PYTANIA NA ROZMOWACH KWALIFIKACYJNYCH

Autor: Mariusz Czarny

18 PAŹDZIERNIKA 2024

Spis treści

ALGORYTMY 2
PARADYGMATY PROGRAMOWANIA OBIEKTOWEGO 5
SQL 6
SIECI 16
SECURITY 17
JAVA 18
TESTOWANIE 27
WZORCE PROJEKTOWE 29
HIBERNATE 33
SPRING 36

Algorytmy

„Komputer zasługiwałby na miano inteligentnego, gdyby mógł oszukać człowieka, tak by uwierzył, że też jest człowiekiem."

Zdefiniuj pojęcie złożoności obliczeniowej?

Jest to liczba zasobów niezbędnych do wykonania algorytmu.

Jakie rodzaje złożoności wyróżniamy?

Czasowa (liczbę operacji podstawowych w zależności od danych wejściowych) i pamięciowa (miarą ilości wykorzystanej pamięci).

W jaki sposób matematycznie określa się złożoność?

Za pomocą notacji dużego *O*.

Jakie są rzędy złożoności?

O(1) – złożoność stałą

O(n) – złożoność liniowo

O(log(n) – złożoność logarytmiczną

O(n^2) – złożoność kwadratowa

O(n^x) – złożoność wielomianowa

O(x^n) – złożoność wykładnicza

Opowiedz o problemie producenta i konsumenta oraz zaproponuj jego rozwiązanie?

Jeden lub wiele procesów (producent) generuje dane a inny (konsument) je pobiera. Aby rozwiązać ten problem stosuje się wspólny bufor, który cyklicznie zapełnia się i zwalnia tak, aby uniknąć momentu całkowitego zapełnienia lub wyczyszczenia. Praktyczną realizacją tego rozwiązania jest struktura danych o nazwie kolejka.

Napisz jeden z poniższych algorytmów.

Silnia (rekurencyjnie):

```
public static int getFactorial(int number) {
  if (number <= 1) {
     return 1;
  }
  else {
     return number * getFactorial(number - 1);
  }
}
```

Fibonacci:

```
public static int fibonacci(int number) {
    if (n == 0)
        return 0;

    if (n == 1 || n == 2)
        return 1;

    else
        return (fibonacci (number - 1) + fibonacci (number - 2));
}
```

Sortowanie bąbelkowe (ang. bubblesort):

```
private static void sort(int array[]) {
    int n = array.length;

    for (int i = 0; i < n - 1; i++) {
       for (int j = 0; j < n - i - 1; j++) {
           if (array[j] > array[j + 1]) {
                swapArray(array, j);
           }
       }
    }
}
```

```java
private static void swapArray(int[] array, int j) {
    int temp = array[j];
    array[j] = array[j + 1];
    array[j + 1] = temp;
}
```

Sortowanie szybkie (ang. quicksort):

```java
private static void sort(int[] array, int left, int right) {
    int i = left;
    int j = right;

    int pivot = array[left + (right - left) / 2];

    while (i <= j) {
      while (array[i] < pivot) {
        i++;
      }

      while (array[j] > pivot) {
        j--;
      }

      if (i <= j) {
        swapArray(array, i, j);
        i++;
        j--;
      }
    }

    if (left < j) {
      sort(array, left, j);
    }

    if (i < right) {
      sort(array, i, right);
    }
}

private static void swapArray(int[] array, int i, int j) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

Paradygmaty programowania obiektowego

*„Każdy głupiec może napisać kod zrozumiały dla komputera.
Dobrzy programiści piszą kod, który zrozumiały dla innych ludzi."*

Wymień wszystkie paradygmaty programowania obiektowego?

a) Abstrakcja:

Każdy obiekt w systemie służy jako model abstrakcyjnego "wykonawcy", który może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez ujawniania, w jaki sposób zaimplementowano dane cechy. Procesy, funkcje lub metody mogą być również abstrahowane, a kiedy tak się dzieje, konieczne są rozmaite techniki rozszerzania abstrakcji.

b) Hermetyzacja:

Czyli ukrywanie implementacji, enkapsulacja. Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko własne metody obiektu są uprawnione do zmiany jego stanu. Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy. Pewne języki osłabiają to założenie, dopuszczając pewien poziom bezpośredniego (kontrolowanego) dostępu do "wnętrzności" obiektu. Ograniczają w ten sposób poziom abstrakcji.

c) Polimorfizm:

Referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego. Jeśli dzieje się to w czasie działania programu, to nazywa się to późnym wiązaniem lub wiązaniem dynamicznym.

d) Dziedziczenie:

Porządkuje i wspomaga polimorfizm i enkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy. W typowym przypadku powstają grupy obiektów zwane klasami, oraz grupy klas zwane drzewami. Odzwierciedlają one wspólne cechy obiektów.

SQL

„Najpierw rozwiąż problem. Potem napisz kod.”

Co robi *GROUP BY* i *HAVING* ?

SELECT .. GROUP BY .. HAVING .. - dokonuje grupowanie danych a potem ustawia warunek na wyniku grupowania. Przykładem użycia może być zliczanie danych w grupach.

Jaka jest podstawowa różnica między *WHERE* i *HAVING* ?

WHERE filtruje wiersze przed grupowaniem i obliczeniami (decyduje, które wiersze wejdą do obliczeń funkcji agregujących), podczas gdy *HAVING* selekcjonuje wiersze już pogrupowane, po wykonaniu obliczeń. Dlatego klauzula *WHERE* nie musi zawierać funkcji agregujących, ponieważ nie ma sensu użycie funkcji do zdecydowania, które wiersze wejdą do funkcji. Z drugiej strony, klauzula *HAVING* zawsze zawiera funkcje agregujące. (Mówiąc wprost, możesz zastosować klauzulę *HAVING* bez funkcji, ale jest wtedy mniej efektywna niż *WHERE* z tym samym warunkiem).

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

```
SELECT `imię`,`nazwisko`,sum(`wartosc_zamowienia`)
FROM `zamowienia`
GROUP BY `imię`,`nazwisko`
HAVING SUM(`zamowienia`)>200
```

W jaki sposób posortować wyniki zapytania?

ORDER BY - sortuje wyniki zapytania względem wybranej kolumny wg alfabetu lub wg wartości numerycznej.

```
SELECT `nazwa_kolumny`
FROM `nazwa_tabeli`
[WHERE `nazwa_kolumny`]
ORDER BY 'nazwa_kolumny' [ASC,DESC]
[, 'nazwa_kolumny2' [ASC,DESC] ]
```

Wymień przykładowe funkcje agregujące.

SUM - zwraca sumę wszystkich wartości z podanej kolumny.

AVG - wylicza średnią wartość z kolumny podanej jako argument funkcji.

MIN - zwraca najmniejszą wartość kolumny, w przypadku kolumny składającej się z łańcuchów tekstowych porównywane litery na kolejnych pozycjach wg kodu ASCII.

MAX - zwraca największą wartość kolumny.

ROUND - zaokrągla wartość z podanej kolumny zmiennoprzecinkowej do podanej jako argument liczby miejsc po przecinku.

```
SELECT ROUND(`nazwa_kolumny`, ilość_miejsc_po_przecinku)
FROM `nazwa_tabeli`
```

MID - zwraca ciąg znaków z pola `nazwa_kolumny` od pozycji start. Jeśli zostanie podany trzeci parametr to zostanie zwrócona podana ilość znaków od pozycji start.

