtowego przed jego uruchomieniem, której celem jest sprawdzenie poprawności wszystkich odwołań oraz upewnienie się, że wykonanie danego fragmentu nie zaszkodzi stabilności lub bezpieczeństwu systemu, na którym uruchamiana jest maszyna wirtualna. Zajmuje się tym weryfikator kodu bajtowego.
* **Java API** – zestaw bibliotek programistycznych udostępniających takie usługi, jak obsługę plików czy [GUI](https://pl.wikipedia.org/wiki/Graficzny_interfejs_u%C5%BCytkownika), z których korzystają wykonywane aplikacje. Większość biblioteki standardowej napisana jest w języku Java, dlatego maszyny wirtualne nie muszą dostarczać własnej implementacji.

Na JVM działają : Grooove, Scala, Kotlin, JRuby

## 2. Wzorce projektowe.

Sposoby pisania programu, aby był on przejrzysty i bardziej zrozumiały.

Nie są to gotowe rozwiązania. Trzeba je zaadaptować do swoich programów.

### a)AdapterPattern

Czyli gdy mamy dwa interfejsy : old i new i chodzi o ty by zapewnić to, by osoby korzystające ze starszego interfejsy mogły nadal to robić w sposób przezroczysty mimo, że już powstał nowy interfejs.

**public class**NoteAdapter**implements**OldInterface {  
  
**public** List<String>**noteList**;  
**public** NewNote**newNote**;  
  
  
**public** NoteAdapter(){  
**noteList**= **new**ArrayList<>();  
**newNote**= **new**NewNote();  
 }  
  
@Override  
**public void**addNote(String note) {  
**noteList**.add(note);  
**newNote**.addNote(note, **null**);  
 }

**public void**addNote(String note, String toWho){  
**noteList**.add(note+**" "**+toWho);  
**newNote**.addNote(note, toWho);  
}

**main**

NoteAdapternote = **new**NoteAdapter();  
note.addNote(**"Dziś jest piękny dzień, zajmujemy się wzorcami :)"**);  
note.addNote(**"Odebrać dzieciaka z przedszkola!"**);

### b) Builder Pattern

To sposób na rozwiązanie zbyt długiego wywołania konstruktora podczas tworzenie obiektu.

Przykładowo mamy klaseCat i wewnątrz niej klasę statyczną

*// Klasa wewnętrzna statyczna Builder***public staticclass**Builder{  
 String **name**;  
 String **sex**;  
**intage**;  
**intweight**;  
  
**public** Builder() {  
 }  
  
  
**public** Builder name(String name) {  
**this**.**name** = name;  
**return this**;  
 }  
  
**public** Builder sex(String sex){  
**this**.**sex**=sex;  
**return this**;  
 }  
  
**public** Builder age(**int**age){  
**this**.**age**=age;  
**return this**;  
 }  
  
**public** Builder weight(**int**weight){  
**this**.**weight**=weight;  
**return this**;  
 }  
  
**public** Catbuild(){  
**return new**Cat(**this**);  
 }  
}  
*// koniec klasy Builder*

main

i uruchamiamy np. tak

Catcat=Cat.*builder*()  
 .name(**"Kropka"**)  
 .age(15)  
 .sex(**"plec"**)  
 .weight(12)  
 .build();  
System.***out***.println(cat);

### c) Singleton

Kiedy potrzebujemy tylko jednej instancji obiektu, np. połączenie mysql.

**public class**MySqlConnector {  
**privatestatic**MySqlConnector*instance*;  
**private**MySqlConnector(){  
  
 }  
**public static**MySqlConnectorgetInstance(){  
**if**(*instance*==**null**){  
*instance*=**new**MySqlConnector();  
 }  
**return** *instance*;  
 }  
}

## 3. GarbageCollector

Komponent JVM odpowiedzialny za czyszczenie pamięci.

## 4.Klasa String

Obiekt typu immutable.

## 5.Wątek i proces

Wątek jest podzbiorem procesu. Jeden proces może zawierać wiele wątków.

## 6.Na czym polega debugowanie?

Usuwanie błędów. Wstrzymane działania programu aby odczytać wartości zmiennych.

