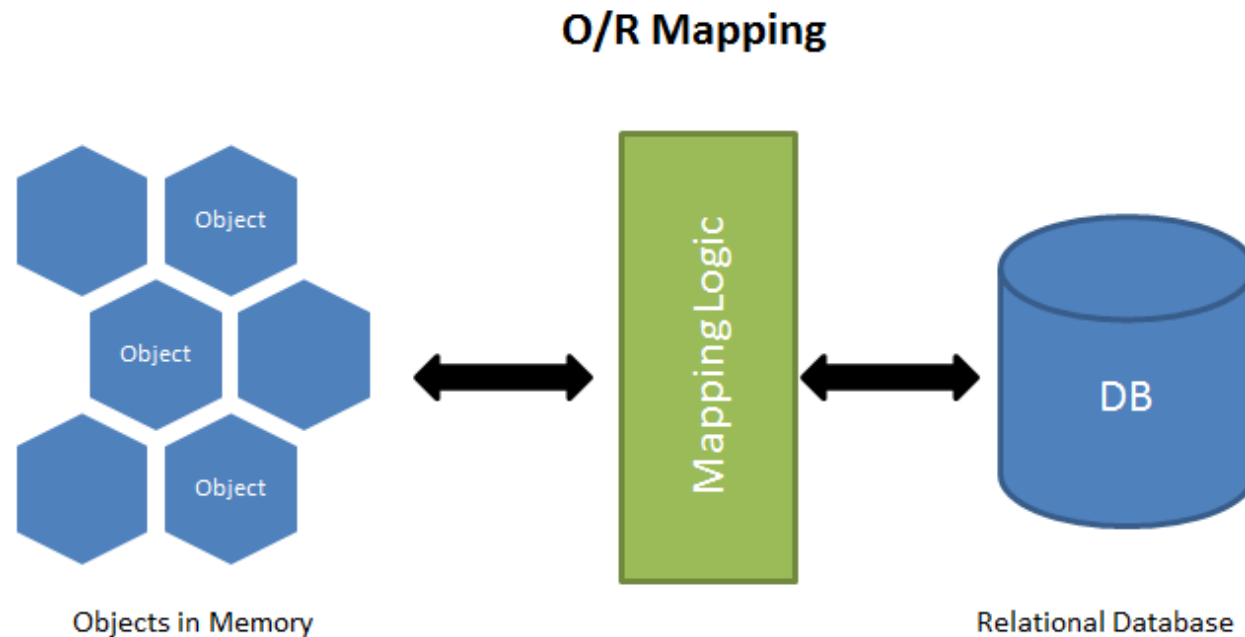




Prowadzący:  
Marcin Dziadoń  
[dziadonm@gmail.com](mailto:dziadonm@gmail.com)

# Mapowanie obiektowo relacyjne - ORM

Mapowanie obiektowo-relacyjne relacyjne (ang. Object-relational mapping) jest to konwertowanie danych z tabel w relacyjnej bazie danych na obiekty aplikacji klienckiej i na odwrót.