```
SELECT MID(`nazwa_kolumny`,start[,długość])
FROM `nazwa_tabeli`
SELECT MID(`imie`,1,3) AS imie_skrót FROM `osoby`
```

COUNT - zlicza liczbę wierszy wg zadanych kryteriów jako argumenty do funkcji.

W jaki sposób uzyskać tylko unikalne rekordy?

DISTINCT - stosuje się zaraz po *SELECT* aby wybrać unikalnie kolumny. Przydaje się, gdy trzeba wyświetlić jakie występują wartości w danej kolumnie bez wyświetlania powtarzający się danych.

Czym jest widok?

Widok (ang. *view*) to tabela „wirtualna", którą tworzy się za pośrednictwem zapytania *SELECT*. Stosuje się go do bardzo rozbudowanych zapytań w celu ułatwienia ich wywoływania.

```
CREATE VIEW nazwa_widoku AS
SELECT `nazwa_kolumny1` [,`nazwa_kolumny2`,...]
FROM `nazwa_tabeli1` [,`nazwa_tabeli2`,...]
[WHERE warunek]
```

Czym jest indeks?

Indeks to nic innego jak struktura danych, która przechowuje wartości dla określonej kolumny w tabeli. Silnik wyszukiwania w bazie danych może wykorzystać go do przyspieszenia pobierania danych. Przykładem może być książka, która na końcu posiada indeks popularnych haseł z przypisanymi stronami, gdzie jest ono omawiane.

Indeks pomaga przyspieszyć zapytania *SELECT* i klauzule *WHERE*, ale spowalnia wprowadzanie danych za pomocą instrukcji *UPDATE* i *INSERT*. Indeksy można tworzyć lub usuwać bez wpływu na dane.

```
CREATE INDEX nazwa_indexu
ON `nazwa_tabeli` (`nazwa_kolumny`[,`nazwa_kolumny2`])
```

Wymień podstawowe komendy modyfikujące dane?

UPDATE - pozwala na zmianę, aktualizację wartości kolumny w tabeli.

```
UPDATE `nazwa_tabeli`
SET `nazwa_kolumny`='wartość'
[WHERE `nazwa_kolumny` operator 'wartość']
```

DELETE - pozwala na usunięcie rekordów z tabeli.

```
DELETE FROM `nazwa_tabeli`
[WHERE `nazwa_kolumny` operator 'wartość']
```

DROP - pozwala na usunięcie indexów, tablic, baz danych.

```
DROP DATABASE/TABLE/INDEX `nazwa `
```

INSERT - pozwala na wstawianie nowych wartości do tabeli.

```
INSERT INTO `osoby` (`imie`,`nazwisko`)
VALUES ('wartość','wartość2')
```

ALTER - umożliwia wykonanie operacji na strukturze istniejącej tabeli takie jak: dodawanie, usuwanie, zmiana nazwy, zmiana typu danych kolumny, a także dodawanie, usuwanie indexu dla kolumny.

```
ALTER TABLE `nazwa_tabeli`
ADD/MODIFY/DROP/CHANGE `nazwa_kolumny` typ_danych
```

Czym jest więź (relacja) między tabelami?

Więź (ang. *relationship*) to powiązanie pomiędzy parą tabel czyli relacji. Istnieje ona wtedy, gdy dwie tabele są połączone przez klucz podstawowy (*PRIMARY KEY*) i klucz obcy (*FOREIGN KEY*). Każda więź jest opisywana przez typ więzi istniejący między dwoma tabelami, typ uczestnictwa oraz stopień uczestnictwa tych tabel.

Wymień rodzaje więzi?

Więź jeden-do-jednego - występuje, gdy pojedynczemu rekordowi z tabeli A przyporządkowany jest dokładnie jeden rekord z tabeli B i na odwrót.

Więź jeden-do-wielu - występuje, gdy pojedynczemu rekordowi z tabeli A może odpowiadać jeden lub więcej rekordów z tabeli B, ale pojedynczemu rekordowi z tabeli B odpowiada najwyżej jeden rekord z tabeli A. Nie jest to więc tak wzajemna więź jak w przypadku jeden-do-jednego.

Więź wiele-do-wielu - występuje, gdy jednemu rekordowi z tabeli A przypisanych jest wiele rekordów z tabeli B oraz jednemu rekordowi z tabeli B przyporządkowanych jest wiele rekordów z tabeli A

Wymień trzy pierwsze stopnie normalizacji bazy danych.

- **I postać normalna** - zachodzi jeśli wartości atrybutów są elementarne czyli niepodzielne. Tabela w tej postaci nie może zawierać powtarzających się grup informacji. Każda kolumna jest wartością skalarną a nie macierzą, listą lub jakąkolwiek złożoną strukturą danych.

- **II postać normalna** - relacja (tabela) jest w II postaci jeśli :
  - jest w I postaci normalnej
  - jeśli każdy atrybut tej relacji nie wchodzący w skład żadnego klucza jest w pełni funkcyjnie zależny wyłącznie od podrelacji klucza głównego

- **III postać normalna** - relacja (tabela) jest w III postaci jeśli :
  - jest w II postaci normalnej
  - każdy atrybut jest funkcjonalnie zależny jedynie od klucza głównego, nie mogą więc istnieć jakiekolwiek zależności przechodnie (brak nadmiarowości, powtarzających się danych*)*

Czym jest Diagramy *ERD*?

Diagram związków encji **ERD** (ang. ***E**ntity **R**elationship **D**iagram*), który jest graficznym przedstawieniem struktur danych oraz relacji pomiędzy nimi .

Czym jest encja, atrybut i związek?

encja - reprezentacja wyobrażonego lub rzeczywistego obiektu o którym należy znać lub przechowywać informacje.

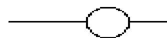
atrybut - element informacji służący do klasyfikowania, identyfikowania, kwalifikowania, określania ilości lub wyrażania stanu encji. Może być liczbą, tekstem, wartością logiczną lub obrazem.

związek - jest to powiązanie między dwiema lub kilkoma encjami. Związki na diagramie można opisać poprzez liczebność, opcjonalność i relację między encjami.

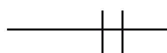
Każdy związek ma dwa końce i każdy z nich ma przypisane dwa atrybuty : nazwę i stopień związku.

Opcjonalność związku :

związek opcjonalny



związek wymagany



Jak wygląda składnia JOIN?

Składnia polecenia *SELECT* z klauzulą *JOIN* jest następująca:

```
SELECT * FROM table1 JOIN table2 ON condition ...
```

Jakie występują wersje klauzuli JOIN?

INNER JOIN - dana krotka zostanie uwzględniona wyłącznie w wypadku, gdy w drugiej tabeli występuje krotka (lub krotki), których wartość dla kolumn określonych w warunku jest taka sama. Klauzula *INNER* jest opcjonalna (tzn. klauzula *JOIN* bez modyfikatorów działa jak *INNER JOIN*).

```
SELECT * FROM `osoby`
INNER JOIN `zamowienia`
ON `zamowienia`.`id_osoby`=`osoby`.`id`
```

LEFT JOIN - dana krotka zostanie uwzględniona w wyniku, nawet jeżeli w drugiej tabeli nie będzie krotek, które mogłyby być z nią połączone (dla których były spełniony warunek *ON…*).

```
SELECT *
FROM `osoby`
LEFT JOIN `zamowienia`
ON `zamowienia`.`id_osoby`=`osoby`.`id`
```

RIGHT JOIN - uwzględniane są krotki z drugiej tabeli, które nie posiadają odpowiedników wśród krotek tabeli pierwszej.