## 7. Jakie są rodzaje kolekcji?

List, Set, Queue

## 8. Co to jest TDD

Test Driven Development (metodyki zwinne, kiedyś programowanie ekstremalne)

1. Najpierw programista pisze automatyczny test sprawdzający dodawaną funkcjonalność. Test w tym momencie nie powinien się udać.
2. Później następuje implementacja funkcjonalności. W tym momencie wcześniej napisany test powinien się udać.
3. W ostatnim kroku programista dokonuje [refaktoryzacji](https://pl.wikipedia.org/wiki/Refaktoryzacja" \o "Refaktoryzacja) napisanego kodu, żeby spełniał on oczekiwane standardy.

## 9. Final, finally, finalize

final zmienna finalna

finally wyjątki

finalize metoda

## 10.Co to jest czysty kod

Napisane testy, docs, wzorce projektowe, zgodne z SOLID

## 11.Profiler

Program używany do monitorowania aplikacji.

Szybkość akcji, wykorzystanie pamięci

Np. JProfiler

# Spring

## 1.Co to jest IoC (inversion of control)

Odwrócenie sterowanie. Jest to zmiana odpowiedzialności pewnej części aplikacji

„Nie dzwoń do nas, my zadzwonimy do Ciebie”

Zewnętrzne biblioteki, frameworki wywołują w odpowiednich momentach kod programisty.

## 2. Co to jest WebService

Usługa zewnętrzna z której mogą korzystać programiści

REST i SOAP

## 3.Korzyści używania Springa

Lekki framework.

Liczy około kilkadziesiąt MB.

Pozwala na korzystanie z IoC.

Pozwala programować aspektowo

**Programowanie aspektowe** (aspect-orientedprogramming, AOP) to paradygmat tworzenia programów komputerowych wspomagający separację zagadnień i rozdzielenie programu na części w jak największym stopniu niezwiązane funkcjonalnie.

Zarządza cyklem życia aplikacji.

Pozwala na korzystanie z MVC . Struts przeżytek

## 4.Co to jest EJB

Działa po stronie serwera. Technologia umozliwia pisanie cześć backend aplikacji Umożliwia transakcyjność, trwałość (persistent) aplikacji, rozproszenia, bezpieczeństwa.

# SQL

## 1.Co to jest JDBC ?

– łącze do bazy danych opracowane przez firmę Sun. Umożliwia komunikację klasom Javy z bazą danych przy użyciu języka SQL.

## 2. LEFT OUTER JOIN

Złączenie lewostronne zewnętrzne

# Hibernate

## 1.Co to jest ORM?

ORM – Object RelationalMapping, mapowanie obiektowo-relacyjne.

Pozwalają na odzwierciedlenie obiektów na tabele w relacyjnej bazie danych. Komunikujemy się obiektowo, a nie przez sql. Najpopularniejszy używany w Javie ORM to Hibernate.

# Zasady

## 1.Co to jest SOLID?

* **Zasada pojedynczej odpowiedzialności** (ang. ***Single-ResponsibilityPrinciple*** *– SRP*),

**Jedna klasa to jedna odpowiedzialność.**

Żadna klasa nie może być modyfikowana z więcej niż jednego powodu”.

* **Zasada otwarte – zamknięte** (ang. *Open/ClosedPrinciple – OCP*),

„Składniki oprogramowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji”.

* 1. **„otwarty na rozbudowę”?** Mianowicie to, że musi istnieć stosunkowo prosty sposób rozbudowy zachowań takiego modułu.
  2. **„Zamknięte dla modyfikacji”** oznacza z kolei, że rozbudowa modułu nie może być przeprowadzona w sposób, który spowoduje zmianę istniejącego kodu źródłowego.\
* **Zasada podstawiania Liskov** (ang. *LiskovSubstitutionPrinciple – LSP*),

„Musi istnieć możliwość zastępowania typów bazowych ich podtypami”***.***

Załóżmy, że posiadamy dwie klasy, w których znajdujemy wiele wspólnych składowych. Wspólnymi składowymi mogą być: zbiór właściwości, metod itd. Co w takiej sytuacji należałoby zrobić? Zgodnie z zasadą podstawiania Liskov, wypadałoby wyodrębnić wspólne elementy obu klas do jednej klasy – najlepiej abstrakcyjnej.

