# Spis treści



***https://codebykris.com/hibernate-najpopularniejsze-pytania/***

## 1. Co to jest Hibernate?

Hibernate jest tak zwanym framework'em ORM (ang. **O**bject-**R**elational **M**apping), który pozwala programistom skoncentrować się na logice biznesowej, dbając o samo trwałość danych. Programista Java może pisać kod wykorzystując obiekty, a Hibernate zajmuje się tworzeniem tych obiektów z danymi załadowanymi z bazy danych i zapisaniem aktualizacji z powrotem do bazy danych.

## 2. Na czym polega przewaga Hibernate nad JDBS?

Oprócz trwałości, czyli zapisywania i ładowania danych z bazy danych, Hibernate zapewnia również następujące korzyści:

- Buforowanie
- Leniwe ładowanie
- Zarządzanie relacjami i dostarczenie kodu umożliwiającego odwzorowania obiektów na dane relacyjne
- Programista jest zwolniony z pisania kodu ładującego / zapisującego dane do bazy danych.

## 3. Różnica między metodą get () i load ()?

Jest to jedno z najczęściej zadawanych pytań w ramach rozmów kwalifikacyjnych na temat Hibernate, odpowiadałem już na nie kilka razy. Kluczowa różnica między metodą get() i load() polega na tym, że load () wygeneruje wyjątek, jeśli nie zostanie znaleziony obiekt o zadanym identyfikatorze. Funkcja get() dla odmiany zwróci wartość null. Inną ważną różnicą jest to, że load() może zwrócić obiekt proxy bez strzelania do bazy danych, obiekt zostanie zaciągnięty z bazy dopiero w momencie gdy będziesz próbował uzyskując dostęp do dowolnego atrybutu innego niż id. Metoda get() zawsze odpytuje bazę danych, więc czasami użycie load () może być szybsze niż metody get() . Sensowne jest używanie metody load(), jeśli wiesz, że obiekt istnieje w bazie, a metoda get (), jeśli nie jesteś pewien istnienia obiektu.

## 4. Jaki jest wymóg, aby obiekt Java stał się encją Hibernetową?

Klasa nie może być finalna i musi zawierać domyślny konstruktora bezargumentowy.

## 5. Jaka jest różnica między metodami save, persist i saveOrUpdate?

Wszystkie trzy metody są używane do zapisywania obiektów w bazie danych, ale są miedzy nimi subtelne różnice. Metoda save() może tylko wstawić rekordy do bazy, ale saveOrUpdate () może już wykonać INSERT lub UPDATE rekordu w zależności od tego czy obiekt posiada ustawione id. Ponadto typem zwrotnym save() jest obiekt Serializable, natomiast metoda persist() jest typu void.

*https://rekrutacjajava.pl/hibernate-czy-klasa-entity-do-poprawnego-dzialania-musi-posiadac-konstruktor-oraz-settery-dla-pol-2/*

## 6. Czy klasa @Entity do poprawnego działania musi posiadać konstruktor oraz settery dla pól?

Jeśli chodzi o konstruktor, to sprawa jest dość prosta. Standard JPA 2.0 wymaga bezparametrowego konstruktora. Jeśli mamy w klasie zdefiniowany konstruktor parametrowy, to musimy również wprost zdefiniować ten bezparametrowy. Jeżeli natomiast nie mamy żadnego konstruktora parametrowego, to nie musimy się o nic martwić – kompilator stworzy za nas konstruktor domyślny.
Jeśli chodzi o settery, to sprawa jest nieco bardziej skomplikowana. Wszystko zależy od tego, jakiej strategii dostępu do pól będzie używał Hibernate, a decydujemy o tym my – programiści, choć praca z ludźmi pokazuje, że często nieświadomie.

Zazwyczaj adnotację @Id umieszczamy nad polem klasy związanym z kluczem głównym:

Przykład:

```
@Entity
public class Person {

    @Id
    @GeneratedValue
    private long id;

    private String lastName;

    public String getLastName() {
        return lastName;
    }

}
```

W takim wypadku, Hibernate do odczytywania i zapisywania wartości atrybutów naszej klasy, będzie używał refleksji. Nie potrzebujemy więc definiować getterów oraz setterów dla pól (choć oczywiście jeśli ich potrzebujemy, to możemy to zrobić). Tę strategię możemy kojarzyć z pojęciem field-based access.

Jeśli jednak adnotację @Id umieścilibyśmy nad metodą getId(), a nie nad samym polem id, to settery będą wymagane dla wszystkich pól, które posiadają gettery. Jeśli bowiem chcemy mieć możliwość odczytywać wartość jakiegoś pola, to Hibernate musi mieć możliwość zapisania w nim jakiejś wartości. Tę strategię możemy kojarzyć z pojęciem property-based access.

```
@Entity
public class Person {

    private long id;

    private String lastName;

    @Id
    @GeneratedValue
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
```

```
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

}
```

Jeśli w powyższym przykładzie nie zapewnilibyśmy settera dla pola lastName, to podczas startu aplikacji napotkalibyśmy następujący wyjątek: org.hibernate.PropertyNotFoundException: Could not locate setter method for property [pl.rekrutacjajava.rj.person.Person#lastName].

Przy użyciu odpowiednich adnotacji, możemy teoretycznie mieszać obie powyższe strategie, aby dla jednych pól stosować pierwszą, a dla innych drugą. Wprowadzanie takiego braku spójności rzadko będzie jednak dobrym pomysłem.

Odpowiedź na to pytanie dotyczy ściśle Hibernate i jego wymagań. Jeśli pytanie miałoby bardziej ogólny wydźwięk, to warto pamiętać, że brak getterów i setterów może mieć swoje konsekwencje również w innych miejscach, np. przy serializacji/deserializacji JSONa w Springowym controllerze.