```
SELECT *
FROM `osoby`
RIGHT JOIN `zamowienia`
ON `zamowienia`.`id_osoby`=`osoby`.`id`
```

Różnica pomiędzy *UNION* a *JOIN*?

UNION łączy rezultaty dwóch zapytań w jedną tabelę. Zdublowane rekordy są automatycznie usuwane.

Jakie są cechy kluczy w SQL?

- ✓ Klucz jest podstawowym zbiorem identyfikującym.
- ✓ Taki zbiór atrybutów relacji, których kombinacje wartości jednoznacznie identyfikują każdą, krotkę tej relacji a żaden podzbiór tego zbioru nie posiada tej własności.
- ✓ W kluczu nie może zawiera się wartość *NULL*.
- ✓ Służy do identyfikowania i/lub sortowania danych.

Jakie występują rodzaje kluczy?

Klucze proste i złożone - jeśli zbiór tworzący klucz jest jednoelementowy to mówi się wtedy o **kluczu prostym**, w przeciwnym przypadku jest to **klucz złożony**.

Klucz unikalny (ang. *unique key*) - jest to wartość, która w obrębie jednej tabeli jest niepowtarzalna, tak wiec nie mogą pojawia się dwa wiersze o tej samej wartości unikalnej. Np. PESEL lub nr ISBN dla książki. Dla kolumny można wymusić to, by była ona unikalna poprzez ustalenie indeksu unikalnego UNIQUE.

Klucz podstawowy, **klucz główny** (ang. *primary key*) - jest to klucz wybrany spośród kluczy kandydujących i służy do jednoznacznej identyfikacji każdego wiersza w tabeli. Nie może zawierać wartości NULL. Takim kluczem może być np. nr albumu studenta lub nr PESEL. Istnieje także możliwość utworzenia sztucznego klucza, który będzie nadawany automatycznie poprzez system bazy danych.

Klucz obcy (ang. *foreign key*) - ten typ klucza wykorzystywany jest do tworzenia relacji pomiędzy parą tabel. Wartość klucza obcego w jednej tabeli (**tabeli zależnej**) musi korespondować do wartości klucza podstawowego lub kandydującego w drugiej tabeli (**tabeli nadrzędnej**).

Wymień rodzaje integralności więzi?

Klucz obcy wymusza więzi integralności (ang. *referances*). Przy ustalaniu zależności pomiędzy tabelami można ustalić jakie działania będą podejmowane w przypadku aktualizacji lub usuwaniu rekordu z tabeli nadrzędnej:

- **CASCADE**
  - ON UPDATE CASCADE - zmiana klucza w tabeli nadrzędnej powoduje zmianę klucza obcego w tabeli zależnej.
  - ON DELETE CASCADE - usuniecie rekordu z tabeli nadrzędnej powoduje usuniecie korespondującego rekordu z tabeli zależnej.
- **RESTRICT**
  - ON UPDATE RESTRICT - nie może zostać przeprowadzona żadna aktualizacja w rekordzie tabeli nadrzędnej, jeśli istnieje powiązany rekord w tabeli zależnej.
  - ON DELETE RESTRICT - nie można usunąć rekordu z tabeli nadrzędnej jeśli istnieje powiązany rekord w tabeli zależnej.
- **SET NULL**
  - ON UPDATE SET NULL - w przypadku zmiany klucza podstawowego w tabeli nadrzędnej wstaw wartość *NULL* dla klucza obcego w tabeli zależnej.
  - ON DELETE SET NULL - w przypadku usunięcia rekordu w tabeli nadrzędnej wstaw wartość *NULL* dla klucza obcego w tabeli zależnej.
- **SET DEFAULT**
  - ON UPDATE SET DEFAULT - w przypadku zmiany klucza podstawowego w tabeli nadrzędnej ustal wartość domyślną (*DEFAULT*) dla klucza obcego.
  - ON DELETE SET DEFAULT - w przypadku usunięcia rekordu w tabeli nadrzędnej ustal wartość domyślną (*DEFAULT*) dla klucza obcego.
- **ON UPDATE NO ACTION** - w przypadku zmiany klucza podstawowego w tabeli nadrzędnej nie zmieniaj wartości dla klucza obcego.

Czym jest PL/SQL?

Język PL/SQL jest ładowalnym językiem proceduralnym, który pozwala na :

- tworzenie procedur wyzwalanych oraz funkcji
- rozszerzenie SQL o struktury sterujące podobne jak w językach proceduralnych

Jest to język blokowo-strukturalny, podobny do języka Pascal lub C, z deklaracjami zmiennych i zakresami bloków. Każdy blok ma opcjonalną etykietę, może posiadać kilka deklaracji zmiennych i zamyka instrukcje tworzące blok pomiędzy słowami kluczowymi *BEGIN* oraz *END*.

Czym są transakcje?

Jest to zbiór operacji na bazie danych, które stanowią pewną całość. Powinny zostać wykonane wszystkie albo żadna z nich. Nie ma mowy o częściowym wykonaniu. Warunki jakie powinny spełniać transakcje są dokładniej opisane w zasadach *ACID*.

Jakie są podstawowe operacje na transakcjach?

- **BEGIN WORK** - rozpoczęcie transakcji
- **COMMIT WORK** - wszystkie elementy wchodzące w skład transakcji są kompletne i powinny zostać zatwierdzone. Od tej pory mają być dostępne dla wszystkich jednoczesnych transakcji oraz wszystkich kolejnych.
- **ROLLBACK WORK** - transakcja ma zostać porzucona, a zmiany przez nią dokonane na bazie danych mają zostać anulowane

Jakie są podstawowe warunki jakie transakcja musi spełnić?

- **ATOMIC** (niepodzielność) - transakcja mimo tego, że jest zbiorem działań musi zostać wykonana jako jedna jednostka. Musi odbywać się w jednym momencie i nie może zostać podzielona na podzbiory.
- **CONSISTENT** spójność) - system musi być spójny po zakończeniu transakcji
- **ISOLATION** (odizolowanie) - każda transakcja musi być wykonywana niezależnie od innych transakcji, które mogą być wykonywane w tym samym czasie
- **DURABILITY** (trwałość) - wykonana transakcja musi zostać utrwalona na stałe.

Jakie zjawiska niepożądane mogą wystąpić podczas korzystania z transakcji?