* **Zasada segregacji interfejsów** (ang. *Interface SegregationPrinciple – ISP*),

Zasada segregacji interfejsów ma za zadanie przede wszystkich wyeliminowanie nieporęcznych, niepotrzebnie rozbudowanych interfejsów.

Załóżmy, że posiadamy pewien interfejs oraz dwie klasy, które go implementują. Klasa Aimplementuje nasz przykładowy interfejs w całości – wszystkie funkcje, natomiast klasa B posiada pełną implementację tylko części z nich, pozostałe, które są niepotrzebne pozostają puste lub zwracają domyśle wartości. Jeśli mamy do czynienia z taką sytuacją, powinniśmy rozważyć rozdzielenie interfejsu na mniejsze tak, aby każdy z nich deklarował tylko te funkcje, które rzeczywiście są wywoływane przez danego klienta lub grupę klientów (klasa/grupy klas implementujące dany interfejs).

* **Zasada odwracania zależności** (ang. *DependencyInversionPrinciple – DIP*).

1. *Moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. Obie grupy modułów powinny zależeć od abstrakcji.*
2. *Abstrakcje nie powinny zależeć od szczegółowych rozwiązań. To szczegółowe rozwiązania powinny zależeć od abstrakcji.*

1)Moduły wysokiego poziomu z natury rzeczy zawierają ważne decyzje strategiczne i modele biznesowe danej aplikacji. Myślę, że wszyscy się ze mną zgodzą, iż te właśnie moduły w największym stopniu odpowiadają za funkcjonowanie aplikacji. Gdyby okazało się, iż zależą one od modułów niskiego poziomu, to zmiany elementów niskiego poziomu mogłyby mieć wpływ na funkcjonowanie modułów wysokopoziomowych, a co za tym idzie wymuszałyby zmiany na wyższych poziomach. Dodatkowo, gdy moduły wysokopoziomowe zależą od niskopoziomowych, ponowne ich wykorzystanie staje się niezwykle trudne. Jeśli jednak odwrócimy tę zależność w drugą stronę, to bardzo łatwo będzie można wielokrotnie je wykorzystywać.

2)Jeśli chodzi o zależność od abstrakcji, można to zamknąć w jednej prostej formule: *pisany przez nas kod nie powinien być uzależniony od konkretnej klasy, zależności takie powinny kończyć się na klasach abstrakcyjnych bądź interfejsach*. Zasadę zależności od abstrakcji (ang. *depend on abstractions)* można również streścić w trzech prostych punktach:

* Żadna zmienna nie powinna zawierać referencji do konkretnej klasy.
* Żadna klasa nie powinna dziedziczyć po konkretnej klasie.
* Żadna metoda nie powinna przykrywać metody zaimplementowanej w którejkolwiek z klas bazowych.

