**TEST WIEDZY W ZAKRESIE TECHNOLOGII BACKEND**

Imię i nazwisko: **Artur Olejniczak**

Data: 18.06.2021

1. **Czym różni się obiekt od klasy?**

|  |  |  |
| --- | --- | --- |
| ***Lp.*** | ***Obiekt*** | ***Klasa*** |
| *1)* | *Obiekt jest* ***instancją*** *klasy.* | *Klasa to* ***plan lub szablon,*** *z którego tworzone są obiekty.* |
| *2)* | *Obiekt to obiekt ze* ***świata rzeczywistego,*** *taki jak długopis, laptop, telefon komórkowy, łóżko, klawiatura, mysz, krzesło itp.* | *Klasa to* ***grupa podobnych obiektów****.* |
| *3)* | *Obiekt jest bytem* ***fizycznym.*** | *Klasa to jednostka* ***logiczna****.* |
| *4)* | *Obiekt jest tworzony poprzez* ***nowe słowo kluczowe*** *głównie np. Student s1=new Student();* | *Klasa jest deklarowana za pomocą* ***słowa kluczowego*** *class, np. class Student{}* |
| *5)* | *Obiekt jest tworzony* ***wielokrotnie*** *zgodnie z wymaganiami.* | *Klasa jest deklarowana* ***raz****.* |
| *6)* | *Obiekt* ***alokuje pamięć, gdy jest tworzony****.* | *Klasa* ***nie alokuje pamięci podczas jej tworzenia****.* |
| *7)* | *Istnieje* ***wiele sposobów tworzenia obiektu*** *w java, takich jak nowe słowo kluczowe, metoda newInstance(), metoda clone(), metoda fabryki i deserializacja.* | *Istnieje tylko* ***jeden sposób zdefiniowania klasy*** *w java za pomocą słowa kluczowego class.* |

1. **Czy różni się klasa abstrakcyjna od interfejsu?**

|  |  |
| --- | --- |
| *Klasa abstrakcyjna* | ***Interfejs*** |
| *1) Klasa abstrakcyjna może* ***mieć*** *metody* ***abstrakcyjne  i*** *nieabstrakcyjne.* | *Interfejs może mieć* ***tylko*** *metody* ***abstrakcyjne****. Od wersji Java 8 może mieć również* ***metody domyślne i statyczne****.* |
| *2) Klasa abstrakcyjna* ***nie obsługuje wielokrotnego dziedziczenia****.* | *Interfejs* ***obsługuje dziedziczenie wielokrotne*** *.* |
| *3) Klasa abstrakcyjna* ***może mieć zmienne końcowe, nieostateczne, statyczne i niestatyczne****.* | *Interfejs posiada* ***tylko zmienne statyczne i końcowe****.* |
| *4) Klasa abstrakcyjna* ***może zapewnić implementację interfejsu****.* | *Interfejs* ***nie może zapewnić implementacji klasy abstrakcyjnej****.* |
| *5)* ***Słowo kluczowe abstract*** *służy do deklarowania klasy abstrakcyjnej.* | ***Kluczowe interfejs*** *służy do deklarowania interfejs.* |
| *6)* ***Klasa abstrakcyjna*** *może rozszerzać inną klasę Javy i implementować wiele interfejsów Javy.* | ***Interfejs*** *może przedłużyć tylko inny interfejs Java.* |
| *7)* ***Klasę abstrakcyjną*** *można rozszerzyć za pomocą słowa kluczowego „extends”.* | ***Interfejs*** *może być realizowane za pomocą słów kluczowych „narzędzi”.* |
| *8)****Klasa abstrakcyjna*** *Java może mieć członków klasy, takich jak prywatne, chronione itp.* | *Elementy interfejsu Java są domyślnie publiczne.* |

1. **Proszę wyjaśnić mechanizm dziedziczenia w Javie.**

*Dziedziczenie to mechanizm dzięki, któremu obiekt, który dziedziczy nabywa wszystkie właściwości i zachowanie obiektu, po którym dziedziczył (obiektu nadrzędnego).*

*Gdy klasa podrzędna ‘Bob’ dziedziczy z klasy nadrzędnej ‘People’ to stworzona instancja klasy ‘Bob’ może używać metod i pól klasy ‘People’ oraz również może dodać nowe metody i pola w swojej obecnej klasie.*

*Każda klasa dziedziczy po object..*

1. **Co należy zdefiniować w klasie podrzędnej**
   1. Nic
   2. Tylko różnice między klasą podrzędną, a nadrzędną
   3. Wszystko co dotyczy tej klasy

Odp. A

klasa podrzędna może mieć swoje pola i metody ale nie musi, ponieważ posiada już pola i metody od klasy nadrzędnej zakładając że klasa nadrzędna je posiada itd.

1. **W którym miejscu klasy można deklarować zmienne**
   1. W dowolnym miejscu
   2. Po deklaracji klasy, ale przed pierwszą metodą
   3. Poza metodami klasy

Odp. Wszystkie odpowiedzi błędne :)

Zmienne są umieszczone w metodach.