- **DIRTY READS (brudny odczyt)** - odczyt wewnątrz transakcji pewnych danych, które zmienionych przez inną transakcję , która nie została zatwierdzona.
- **NON-REPEATABLE READS (odczyty nie dające się powtórzyć)** - odczyt zbioru danych, który przy ponownym odczycie tych samych danych daje zupełnie inny rezultat.
- **PHANTOM READS (odczyty widmo)** - występuje, gdy jedna transakcja odczytuje lub aktualizuje tabele, a druga transakcja w tym czasie dodała nowy wiersz, który powinien zostać dodany później, problem podobny do zjawiska poprzedniego.
- **LOST DATA (utracone aktualizacje)** - do bazy danych zapisywane są dwie różne aktualizacje i druga zmiana powoduje, że pierwsza zostaje utracona.

Jakie wyróżnia się poziomy izolacji transakcji?

- **READ UNCOMMITTED** - możliwy brudny odczyt, odczyt nie dający się powtórzyć i odczyt widmo.
- **READ COMMITTED** - niemożliwy brudny odczyt, możliwy odczyt nie dający się powtórzyć i odczyt widmo.
- **REPEATABLE READ** - niemożliwy brudny odczyt i odczyt nie dający się powtórzyć, możliwy odczyt widmo.
- **SERIALIZABLE** - żaden z niepożądanych odczytów nie jest możliwy.

Sieci

„Internet staje się placem miejskim globalnej wioski jutra."

Czym jest NAT – *Network Address Translation*?

NAT to zaawansowany scenariusz routingu, który wykorzystuje adresację prywatną i publiczną (*private / global*). Pozwala ona wielu urządzeniom, najczęściej połączonym za pomocą sieci lokalnej, korzystać z jednego, publicznego adresu IP.

Czym jest translacja statyczna i dynamiczna?

Kiedy mówimy o **NAT** (*Network Address Translation*) to musimy wiedzieć, że translacje NAT mogą być dokonywane statycznie (dokonywane ręcznie) lub dynamicznie (IP jest zależny od różnorodnych warunków działania i może być kompletnie inny dla każdej pojedynczej sesji).

Czym jest model OSI (*Open Systems Interconnection - OSI Reference Model*) ?

Model referencyjny **OSI**, służy do projektowania i budowy dowolnych sieci telekomunikacyjnych. Open oznacza, iż dwa dowolne systemy spełniające warunki standardu ISO mogą wzajemnie się komunikować.

Czym jest *Three Way Handshake*?

Jest to proces za pomocą którego zostaje zestawiona sesja pomiędzy urządzeniami sieciowymi.

Podaj najczęściej używane porty odpowiednich protokołów?

DNS – 53
TELNET – 23
FTP (przesyłanie poleceń) – 21
FTP (przesyłanie danych) – 20
SSH – 22
HTTP – 80, 8080
HTTPS – 443

Security

„Hakowanie to po prostu wykorzystywanie luk bezpieczeństwa w aspekcie technicznym, fizycznym lub ludzkim."

Wyjaśnij atak *Session Hijacking*?

Atak *Session Hijacking* polega na wykorzystaniu mechanizmu kontroli sesji, którym zwykle zarządza się poprzez specjalny token. Ponieważ komunikacja http wykorzystuje wiele różnych połączeń TCP, serwer WWW potrzebuje metody rozpoznawania połączeń każdego użytkownika. Można tego dokonać używając tokena, który serwer aplikacji wysyła do przeglądarki klienta po pomyślnym jego uwierzytelnieniu. *Session Hijacking* to strategia włamywania się poprzez kradzież lub przewidywanie prawidłowego tokenu sesji w celu uzyskania nieautoryzowanego dostępu do serwera.

Czym jest *Cross Site Scripting (XSS)*?

Atak typu Cross-Site Scripting (XSS) polega na wykrzykiwaniu złośliwych skryptów do innych, zaufanych witryn internetowych. Ataki XSS występują, gdy atakujący używa aplikacji internetowej do wysłania złośliwego kodu, zazwyczaj w postaci skryptu po stronie przeglądarki, do innego użytkownika końcowego. Błędy, które pozwalają na powodzenie tych ataków, są dość powszechne i występują wszędzie tam, gdzie aplikacja internetowa wykorzystuje dane wejściowe użytkownika w generowanych przez siebie danych wyjściowych bez ich walidacji lub kodowania.

Co to jest *SQL Injection*?

Jest to rodzaj ataku polegający na skorzystaniu z formatki przeznaczonej na wpisywanie danych wejściowych w aplikacji do „wykrzyknięcia" zapytania SQL. Niezabezpieczone tego typu pola, mogą umożliwiać odczytywanie lub modyfikację niepożądanych informacji z bazy.

Java

„Prawdę można znaleźć tylko w jednym miejscu: w kodzie."

Rozwiń i wyjaśnij poniższe skróty:

javac – kompilator,

jar – archiwizator,

javadoc – generator dokumentacji,

javah – generator plików nagłówkowych,

javap – deasembler,

jdb – debugger.

JRE (*Java Runtime Environment*) - środowisko uruchomieniowe Java. Pozwala na uruchamianie aplikacja zwanych apletami, które są napisane w języku programowania Java.

JDK (*Java Development Kit)* - zawiera narzędzia do tworzenia i testowania programów napisanych w języku programowania Java i działających na platformie Java.

Garbage Collector - odpowiedzialny jest za zwalnianie pamięci.

Opisz w jaki sposób Java kompiluje kod?

Efektem kompilacji jest kod pośredni, tzw. ***byte-kod***, dzięki czemu programy w JAVIE są przenośne i mogą zostać zrozumiane dla każdego rodzaju procesora.

Czy Java jest w pełni obiektowym językiem programowania?

Tak.

Czy jest klasa?

Klasa to opis obiektu.

Od czego rozpoczyna się uruchomienie kodu w Javie?

Od wykonania metody *main*, która musi być statyczna i publiczna:

```
public static void main (String args[]).
```

Jedna klasa powinna zawierać się w osobnym pliku o tej samej nazwie.

Wymień typy odnośnikowe (ang. reference types)?

- klasowe
- tablicowe
- interfejsy
- typy wyliczeniowe (*enum*)
- adnotacje

Wymień podstawowe rodzaje kolekcji w Javie?

ArrayList – sekwencja indeksowana o zmiennych rozmiarach

LinkedList – sekwencja uporządkowana umożliwiająca efektywne wstawianie i usuwanie dowolnej pozycji.

TreeSet – zbiór uporządkowany.

HashSet – kolekcja nieuporządkowana, nie dopuszcza duplikatów

EnumSet – zbiór wartości typu wyliczeniowego

LinkedHashSet – zbiór umożliwiający kolejność wstawiania elementów

HashMap – struktura przechowująca asocjacje klucz/wartość

Czym jest klasa i obiekt?

Program w Javie składa się z klas, czyli z opisów obiektów, gdzie każdy obiekt jest instancją, czyli wystąpieniem danej klasy. Klasa definiuje typ danego obiektu.

Czym jest konstruktor?

To metody, które są wywoływane automatycznie przy tworzeniu nowego obiektu. Konstruktory nie zwracają rezultatu i mają nazwę identyczną z nazwą klasy.

Jakie są rodzaje konstruktorów?

Bezargumentowy oraz z argumentami.

Czym jest przeciążanie konstruktorów?

Użycie w klasie dwóch lub więcej konstruktorów (argumentowych lub bez) o tej samej nazwie.

Co to są specyfikatory dostępu?

Określają prawa dostępu do składowych klas i do samych klas.

Wyszczególnij wszystkie specyfikatory dostępu.

publiczny (*public*) - wszyscy mają do nich dostęp, są dziedziczone.

prywatny (*private*) - można się do nich dostać tylko z wnętrza klasy.

chroniony (*protected*) - można się dostać z wnętrze klasy, klas potomnych oraz innych klas zawartych w danym pakiecie.

pakietowy - jeżeli nie występuje żadne z powyższych słów, dostęp jest w obrębie pakietu.

Jak dokonać jawnego rzutowanie (konwersji) zmiennej?

Odbywa się poprzez: *(nazwaTypu) zmienna.*

Czym jest pakiet?

To grupa klas zajmująca się zwykle jednym zagadnieniem.

Jak odwołać się do klasy z danego pakietu?

import nazwaPakietu.nazwaKlasy;

Czym jest dziedziczenie?

Oznacza, że klasa potomna przejmie właściwości klasy bazowej (czyli będzie jej rozszerzeniem).

Co robi słowo kluczowe *super* w konstruktorze?

Powoduje wywołanie konstruktora klasy bazowej i przekazanie mu wartości (jeśli to konstruktor argumentowy). Wywołanie: super(); lub super(argumenty); Ważne aby instrukcja super() była pierwszą linią konstruktora.

Czym jest przesłanianie metod?

Polega na użyciu w klasie bazowej i potomnej metody o takiej samej nazwie. W przypadku gdy należy odwołać się do przysłoniętej metody (zawartej w klasie bazowej) z poziomu klasy potomnej używa się polecenia: super.nazwaMetody();

Czym są składowe statyczne?

To pola lub metody, które mogą istnieć, nawet jeśli nie istnieje obiekt tej klasy. Każda taka metoda lub pole jest wspólne dla wszystkich obiektów tej klasy.

Czym są składowe finalne?

Czyli takie, które nie można zmieniać. Służy do tego słowo ***final***. Klasa finalna to taka, z której nie można wyprowadzać innych klas (brak pochodnych), a ich postać jest z góry ustalona. Finalne pola to takie, których wartość jest stała i nie można jej zmieniać. W polach finalnych typów prostych nie wolno zmieniać raz przypisanych wartości. W przypadku typów referencyjnych nie można zmieniać finalnej referencji, ale można zmieniać zawartość obiektu. Metoda finalna to taka, której przesłonięcie nie będzie możliwe w klasie potomnej. Finalne mogą być też argumenty, co oznacza, że nie wolno zmieniać go w ciele metody.

Czym jest słowo kluczowe *this*?

To odwołanie się do bieżącego obiektu, które można traktować jako referencje do aktualnego obiektu. Jeśli odwołanie dotyczy składowej klasy (pole, metodę) to wykorzystuje się: this.nazwa_pola/metody. Jeśli trzeba wywołać konstruktor z wnętrza innego konstruktora to także używa się słowa this: this(arg1, arg2, …). Konstruktor można wywołać jawnie tylko w innym konstruktorze i musi on być pierwszą instrukcją wykonywaną.

Czym jest metoda *finalize*?

Finalize służy do jawnego czyszczenia pamięci, za pomocą instrukcji *System.gc()*.

Do czego służy klasa *Object*?

W Javie wszystkie klasy dziedziczą pośrednio lub bezpośrednio z klasy ***Object***, która jest praklasą, z której wywodzą się inne klasy. Każda metoda dziedziczy wszystkie pola i metody klasy Object (np. *toString* która zwraca opis obiektu w formie ciągu znaków).

Opisz polimorfizm w Javie.

To inaczej wiązanie dynamiczne, czyli wiązanie obiektów, zmiennych lub metod z ich klasami na poziomie działania programu (a nie kompilacji). W przypadku rzutowania obiektu w dół uniemożliwia wykonanie niedozwolonych operacji (np. odwołanie: *punkt.x = 10;* gdy nie ma takiego pola w klasie). W przypadku rzutowania w górę (rzutowanie na klasę nadrzędną) wywołania metod są polimorficzne (czyli skojarzenie obiektu z metodą jest wykonywane w trakcie działania programu). Tym samym sprawdzany jest rzeczywisty typ obiektu, którego metoda jest wywoływana, ale dopiero w trakcie działania programu. Przy użyciu polimorficznych wywołań metod należy pamiętać, że mają one miejsce, kiedy metoda X z klasy bazowej jest przesłaniana przez metodę X z klasy potomnej.

Czym jest klasa abstrakcyjna?

To taka klasa, która przynajmniej jedna metoda jest metodą abstrakcyjną oznaczoną słowem kluczowym ***abstract***. Słowo to musi się pojawić też przed nazwą klasy. Metoda abstrakcyjna posiada jedynie definicję, nie może zawierać kodu. Nie można tworzyć obiektów tej klasy. Jeśli klasa bazowa zawiera metodę abstrakcyjną to każda klasa potomna również musi ją zawierać. Wywołania w konstruktorach nie są polimorficzne, oprócz wywołania metod.

Czym jest Interfejs? (JDK 7)

To klasa czysto abstrakcyjna, nie posiadająca żadnej implementacji oraz w której wszystkie metody są traktowane jako abstrakcyjne. Tak zdefiniowany interfejs może być implementowany przez dowolną klasę, co oznacza, że zawiera ona definicje wszystkich metod zadeklarowanych w interfejsie. To, że klasa ma implementować interfejs. Interfejs określa jakie metody muszą znaleźć się w klasie, która go implementuje. Metoda bazowa musi implementować dany interfejs jeśli implementują go metody pochodne. Jeśli klasa bazowa będzie abstrakcyjna i implementuje dany interfejs, wtedy będą musiały go implementować wszystkie nieabstrakcyjne klasy pochodne. Interfejs, oprócz deklaracji metod, może zawierać pola. Pola interfejsu zawsze są publiczne, statyczne oraz finalne, czyli trzeba przypisać im wartości już w momencie deklaracji.

Czy można w Javie dziedziczyć po wielu klasach?

W Javie klasa potomna może dziedziczyć tylko z jednej klasy bazowej, nie ma więc wielodziedziczenia. Istnieje natomiast możliwość implementowania wielu interfejsów. Klasa implementująca interfejs musi zawierać definicje wszystkich metod zadeklarowanych w obu interfejsach. Należy pamiętać przy tym o konfliktach nazw kilku metod.

Czy interfejs może dziedziczyć po wielu interfejsach?

Interfejsy można dziedziczyć w podobny sposób jak klasy, jednak w tym przypadku bez ograniczeń. Klasa, która będzie implementować taki interfejs, będzie musiała zawierać zarówno metody z interfejsu bazowego jak i z potomnego.

Czy może istnieć klasa w innej klasie?

Taka klasa to klasa wewnętrzna. Jest ona zdefiniowana we wnętrzu innej klasy. W jednej klasie może istnieć dowolna liczba klas wewnętrznych. Dostęp do klasy zewnętrznej jest pełen z klasy wewnętrznej i na odwrót. Klasa wewnętrzna nie może istnieć samoistnie bez klasy zewnętrznej. Klasy wewnętrzne domyślnie są klasami typu pakietowego. Mogą jednak także być: publiczne (da się do nich odwołać z poza klasy zewnętrznej), prywatne (tylko można się odwołać w klasie zewnętrznej) lub chronione. Klasy wewnętrzne podlegają także dziedziczeniu (w przypadku dziedziczenia klasy zewnętrznej).

Czym są klasy anonimowe? (JDK 7)

Klasa, która nie ma nazwy. Dostęp do obiektu jest jedynie poprzez interfejs. Można w ten sposób tworzyć obiekty anonimowej klasy implementującej interfejs. Nie jest to wywołanie konstruktora klasy. Dostęp do obiektu jest jedynie przez interfejs. Konsekwencją tego stanu rzeczy jest brak bezpośredniego dostępu do jakichkolwiek składowych tej klasy innych niż zdefiniowane w interfejsie. Nie ma możliwości manipulacją pól (tylko poprzez wykorzystanie metod).