# Narzędzia

## 1.Różnice SVN i git

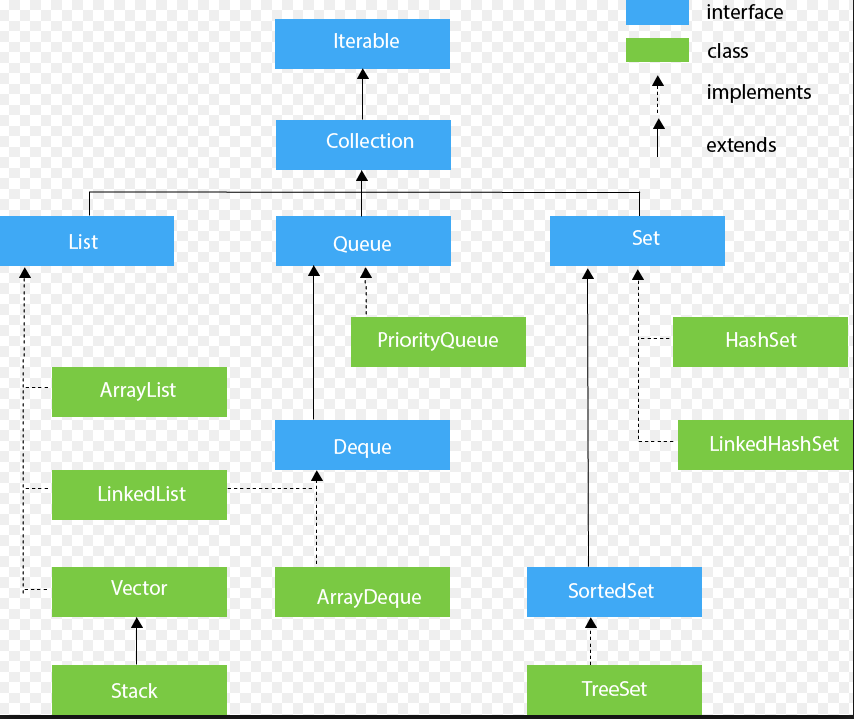
Kontrola wersji

A różnice

**Git:** szybszy, repo są mniejsze, bo trzymają wskaźniki do blobów a nie kopiowanie, system rozproszony

SVN – istnieje jedno centralne repo

## Kolekcje



## Wyrażenia regularne

Pattern.*matches*(**".{3}[0-9]{3}"**,**"abc345"**)); *//Czy pierwsze trzy znaki są dowolne, zaś następne 3 znaki tylko cyframi?*Pattern.*matches*(**"[0-9]+"**,**"545"**)); *//Czy string zawiera tylko cyfry?*Pattern.*matches*(**"[a-zA-z]+"**,**"a"**)); *//Czy string zawiera tylko litery?*Pattern.*matches*(**"\\w+"**,**"3443dfdf"**));*//Czy string zawiera tylko cyfry i litery?*Pattern.*matches*(**"[0-9]{5}"**,**"12345"**));*//Czy string zawiera tylko cyfry i ma dokładnie 5 znaków?*Pattern.*matches*(**"J.\*"**,**"J"**));*//Czy string rozpoczyna się od "J"?*Pattern.*matches*(**"J.\*U"**,**"J5UUU"**));*//Czy string rozpoczyna się od "J" i kończy na "U"?*Pattern.*matches*(**"[A-Z].\*"**,**"Ala"**));*//Czy string rozpoczyna się od wielkiej litery?*Pattern.*matches*(**".\*\\p{Space}.\*"**,**"g g"**));*//Czy string zawiera spację? (biały znak)?*Pattern.*matches*(**"[7-9][0-9]{8}"**,**"777777777"**));*//Czy string jest numerem telefonu zaczynającym się od 7 lub 8 lub 9?*Pattern.*matches*(**"[a-zA-Z]{2}([0-9]{4}){4}"**,**"ab1234123412341234"**));*//Czy string jest numerem bankowym? (Zakładając, że zaczyna się od dwóch liter, następnie 4 grupy po 4 cyfry)*Pattern.*matches*(**"[0-9]{2}-[0-9]{3}"**,**"20-533"**));*//Czy string jest kodem pocztowym?*

## REST vs SOAP

Rest **RE**presentational **S**tate **T**ransfer

SOAP **S**imple **O**bject **A**ccess **P**rotocol

Nie jest to dobre pytanie, bo SOAP to protokół a REST nie.

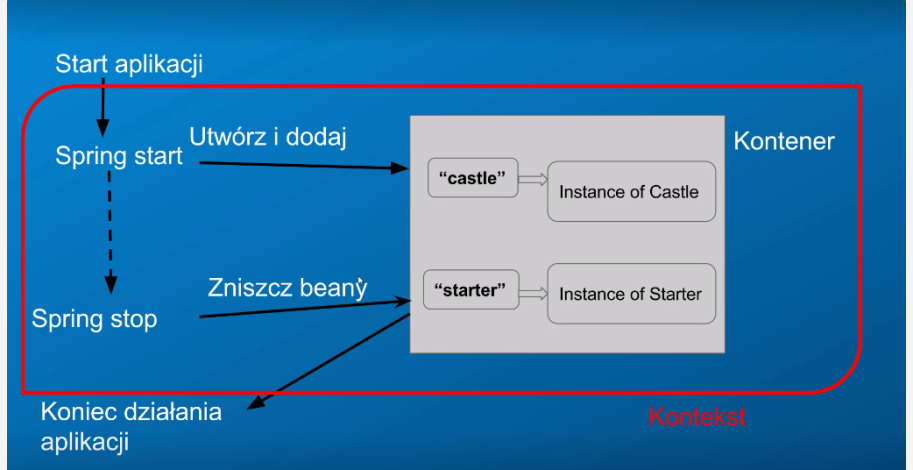
Rest to styl architektury i wzorzec budowy aplikacji.