# Zalety i wady ORM

## Zalety:

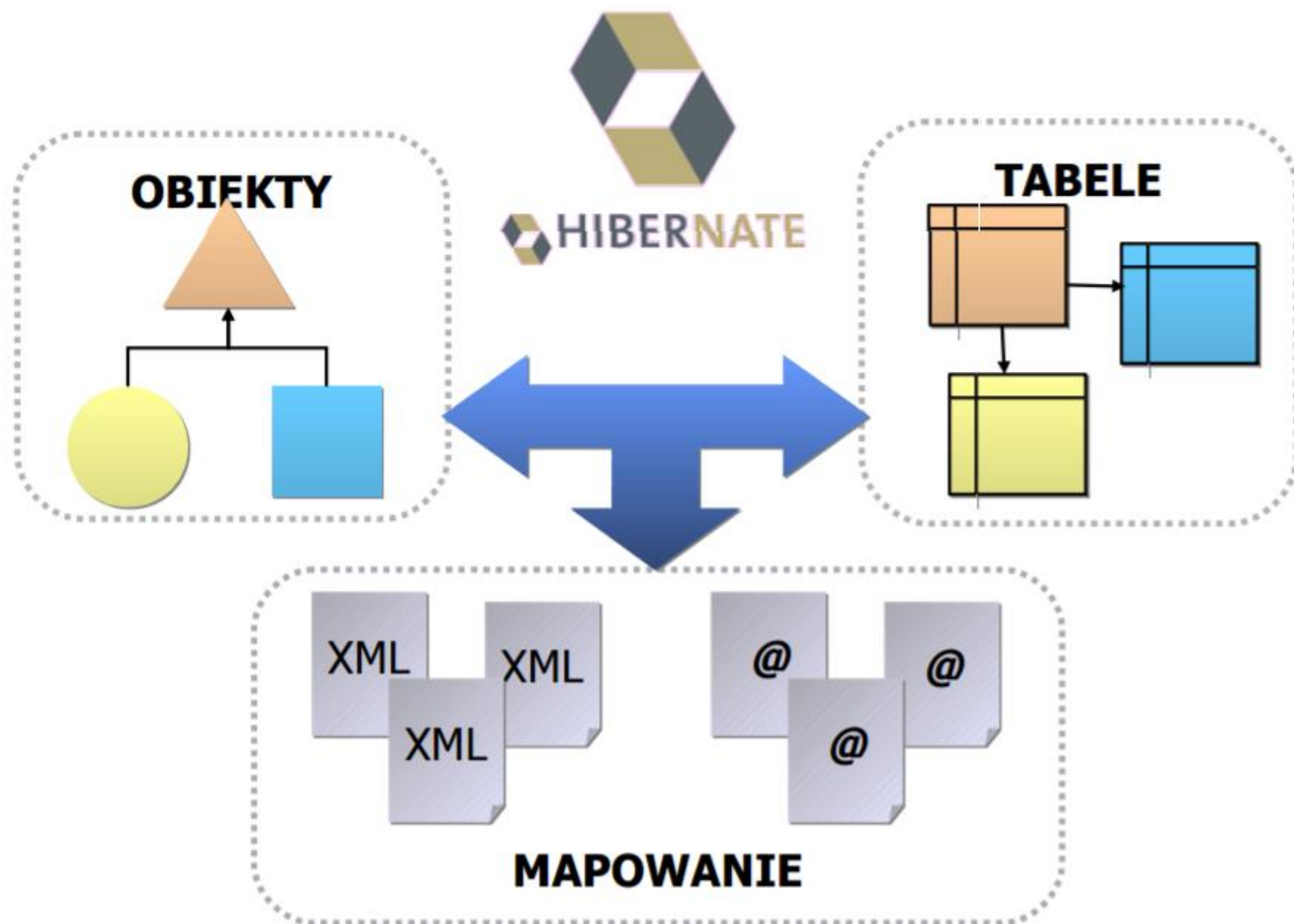
- Znaczne zredukowanie ilości pracy związanej z oprogramowaniem dostępu do danych
- Skorzystanie z zalet relacyjnych baz danych jednocześnie nie rezygnując z obiektowości programowania
- Uniezależnienie się od rodzaju bazy danych
- Automatyczna obsługa transakcji i pula połączeń z bazą

## Wady

- Łatwo można spowodować problemy z wydajnością aplikacji

# Hibernate

- Hibernate - framework do realizacji warstwy dostępu do danych.
- Hibernate zwiększa wydajność operacji na bazie danych dzięki buforowaniu i minimalizacji liczby przesyłanych zapytań.
- Jest to projekt rozwijany jako open source.



# Zalety Hibernate'a

- Dowolna klasa może reprezentować encje
- Wygodne i intuicyjne mapowanie
- Automatyczna optymalizacja = wydajność
- Mniej kodu = mniej błędów
- Wykorzystywany w wielu projektach
- Duża społeczność
- Open Source

# Wady Hibernate'a

- Dla złożonych danych, mapowanie z obiektu do tabeli i vice versa zmniejsza wydajność i zwiększa czas konwersji.
- Hibernate nie pozwala używać niektórych zapytań, które są dostępne w JDBC

# Hibernate – czy zawsze warto?

Używanie Hibernate'a jest przesadą dla aplikacji, które:

- są proste i używają jednej bazy danych, która nigdy się nie zmienia
- wstawiają dane bezpośrednio do tabel, poza tym nie używa żadnych innych zapytań SQL
- nie ma w niej żadnych obiektów, które są zmapowane na dwie różne tabele

Hibernate w takim przypadku zwiększa niepotrzebnie liczbę warstw i złożoność aplikacji. Dla takich aplikacji najlepszym wyborem jest JDBC.