Czym jest obsługa błędów i wyjątków?

Obsługa błędów i wyjątków służy do wyeliminowania błędów zawartych w kodzie programu. Typowe sytuacje, w których kompilator wyświetli błąd to: odwołanie się do nieistniejącego elementu tablicy (sprawdzenie poprawności zakresu), nieprawidłowa współpraca dwóch klas, itp.

Do czego służą wyjątki?

Wyjątek to obiekt, który występuje w przypadku błędu. Czyli należy zaprogramować metodę sprawdzającą warunek zajścia wyjątku oraz efekt w przypadku zajścia lub nie. Można go przechwycić i wykonać własny kod. Jeśli tego się nie zrobi zostanie on obsłużony przez system a na konsoli wyświetli się odpowiedni komunikat.

Czym jest sekcja *finally*?

Instrukcje wykonywane po słowie ***finally*** wykonywane są zawsze (niezależnie od tego czy wyjątek wystąpił czy nie). Blok *finally* nie jest konieczny do stosowania.

W jaki sposób propagować wyjątek?

W celu samodzielnego utworzenia wyjątku stosuje się instrukcję ***throw***.

Czy istnieje korelacja między pozycją przechwytywania wyjątku w sekcji *catch*?

Kolejność przechwytywania wątków ma znaczenie i zależy od ich priorytetu.

Jak wygląda hierarchia wyjątków?

Object -> Throwable -> Exception -> RuntimeException -> wyjątki szczegółowe (np. ArithmeticException).

Jakie są sposoby tworzenia wątków w Javie?

1. Kod realizujący zadanie umieszcza się wewnątrz metody *run* klasy implementującej interfejs *Runnable*. Interfejs posiada tylko jedną metodę: *public interface Runnable { void run(); }*. Klasę implementująca interfejs tworzy się w następujący sposób:
```
class MyRunnable implements Runnable {
      public void run()
            { // kod }
      }
}
```
2. Następnie tworzony jest obiekt klasy:
```
Runnable runnable = new MyRunnable();
```
3. Tworzy się obiekt *Thread* na podstawie obiektu *Runnable:*
```
Thread thread = new Thread(runnable);
```
4. Uruchamia się wątek:
```
thread.start();
```

Jaka jest różnica pomiędzy metodami *start*() i *run*()?

Główna różnica polega na tym, że gdy program wywołuje metodę start(), tworzony jest nowy wątek, w którym to kod w metodzie *run*() jest wykonywany, podczas gdy bezpośrednio wywołanie metody *run*() nie spowoduje utworzenia nowego wątku, lecz jej kod zostanie wykonany w wątku bieżącym.

W jaki sposób można zatrzymać działanie wątku?

Wykonanie wątku kończy się w momencie, gdy metoda run zwraca sterowanie. Nie istnieje sposób, aby wymusić zakończenie wątku. Można za to zażądać zakończenia wątku za pomocą metody ***interrupt***. Powoduje ona nadanie wątkowi status przerwania.

Na czym polega synchronizacja wątków?

Polega na odpowiedniej modyfikacji stanu obiektu tak aby dwa lub więcej wątków pracujących na tych samych danych było wstanie prawidłowo je wykorzystywać bez utraty danych. Taka sytuacja nazywana jest wyścigiem.

Wymień znane Ci synchronizatory (mechanizmy z gotowymi funkcjonalnościami współpracy między wątkami)?

- ***CyclicBarier*** – zbiór wątków czeka na zwolnienie aż osiągnie ją określona liczba wątków.
- ***CountDownLatch*** – zbiór wątków oczekuje aż licznik = 0.
- ***Exchanger*** – umożliwia dwóm wątkom wymianę obiektów, gdy są do tego gotowe.
- ***SynchronousQueue*** – umożliwia wątkowi przekazanie obiektu innemu wątkowi.
- ***Semaphore*** – zbiór wątków ubiega się o pozwolenie kontynuowania przetwarzania.

Czym jest *JDBC*?

Pakiet *Java Database Connectivity* umożliwia łączenie się z bazami danych i wykonywaniem zapytań *SQL*. Dzięki niemu powstałe aplikacje stają się niezależne od platformy i od bazy danych, której używają. Koncepcją używania *JDBC* jest to, że programy używające interfejsu programowego komunikują się z programem zarządzającym, który z kolei wykorzystuje odpowiedni sterownik do dostępu do bazy danych.

Opisz nowe funkcjonalności, które zostały dodane w JDK 8?

Zmiany w interfejsie – dodano możliwość implementowania metod domyślnych (czyli metod z ciałem, które są użyte, gdy klasa nie nadpisuje sygnatury) oraz metod statycznych. Dodano interfejs funkcyjnych (czyli z jedną sygnaturą), używany w wyrażeniach lambda.

Optional - klasa opakowująca, która wymusza na programiście obsługę wyjątków typu *nullpointer*.

Lambda – wyrażenia typu lambda, które umożliwiają implementowanie klas anonimowych w sposób zbliżony do programowania funkcyjnego.

Stream API - Strumieniem nazywamy specjalną sekwencję operacji, które można wykonywać jedną po drugim (w formie tzw. pipeline'u), tak aby przekształcić go do dowolnego rezultatu. *Stream API* szeroko wykorzystuje lambdy i interfejsy funkcyjne.

Date API – wprowadzono nowe API do pracy z datami.

Testowanie

„Tu nie chodzi o testowanie. Chodzi o odpowiedzialność."

Jaka jest różnica między testami funkcyjnymi a strukturalnymi?

Testy funkcyjne - Polegają one na tym, że wcielamy się w rolę użytkownika, traktując oprogramowanie jak „czarną skrzynkę", która wykonuje określone zadania. Nie wnikamy w ogóle w techniczne szczegóły działania programu. Testy te często są nazywane testami czarnej skrzynki (*black box testing*).

Testy strukturalne - Tym razem tester ma dostęp do kodu źródłowego oprogramowania, może obserwować jak zachowują się różne części aplikacji, jakie moduły i biblioteki są wykorzystywane w trakcie testu. Te testy czasami są nazywane testami białej skrzynki (*white box testing*).

Wymień rodzaje testów ze względu na automatykę.

Test manualne - mogą być wykonywane ręcznie przez testera, który przechodzi przez interfejs użytkownika zgodnie z określoną sekwencją kroków.

Testy automatyczne, których wykonanie nie wymaga udziału testera. Zazwyczaj zautomatyzowane jest przeprowadzanie testów jednostkowych (ang. *unit tests*). Znacznie bardziej skomplikowaną sprawą jest automatyzacja testów w schemacie czarnej skrzynki. Potrzebne do tego jest specjalistyczne oprogramowanie, które pozwala uruchamiać uprzednio napisane lub nagrane przez testera skrypty.

Wymień rodzaje testów ze względu na zakres.

Testy jednostkowe - testują oprogramowanie na najbardziej podstawowym poziomie – na poziomie działania pojedynczych funkcji (metod).

Testy integracyjne - pozwalają sprawdzić jak współpracują ze sobą różna komponenty oprogramowania.

Testy systemowe - dotyczą działania aplikacji jako całości, wymagań niefunkcjonalnych: szybkość działania, bezpieczeństwo, niezawodność, dobrą współpracę z innymi aplikacjami i sprzętem.

Wymień rodzaje testów ze względu na przeznaczenie.

Testy akceptacyjne – testy wykonywane w celu sprawdzenia na ile oprogramowanie działa zgodnie z wymaganiami klienta (z reguły są to testy funkcjonalne + ewentualnie jednostkowe).

Testy funkcjonalne – testy sprawdzające działanie oprogramowania zgodnie ze specyfikacją.

Testy regresyjne – jest to bardzo ważny rodzaj testów, pełniących zasadniczą rolę jeśli traktujemy poważnie kwestię jakości oprogramowania. Celem testów regresyjnych jest sprawdzenie, czy dodając nową funkcjonalność, poprawiając błędy nie naruszyliśmy niespodziewanie innej funkcjonalności oprogramowania. Testy regresyjne powinny być wykonywanie zarówno na poziomie kodu aplikacji (jeśli to możliwe) – zazwyczaj są to testy jednostkowe – jak i na wyższym poziomie, działania całej aplikacji. Aplikacja jest testowana w ten sposób, że przechodzimy przez wybrane ścieżki działania oprogramowania tak, jak by to robił jego użytkownik.

Testy wydajnościowe i obciążeniowe.

Wzorce projektowe

„Jedynym sposobem, aby robić coś szybko, jest robić to dobrze."

Singleton

Idea tego wzorca polega na tym, aby mogła być tylko jedna instancja danej klasy w projekcie.

Można to osiągnąć w następujący sposób:

1) zablokować możliwość tworzenia obiektu za pomocą operatora new, poprzez zdefiniowanie konstruktora jako prywatny.
2) pole statyczne tego samego typu co klasa, która ma zwracać instancję.
3) getter do pola instancji, tak aby można było się w do niej odwoływać. Metoda typu get powinna tworzyć obiekt, gdy nie jest *null*. W przypadku pracy w środowisku wielowątkowym należy zabezpieczyć ją przynajmniej poprzez dodanie słowa kluczowego *synchronized*.
4) inne pola i metody dotyczące logiki klasy

Przykład prostego Singletonu:

```
public final class ClassicSingleton {
	private static ClassicSingleton instance;
	private static int counter = 0;

	private ClassicSingleton() {
		counter++;
	}

	public static synchronized ClassicSingleton getInstance() {
		if (instance == null) {
			instance = new ClassicSingleton();
		}
		return instance;
	}

	public void methodInSingleton() {
		System.out.println("Invoke classic singleton " + counter);
	}
}
```

Budowniczy

Celem tego wzorca jest ograniczenie liczby i wielkości konstruktorów w przypadku dużej ilości pól w klasie. Dzięki temu wzorcowi można tworzyć obiekty, korzystając ze statycznej klasy wewnątrz Twojego POJO.

Przykład:

```
public class BuilderPattern {
	private final Long id;
	private final String firstname;
	private final String lastname;
	private String address;

	public BuilderPattern(Builder builder) {
		this.id = builder.id;
		this.firstname = builder.firstname;
		this.lastname = builder.lastname;
		this.address = builder.address;
	}

	public BuilderPattern() {
		throw new UnsupportedOperationException("Not supported yet.");
	}

	public Long getId() {
		return id;
	}

	public String getFirstname() {
		return firstname;
	}

	public String getLastname() {
		return lastname;
	}

	public String getAddress() {
		return address;
	}
```

```
        public static class Builder {
              private Long id;
              private String firstname;
              private String lastname;
              private String address;

              public Builder(Long id, String firstname, String lastname) {

                    this.id = id;
                    this.firstname = firstname;
                    this.lastname = lastname;
              }

              public Builder address(String address) {
                    this.address = address;
                    return this;
              }