So I am trying to say that, **sending data over the network in JSON format is cheaper than sending it in XML format regarding payload**

**Here is the first benefit or advantages of REST over SOAP**. SOAP only support XML, but REST supports different format like text, JSON, XML, etc. And we already know, if we use Json then definitely we will be in better place regarding payload.

## Spring

Spring startuje. Skanuje pliki źródłowe. Szuka np. adnotacji @Component. Utworzy obiekt tej klasy i doda do kontenera. A klasa jest dodawana do kontekstu.



**Kontekst**

Odpala metodę run na klasach implementujących CommandLineRunner

**Kontener**

to zwykła mapa, gdzie jako klucz mamy nazwę naszego beana czyli naszej klasy, a jako wartość mamy instancję naszej klasy

W momencie zakończenia aplikacji beany z kontenera są usuwane.

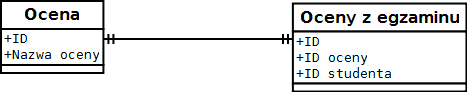
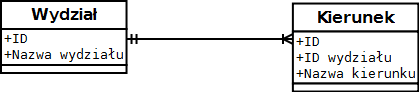
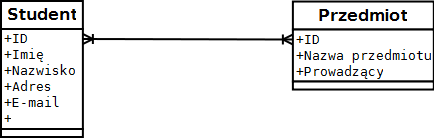
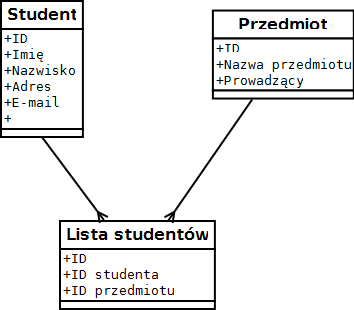
**Cykl życia Spring Beans**

Kontekst tworzy beany i je usuwa – to cykl ich życia.

## Relacje ERD w bazach danych

## Relacje pomiędzy tabelami

Rozróżnia się trzy relacje, jakie mogą zachodzić pomiędzy dwoma encjami:

1. Relacja jeden-do-jeden oznacza, iż jeden rekord w pierwszej tabeli odpowiada dokładnie jednemu rekordowi w tabeli drugiej. W praktyce relację jeden-do-jeden stosuje się w przypadku tzw. tabel słownikowych (rysunek 4).  
     
     
   **Rysunek 4. Relacja jeden-do-jeden.**  
     
   Relacja ta ułatwia późniejszą modyfikację np. nazwy oceny tylko w jednym miejscu, zamiast przeglądania i zamieniania całości rekordów w encji  Oceny\_z\_egzaminu. W klasycznej notacji ERD relację tę zapisuje się za pomocą symbolu przedstawionego na rysunku 5.  
     
   https://i-msdn.sec.s-msft.com/dynimg/IC495542.png  
     
   **Rysunek 5. Symbol relacji jeden-do-jeden.**
2. Relacja jeden-do-wielu oznacza, iż jeden rekord w pierwszej tabeli odpowiada wielu rekordom w tabeli drugiej i jest to relacja najczęściej stosowana. Jako przykład można użyć strukturę Wydziału składającą się z wielu kierunków (rysunek 6).  
     
     
   **Rysunek 6. Relacja jeden-do-wielu.**  
     
   Na diagramach ERD relacja jeden-do-wielu prezentowana jest za pomocą symbolu przedstawionego na rysunku 7.  
     
   https://i-msdn.sec.s-msft.com/dynimg/IC495543.png  
     
   **Rysunek 7. Symbol relacji jeden-do-wielu.**
3. Relacja wiele-do-wielu oznacza, że kilka rekordów z pierwszej tabeli odpowiada wielu rekordom z tabeli drugiej. W praktyce stosowanie tejże relacji jest wielce niewskazane ze względu na redundancję danych, zatem rozwiązaniem tego problemu będzie utworzenie tzw. tabeli łącznikowej, scalającej obie tabele relacją jeden-do-wielu. Przykładowa relacja wiele-do-wielu została zaprezentowana na rysunku 8, natomiast prawidłowa prezentacja – za pomocą tabeli łącznikowej na rysunku 9.  
     