# Używanie Hibernate'a w kodzie

- Uzyskanie dostępu do obiektów persystentnych tak jak do normalnego obiektu POJO (Plain Old Java Object)
- Wykonywanie operacji DAO ( save, update, delete, etc.) do zapisywania zmian do bazy danych

# Klasa persystentna

Klasa persystentna jest po prostu zwykłym beanem

```
public class Person {  
  
    private Long id;  
    private String name;  
    private String surname;  
    private String phoneNumber;  
  
    //getter i setter  
}
```

# Mapowanie za pomocą plików XML

Stare podejście, obecnie już nie używane.

```
<hibernate-mapping>
  <class name="Person" table="PERSONS">
    <id name="id" type="int" column="ID">
      <generator class="increment"/>
    </id>
    <property name="name"/>
    <property name="surname" column="SURNAME"/>
    <property name="phoneNumber" column="PHONE_NUMBER"/>
  </class>
</hibernate-mapping>
```

# Mapowanie za pomocą adnotacji

```
@Entity
@Table(name = "Persons")
public class Person {

    @Id
    @GeneratedValue
    @Column(name="id")
    private Long id;

    @Column(name="name")
    private String name;

    @Column(name="surname")
    private String surname;

    @Column(name="phone_number")
    private String phoneNumber;

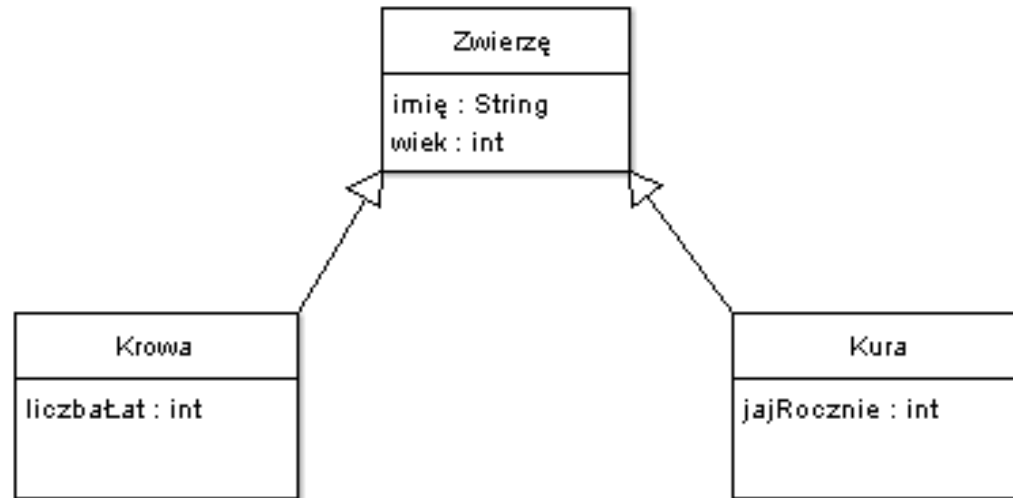
    //gettery i settery
}
```

# Typy relacji między obiektami

- ManyToOne
- OneToMany
- ManyToMany
- OneToOne

# Dziedziczenie

Hibernate oferuje trzy podejścia (sposoby implementacji) do kwestii dziedziczenia. Każdy z nich ma swoje wady i zalety. Wszystkie podejścia zostaną omówione na przykładzie następującego, prostego diagramu encji:



# Dziedziczenie - table per concrete class

Każdej nieabstrakcyjnej klasie odpowiada tabela w bazie danych. Jest to podejście najprostsze, ale jednocześnie najmniej eleganckie. Na poziomie bazodanowym zapominamy tu bowiem o jakimkolwiek związku między klasami dziedziczącymi. Dla naszego przykładu, stworzone zostałyby następujące tabele:







<div>Krowa</div> <div><div>KROWA_ID: BIGINT</div><div>IMIE: VARCHAR</div><div>WIEK: INTEGER</div><div>LICZBA_LAT: INTEGER</div></div>	<div>Kura</div> <div><div>KURA_ID: BIGINT</div><div>IMIE: VARCHAR</div><div>WIEK: INTEGER</div><div>JAJ_ROCZNIE: INTEGER</div></div>
---	--

# Dziedziczenie - table per class hierarchy

Utworzenie jednej tabeli dla wszystkich podklas. Taka tabela zawiera kolumny odpowiadające wszystkim trwałym atrybutom wszystkich podklas.

Dodatkowo, potrzebna jest jedna kolumna nazywana *discriminator*, która określa, do jakiej podklasy należy obiekt odpowiadający danemu rekordowi w tabeli.