              public BuilderPattern build() {
                    return new BuilderPattern(this);
              }
        }
  }
```

Fabryka

To wzorzec, którego celem jest tworzenie obiektów pochodzących z jednej rodziny (np. implementujące ten sam interfejs). W zależności od żądania danych obiekt jest za każdym razem tworzony (przeciwieństwo Singletona).

MVC (ang. *Model-View-Controller*)

Polega na podział architektury aplikacji na trzy części. Kod każdej z nich powinien być przechowywany w innym module. Pierwszy z nich, widok, zajmuje się logiką związaną z wyświetlaniem danych (np. strona HTML). Drugi, kontroler, przechwytuje żądania HTTP i mapuje odpowiedni widok z odpowiednim modelem. Ostatni, model, reprezentuje logikę aplikacji oraz generuje dane docelowe do wyświetlenia w widoku.

Dekorator

Dekorator jest strukturalnym wzorcem, którego głównym zadaniem jest dodanie dodatkowej funkcjonalności, do istniejącego obiektu, bez modyfikowania zachowania bazowego.

Obserwator

Zasadą działania tego wzorca polega na tym, że każda zmiana w obserwowanej klasie jest rejestrowana przez inne klasy. Posiadają one logikę, która będzie w określony sposób reagować w zależności od stanu klasy nadsłuchiwanej.

Adapter

Jego celem jest dostosowanie starego API do nowego wzorca. Adapter jest więc pośrednikiem i stara się przekształcić usługi jednej klasy w taki sposób, by mogły być one widziane w postaci metod innej klasy.

Strategia

ideą tego wzorca jest stworzenie różnych implementacji do rozwiązania jednego problemu. Obiekt podlegający strategii wpływa na algorytm na wybór rodzaju wykonywania. Dobrym przykładem jest wybór strategii rozpakowywania pliku w zależności od rozszerzenia.

Hibernate

„Kiedy dane biznesowe zostaną scentralizowane i ujednolicone, ich wartość jest większa niż suma wcześniej istniejących części.”

Czym jest ORM?

ORM to skrót od Mapowanie Obiektowo – Relacyjne. Jest to zautomatyzowane utrwalanie w tabelach bazy danych SQL obiektów z wykorzystaniem metadanych opisujących odwzorowanie pomiędzy klasami a schematem bazy danych.

Różnica między *JPA* i *Hibernate*?

JPA (czyli *Java Persistence API*) to ogólna specyfikacja przygotowana przez twórców Java, będąca częścią większego projektu o nazwie JAVA EE (*Enterprises Edition*). Służy ona do mapowania obiektowo - relacyjnego. *Hibernate* to najpopularniejsza implementacja JPA (ale nie jedyna).

W jaki sposób zmapować klasę na tabelę?

Mapowanie klasy odbywa się poprzez wyspecyfikowane adnotacje (kiedyś korzystano też z deskryptorów w formie xml, obecnie nikt już tego nie używa, więc pomijam to zagadnienie). Minimalne warunki do tego, aby klasa została zmapowana poprawnie to:

- klasa powinna być typu *POJO* (czyli pola, ich metody typu *get* i *set* oraz domyślny konstruktor)
- Adnotacja @Entity nałożona na klasę
- Adnotacja @Id nałożona na pole, będące kluczem głównym

Jakie są relacje w *Hibernate*?

- jeden do jeden (*@OneToOne*)
- jeden do wiele (*@OneToMany*)
- wiele do jeden (*@ManyToOne*)
- wiele do wiele (*@ManyToMany*)

Czym jest parametr *fetchType*?

Odpowiada za rodzaj ładowania danych w przypadku relacji między klasami. Wyróżnia się dwie możliwości pobierania danych: *lazy* (leniwe) oraz *eager* (zachłanne). Ładowanie leniwe oznacza, że *Hibernate* załaduje dane dopiero wtedy, gdy nastąpi żądanie. W przypadku ładowania zachłannego nastąpi to natychmiast.

W jaki sposób można zarządzać encjami bazy z poziomu kodu Java?

Można wykorzystać dwa rodzaje API, które udostępnia metody, służące do tworzenia, aktualizacji, usuwania encji, itp. Jeden to *EntityManager* (JPA), a drugi sposób to poprzez klasę *Session* (*Hibernate*). Metody z *EntityManagera* mają swoje odpowiedniki w metodach klasy *Session* (np. *get* i *find*).

W jakich stanach mogą być obiekty encji?

TRANSIENT (stan przejściowy, obiekt został stworzony, ale jeszcze nie zapisany w bazie),

PERSISTENT (obiekt ma już reprezentację w bazy danych),

REMOVED (obiekt został usunięty z bazy),

DETACHED (obiekt był wcześniej użyty, ale na razie praca na nim została zakończona).

Czym jest parametr *Cascade*?

Każda encja ma niezależny cykl życia. Podczas niektórych asocjacji, można wymusić zachowanie powiązanych rekordów. Możliwe opcje to:

- **PERSIST** (w czasie synchronizacji zapisywane są powiązane encje),
- **REMOVED** (synchronizacja podczas usuwania),
- **DETACH** (gdy encja jest wstanie 'odłączonym' powiązane rekordy są także odłączane),
- **MERGE** (gdy odłączona wcześniej encja jest znowu scalana z kontekstem, rekordy powiązane będą też scalane),
- **REFRESH** (odświeżane są wszystkie egzemplarze encji powiązane),
- **ALL** (wszystko powyżej).

Co robi metoda *Flush*?

Flushing to proces synchronizowania stanu kontekstu *Hibernate* z faktycznymi wartościami w bazie danych.

Czym się różnią metody *load* i *get*?

Obie metody, służą do pobierania danych dla podanego identyfikatora. Jednak metoda ***get*** zwraca wartość *null*, jeżeli w pamięci podręcznej sesji lub w bazie danych dla podanego identyfikatora nie jest dostępny żaden wiersz, natomiast metoda ***load*** zgłasza wyjątek dotyczący nieznalezionego obiektu.

Jaka jest różnica między metodami *save* i *update*?

Save pobiera instancję encji, dodaje ją do kontekstu i zarządza tą instancją (tj. przyszłe aktualizacje encji będą śledzone). ***Update*** zwraca zarządzaną instancję, z którą stan został scalony. Zwraca coś, co istnieje w *PersistenceContext* lub tworzy nową instancję twojej encji.

W jaki sposób można wykonywać w *Hibernate* skomplikowane zapytania typu *SELECT*?

- Za pomocą *Namedquery* (zapytania tego typu powinny mieścić się w klasach *Entity*)
- Z wykorzystaniem języka *HQL* (język podobny do *SQL*, jednak odwołuje się do samych obiektów a nie schematu bazy danych)
- Z użyciem *Criteria API* (jest to specjalne API, które pozwala budować zapytanie z wykorzystaniem już wcześniej zaprogramowanych metod, będących odzwierciedleniem składni języka *SQL*).

Spring

„Dobra architektura sprawia, że system jest łatwy do zrozumienia, rozwoju, utrzymania i wdrożenia.”

Czym jest *Spring*?

Spring to rozbudowany *framework*, będący zbiorem wielu bibliotek do Javy i opierający się o kontener inwersji kontroli.

Czym jest wstrzykiwanie zależności? (ang. *Dependency Injection*)

Wstrzykiwanie zależności to technika, w której klasa korzysta z innych obiektów bez tworzenia ich za pomocą słowa kluczowego *new*. Intencją wstrzykiwania zależności jest rozdzielenie zarządzanie cyklem życie obiektu, od jego logiki. Programista zajmuje się tu tylko logiką biznesową, a za tworzenie obiektu odpowiada kontener *IoC* (ang. *Inversion of Control*). Pomaga to zwiększyć czytelność i ponowne wykorzystanie kodu.

Jakie są trzy możliwości wstrzykiwania zależności w *Springu*?

- ✓ poprzez pole
- ✓ poprzez konstruktor (niejawnie domyślne od wersji 4.3)
- ✓ poprzez *setter*

Wyjaśnij poniższe adnotacje.

@Bean – najmniejsza „komórka” Springa. Nakładana na metodę. Głównie używana w klasach konfiguracyjnych.

@Component – nakładane na klasę. Definiuje jeden komponent Springowy.

@Configuration – klasa definiująca konfigurację Springa

@Service – rodzaj komponentu, oznaczający klasę z logiką biznesową

@Controller – specjalny komponent, definiujący mapowanie żądań HTTP pomiędzy aplikacją a klientem

@RestController – kontroler, który pracuje z plikami w formacie JSON

@Repository – komponent, który zajmuje się wykonywaniem zapytań do bazy danych

W jaki sposób zarządzać transakcjami z poziomu *Springa*?

Aby zarządzać automatycznie transakcjami w Springu wystarczy skorzystać z adnotacji @*Transactional* na metodzie, która będzie wykonywać operacje na bazie danych. Spring dzięki tej adnotacji będzie wiedział, w którym momencie otworzyć i zamknąć transakcję.

Czym są poziomy propagacji transakcji? Wymień je.

Spring posiada mechanizm definiowania w jaki sposób transakcja ma się zachować w przypadku, gdy zostaje otwarta we wcześniej zainicjalizowanej transakcji.

Możliwe propagacje to:

REQUIRED (domyślna) - Spring sprawdza, czy istnieje aktywna transakcja, jeśli nie, to tworzy nową. W przeciwnym razie logika biznesowa jest dołączana do aktualnie aktywnej transakcji.

REQUIRES_NEW – Gdy ta propagacja jest użyta, Spring zawiesza bieżącą transakcję, jeśli istnieje, a następnie tworzy nową.

NEVER – Jeśli metoda oznaczona tą adnotacją zostanie użyta w transakcji, to Spring wyrzuci wyjątek.

MANDATORY –Jeśli istnieje aktywna transakcja, zostanie ona użyta. Jeśli nie ma aktywnej transakcji, Spring zgłasza wyjątek.

NESTED – Jeśli transakcja istnieje, Spring wykonuje w tym miejscu punkt zapisu (ang *save point*). Oznacza to, że jeśli kod oznaczony tą adnotacją zgłosi wyjątek, to transakcja zostanie przywrócona do tego momentu. Jeśli nie ma aktywnej transakcji, działa jak propagacja *REQUIRED*.

SUPPORTS – W tym przypadku Spring najpierw sprawdza, czy istnieje aktywna transakcja. Jeśli transakcja istnieje, zostanie ona użyta do przetworzenia kolejnych operacji. Jeśli żadna transakcja nie została wcześniej otwarta, to kod zostanie wykonywana w trybie nietransakcyjnym.

NOT_SUPPORTED - Jeśli istnieje bieżąca transakcja, najpierw Spring ją zawiesza, a następnie logika biznesowa jest wykonywana bez transakcji.

Rodzaje zakresów (*scopes*) w Springu?

SINGLETON (domyślny) - instancja beanu Springa będzie tylko raz ustalona i ta sama instancja zostanie zwrócona przez kontener Springa.

PROTOTYPE - instancja beanu będzie tworzona za każdym razem, gdy zajdzie żądanie.

REQUEST – nowa instancja zostanie utworzona na żądanie HTTP.

SESSION – nowa instancja będzie tworzona na sesję HTTP.

GLOBALSESSION – podobnie jak wyżej ale zostanie stworzony tylko podczas globalnej sesji HTTP.

Czym jest *ApplicationContext*?

To klasa reprezentująca kontekst Springowy. Umożliwia sterowanie kontekstem jak i odwołanie się do poszczególnych beanów, za pomocą kodu a nie adnotacji.