Pola klasy umieszcza się w bloku klasy poza metodami

1. **Proszę omówić modyfikatory dostępu w Javie.**

Modyfikatory dostępu służą do ustawienia widoczności elementu którego poprzedzają np. pola, metody, klasy. Używa się ich w celu hermetyzacji. Jest to pewnego rodzaju ukrycie szczegółów.

Wyróżniamy cztery:

public, protected, brak modyfikatora(package private), private.

- public oznacza, że kod jest dostępny dla wszystkich klas.

- protected oznacza, że kod jest dostępny w tym samym pakiecie i podklas.

- brak modyfikatora(domyślny) oznacza, że kod jest dostępny tylko w tym samym pakiecie

- private oznacza, że kod jest dostępny tylko w zadeklarowanej klasie

1. **Które ze słów kluczowych powoduje przeskoczenie z bloku *try* do bloku *finally***
   1. *return*
   2. *catch*
   3. *while*

Odp.A

return zwróci wartość i wyjdzie z bloku ‘try’ nie zważając na to, że po return może być jeszcze kod do wykonania.

1. **Mając listę** **String[] drzewa = {„brzoza”, „modrzew”, „grab”, „buk”, „wierzba” } proszę zaproponować wyświetlenie jej na ekranie za pomocą iteratora i za pomocą generyków.**

Zadanie w załączniku

1. **Proszę zaproponować kod źródłowy metody weryfikującej czy podane 2 wyrazy składają się z tych samych liter np.  
   MARA – składa się z 1 litery M, 2 liter A, 1 litery R  
   RAMA – składa się z 1 litery R, 2 liter A, 1 litery M  
   Podane przykłady składają się z tych samych liter.  
   Napisać podstawowe testy jednostkowe dla tej metody (jUnit).**Aplikacja dostępna do obejrzenia w załączniku.

Nie zdążyłem zrobić testów.

1. Proszę przedstawić wysokopoziomowy diagram systemu dostępnego przez przeglądarkę internetową, przechowującego informacje o sukcesach sportowych osób znajdujących się w systemie.

Proszę zaproponować model logiczny bazy danych, oraz wysokopoziomowo komponenty systemu. Nie ma potrzeby schodzić na poziom kodu źródłowego.

W sytuacji wymagającej doszczegółowienia wymagań proszę przyjąć własne założenia i opisać je.

Nie zdążyłem zrobić.

1. . Napisać test dla poniższego fragmentu kodu.

**public void** wyslijSms(String numerTel, String trescSms) {   
 smsGateway.send(**new** Sms(numerTel, trescSms));  
}

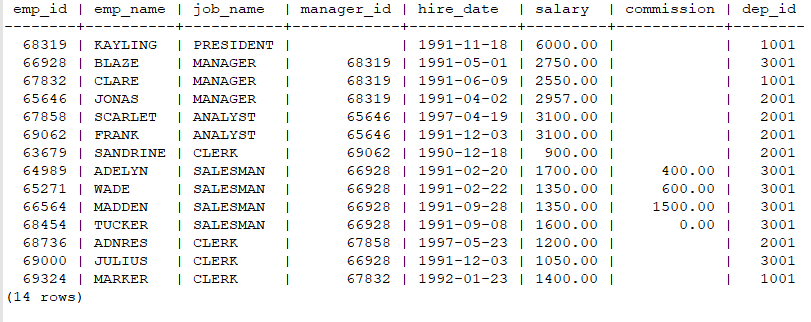
Bramka rzuca wyjątek SMSGatewayException(„Bramka przeciążona”) w momencie, gdy jest zbyt duże obciążenie bramk.

Nie zdążyłem zrobić.

1. Zaprojektować kontrakt usługi REST w formacie OpenApi oraz małą aplikację springbootową, która zwraca listę książek (bez użycia bazy danych). Usługa powinna umożliwiać filtrowanie po części nazwy książki oraz po numerze IBAN.  
   Aplikacja Spring MVC dostępna do obejrzenia w załączniku.  
   Na tą chwile nie ma filtrowania. Aplikacja ma zaimplementowane takie funkcje jak  
   - dodanie książki poprzez <http://localhost:8080/book/addNew>  
   - lista książek/edytowanie/usuwanie dostępne tutaj <http://localhost:8080/book/list>
2. Czy zbiór (Set) może zawierać duplikaty?
   1. Tak
   2. Nie

Odp.B

1. Zaznaczyć metody należące do java.util.Set:
   1. addAll
   2. contains
   3. isEmpty
   4. size
   5. remove
   6. indexOf
   7. lastIndexOf
2. Napisać zapytanie zwracające ilość praconików bez przełożonych z poniższej tabeli.