     
     
   **Rysunek 8. Relacja wiele-do-wielu.**  
     
     
     
     
   **Rysunek 9. Relacja wiele-do-wielu zrealizowana za pomocą tabeli łącznikowej.**  
     
   Relacja wiele-do-wielu prezentowana jest za pomocą symbolu przedstawionego na rysunku 10.  
     
   https://i-msdn.sec.s-msft.com/dynimg/IC495553.png  
     
   **Rysunek 10. Symbol relacji wiele-do-wielu.**  
     
   Do opisu powyższych relacji można dodatkowo wprowadzić parametry charakteryzujące opcjonalne bądź obligatoryjne wykorzystanie danego atrybutu. Opcjonalność zapisuje się, dodając puste koło obok symbolu relacji (rysunek 11).  
     
   https://i-msdn.sec.s-msft.com/dynimg/IC495546.png

//TODO sposoby tworzenie wątków

//TODO programowanie funkcjonalne

//TODO JDBC kwerenda

## JPA

### Relacje w JPA

@Entity

public class StronaJeden {

@Id

Long id;

@OneToMany(mappedBy="stronaJeden")

List stronyWiele;

}

@Entity

public class StronaWiele {

@Id

Long id;

@ManyToOne

@JoinColumn("strona\_jeden\_id")

StronaJeden stronaJeden;

}

Po stronie ‘jeden’ relacji mamy do wyboru dwa rodzaje kolekcji — Set, który jest nieuporządkowany oraz List, która jest uporządkowana. W przypadku listy, możemy wskazać dodatkowy parametr — nazwę pola — na podstawie którego ma być sortowana (parametr orderBy w adnotacji @OneToMany), dzięki czemu móżemy sortować np. na podstawie nazwy albo jakiegoś innego atrybutu

### FetchType — ładowanie kolekcji

Obie te adnotacje (@OneToMany oraz @ManyToOne) mają jeszcz jedną opcję — fetch . Jest to bardzo ważny element, który domyślnie ma wartość LAZY dla @OneToMany oraz EAGER dla @ManyToOne.  
Parametr ten decyduje, kiedy obiekt/obiekty w tym polu powinny zostać pobrane:

**EAGER** — pobierz w momencie wykonywania zapytania

**LAZY** — pobierz dopiero kiedy będzie pierwsze odwołanie do tego obiektu

Każda z opcji ma swoje wady i zalety. Należy bardzo uważać z EAGER, ponieważ możemy przez przypadek spowodowac pobranie całej bazy danych w jednym zapytaniu. W przypadku LAZY, pobieranie ‘na żądanie’ zadziała tylko, jeżeli jesteśmy w ramach transakcji (metody z adnotacją @Transactional, a encja jest zarządzana — najczęściej jest to po prostu miejsce, w którym pobieramy tą encję z bazy danych). Nalezy więc bardzo dokładnie przemyśleć jak będziemy korzystać z danych oraz jaki ma to wpływ na ilosć danych, które pobieramy. Najczęściej domyślne ustawienia są w zupełności wystarczające

//TODO Hibernate

//TODO JUnit

//TODO SpringBoot

//TODO SpringData itd.

//TODO Maven

//TODO Git

//TODO Docker

//TODO SQL kwerendy

//TODO Access modifier

//TODO kolekcje - czasy

## nowości w Java 8 – marzec 2014 r.

* forEach() method in Iterable interface.
* default and static methods in Interfaces.
* Functional Interfaces and **Lambda Expressions**.
* Java Stream API for Bulk Data Operations on Collections.
* **Java Time API**.
* **Collection API improvements**.
* **Concurrency API improvements**.
* Java IO improvements.