Tabela, którą utworzyłby Hibernate dla przykładu ze zwierzętami, miałaby następującą postać:

Zwierzę	
	ZWIERZE_ID: BIGINT
	ZWIERZE_TYP: VARCHAR
	IMIE: VARCHAR
	WIEK: INTEGER
	LICZBA_LAT: INTEGER
	JAJ_ROCZNIE: INTEGER

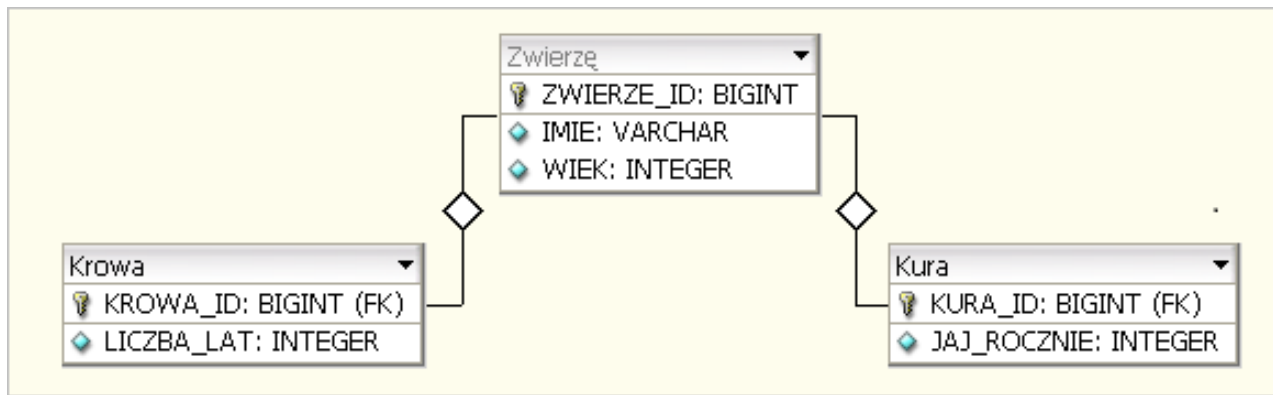


# Dziedziczenie - table per subclass

Wierne odwzorowanie modelu do tabel. Tworzymy tabelę dla nadklasy, oraz po jednej tabeli dla każdej podklasy, przy czym tabela reprezentująca podklasę nie powiela żadnego pola, które wystąpiło w nadklasie.

Klucz w każdej tabeli podklasy jest jednocześnie kluczem obcym tabeli nadklasy.

Tabele w bazie:



# Pobieranie obiektów z bazy

Każda adnotacja mapująca relacje ma atrybut "fetch":

- FetchType.LAZY - relacja nie będzie automatycznie pobierana - np. oznaczając wypożyczenia użytkownika jako LAZY nie będą one pobierane razem z użytkownikiem
- FetchType.EAGER - relacje będzie zainicjowana przez Hibernate albo poprzez wykonanie zapytania JOIN, albo poprzez wygenerowanie dodatkowego zapytania

# Kaskadowość

- Kaskadowość pozwala określić propagację operacji na rekordy w tabelach powiązanych
- Każda adnotacja mapująca relacje ma atrybut "cascade", który pozwala określić kaskadowe zachowanie się tej relacji

# CascadeType

- CascadeType.PERSIST - propagacja jest wykonywana jeśli na encji głównej została wykonana metoda persist
- CascadeType.MERGE - propaguje operacje zmerge'owania
- CascadeType.REFRESH - propaguje operacje odświeżenia
- CascadeType.REMOVE - propaguje operacje usunięcia
- CascadeType.DETACH - propaguje operacje odłączenia obiektów od sesji
- CascadeType.ALL - skrót pozwalający zastosować wszystkie powyższe typy kaskadowości naraz

# Entity Manager

Obiekt służący do wykonywania wszystkich odwołań do bazy danych, podstawowe metody to:

- find - wyszukuje encje na podstawie klucza głównego
- persist - tworzy nowa encje w bazie
- merge - aktualizuje encje
- remove - usuwa encje z bazy
- getReference - podobna do find, ale nie inicjuje obiektu od razu
- createQuery - zwraca obiekt zapytania
- getTransaction - daje dostęp do transakcji związanej z naszą operacją

# Transakcje

Transakcja to operacja typu - "wszystko albo nic"

Przykład: przelew pieniędzy z rachunku A na B

# Hibernate Query Language - HQL

- Dający ogromne możliwości obiektowy język zapytań
- Hibernate tłumaczy HQL do SQL'a
- Zapytania HQL są krótsze i bardziej czytelne niż odpowiedniki SQL
- W zapytaniach HQL posługujemy się nazwami encji i ich atrybutami