Odp.  
select job\_name, count(\*) from tableName where job\_name !='PRESIDENT' and job\_name != 'MANAGER' group by job\_name;

1. Z powyższej tabeli napisać zapytanie zwracające kwotę najmniejszego wynagrodzenia.   
   Odp.  
   select min(salary) from tableName;
2. Napisz fragment kodu w języku Java, który sprawdzi pełnoletność osoby. Parametrem wejściowym do obliczeń jest numer PESEL w postaci String.  
   Odp.  
   Zadanie w załączniku
3. Poniższa metoda:

**private static** *String* getElement(Object list, Integer number) {  
 **return** ((List)list).get(number).toString();  
}

zadziała dla poniższych danych

**public static** *void* main(String[] args) {  
 List<String> list = **new** ArrayList<>();  
 list.add("A");  
 list.add("B");  
 list.add("C");  
 System.out.print(getElement(list, 1));  
}

W jakich przypadkach (dla innych danych wejściowych) metoda getElement może spowodować błąd i jak się przed tym zabezpieczyć?  
**1 przypadek**

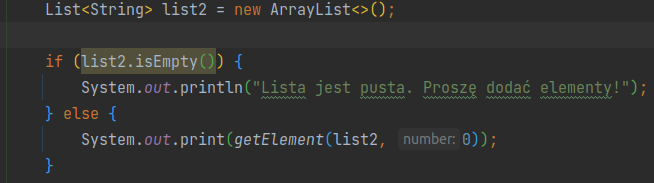
getElement(list, 3);

Dostaniemy IndexOutOfBoundsException, ponieważ odwołaliśmy się do indeksu spoza zakresu. Można zapobiegać poprzez chociażby zastosowanie bloku try-catch, a następnie np. w bloku catch zaimplementować powtórne wpisanie indeksu elementu z listy.

**2 przypadek**

List*<*String*>* list2 = new ArrayList*<>()*;  
System.*out*.print*(getElement(*list2, 2*))*;

w tym przypadku można byłoby sprawdzić zanim wywołamy metode getElement czy lista list2 nie jest pusta.



**3 przypadek**  
Lista może nie istnieć. Błąd zostanie wykryty w czasie kompilacji. W tym przypadku trzeba skorzystać z refleksji.

1. Czym różnią się poniższe zapisy i który jest preferowany oraz dlaczego:

public void test(String wartosc) {

/\*1\*/if (wartosc.equals(""));

/\*2\*/if ("".equals(wartosc));  
 /\*3\*/if (wartosc == "");

„==” służy do porównania referencji, czyli czy oba obiekty wskazują na tę samą lokalizację w pamięci.  
 „.equals()” służy do porównania typów obiektowych. Porównuje wartość w obiektach/równość strukturalną obiektów a nie ich referencje do lokalizacji w pamięci.   
  
Jeżeli miałbym wybrać preferowany zapis dla przypadku powyżej wybrał bym equals(), ponieważ porównujemy wartości typu String(nie jest to typ prymitywny do którego użyłbym ‘==’)  
  
Poprawnie zaimplementowana metoda equals jest symetryczna dlatego zapisy 1 i 2 powinny zwracać to samo.

1. Co to są obiekty immutable? Czy final sprawia że obiekt jest immutable i dlaczego?

Obiekty immutable są to obiekty, których stan wewnętrzny pozostaje niezmienny. Słowo kluczowe final sprawia, że kompilator nie pozwoli nam zmienić wartości tej zmiennej. Nie pozwoli nam zmienić referencji, gdzie przechowywana jest zmienna ale nie chroni nas przed zmianą wewnętrznego stanu obiektu, do którego się odwołuje.  
Np.:  
*final List<String> strings = new ArrayList<>();   
assertEquals(0, strings.size());   
strings.add("baeldung");   
assertEquals(0, strings.size());*Drugie assertEquals zwróci błąd, ponieważ dodanie elementu do listy zmienia rozmiar obiektu.  
Musimy stosować się do kilku zasad, aby obiekt został immutable. Np.:  
- nie dodajemy metod typu setter  
- deklarujemy wszystkie pola jako final  
- gdy pole jest mutowalnym obiektem tworzymy jego zapasową kopie dla metod typu getter  
- nie pozwalać na nadpisywać metod po przez subclasses

1. Czym różni się metoda http POST od GET?  
   Metoda GET żąda reprezentacji określonego zasobu. Żądania korzystające z GET powinny **tylko pobierać dane**. Po wykonaniu żądania:  
    *GET* [*http://www.example.com/customers/12345*](http://www.example.com/customers/12345)w odpowiedzi dostaniemy nagłówek odpowiedzi i treść odpowiedzi, czyli informacje o kliencie numer 12345.

Metoda POST służy do przesyłania danych do określonego zasobu, często powodując **zmianę stanu** lub skutki uboczne na serwerze. Po wykonaniu żądania:  
 POST <http://www.example.com/customers/12345/order/2>  
dla klienta numer 12345 zostanie utworzone zamówienie numer 2.

1. Jak powinna wyglądac sygnatura metody do której można przekazać dowolną ilość parametrów tego samego typu?  
   Odp:  
    [typ] nazwaMetody(typ... grupaArgumentow)

**Rozwiązania można przesłać w dokumencie bądź jako link do github.**