## nowości w Java 9 - 27 lipca 2017 r.

https://techpolska.pl/java-9-zmiany/

### The Java Platform module system

### Linking

### JShell: the interactive Java REPL

### Improved Javadoc

### Collection factory methods

Set<Integer> ints = Set.of(1, 2, 3);

List<String> strings = List.of("first", "second");

### Stream API improvements

There are four new methods added to the Stream interface:

***dropWhile, takeWhile, ofNullable***

IntStream.iterate(1, i -> i < 100, i -> i + 1).forEach(System.out::println);

it's now possible to turn an Optional object into a (possibly empty) Stream with the new *`stream`* method on Optional:

Stream<Integer> s = Optional.of(1).stream();

### Private interface methods

Metody domyślne w interface mogą mieć już zachowanie

### Obsługa http 2.0

Nie trzeba już używac HttpURLConnection

### Multi-release JARs

## nowości w Java 10 – 20 marca 2018 r.

### Rozszerzone wnioskowanie typów i var

Inferencja typów została rozszerzona na deklaracje lokalnych zmiennych. Dzięki temu nie trzeba będzie długo rozpisywać się z deklaracjami zmiennych lokalnych.

### Garbage Collector

Domyślny w Javie 9 odśmiecacz pamięci lepiej znany jako garbage collector stosuje jednowątkowy algorytm typu mark-sweep-compact, kiedy nie potrafi odzyskać pamięci wystarczająco szybko, powraca do starego równoległego GC. Aby ograniczyć najgorsze opóźnienia, udało się nowy Garbage Collector też uczynić równoległym.

### Application Class-Data Sharing

Mechanizm Class-Data Sharing został rozwinięty o przechowywanie klas aplikacji w współdzielonym archiwum. Zmniejsza to rozmiar procesów Javy w pamięci, ułatwia wykorzystanie zarchwizowanych klas i przyśpiesza uruchamianie procesów.

### Thread-Local Handshakes

Nowy sposób wykonania wywołania zwrotnego w wątkach, który nie potrzebuje wywoływania safepoint’a wirtualnej maszyny. Pozwala to na pauzowanie indywidualnych wątków, a nie od razu wszystkich lub żadnego.

### Root Certificates

W JDK pojawia się domyślny zestaw certyfikatów root Urzędów Certyfikacji (CA). Ich otworzenie ma uczynić otwarte wersje OpenJDK bardziej atrakcyjnymi dla deweloperów i zmniejszyć różnice między nimi, a kompilacjami Oracle.

### Uogólniony switch

Do switcha będzie można zasadniczo przekazać niemal cokolwiek, a następnie w blokach case nastąpi odpowiednie dopasowanie do wzorca. Docelowo ma tutaj także działać coś co nazywa się dekonstrukcją obiektów

## nowości w Java 11 – wrzesień 2018 r.

### JAVA FX 11 oddzielnie

a wraz z pojawieniem się 11 wersji Javy zdecydowano, że nie będzie ona już częścią JDK, zamiast tego będzie dostępna do pobrania niezależnie.

Java FX 11 wprowadza też kilka nowości do samej biblioteki. Najciekawsza to chyba wprowadzenie publicznego FX Robot API, które jest odpowiednikiem API Robot z pakietu AWT. Pozwala ono na symulowanie interakcji użytkownika – wciskania klawiszy, klikania przycisków myszy, przesuwania kursora, czy robienia zrzutów ekranu. Wszystko to może być przydatne np. do napisania prostego bota do gry.

### Skrypty

Mając napisaną klasę możemy ją uruchomić korzystając bezpośrednio z polecenia java.

### Nashorn Javascript Engine depreacted

### Czyszczenia API ciąg dalszy

### Drobne usprawnienia API

* **Metody związane z białymi znakami – *isBlank(), strip() , stripLeading() , stripTrailing()***
* **metoda *repeat(int repeats)***
* **metoda *lines() zwracająca strumień z wierszami tekstu***

### var w wyrażeniach lambda

### inne

Wprowadzono także dwa dodatkowe garbage collectory. Pierwszy o nazwie Epsilon jest bardzo nietypowy. Jego zadaniem jest nie odśmiecanie pamięci, co może wydawać się absurdem, ale może to być bardzo przydatne do celów testowych. Drugi o nazwie ZGC ma być bardzo wydajny w wielogigabajtowych scenariuszach.

//TODO biblioteki Guava