

Paweł Rajba

[pawel@ii.uni.wroc.pl](mailto:pawel@ii.uni.wroc.pl)

<http://www.itcourses.eu/>

# NHibernate

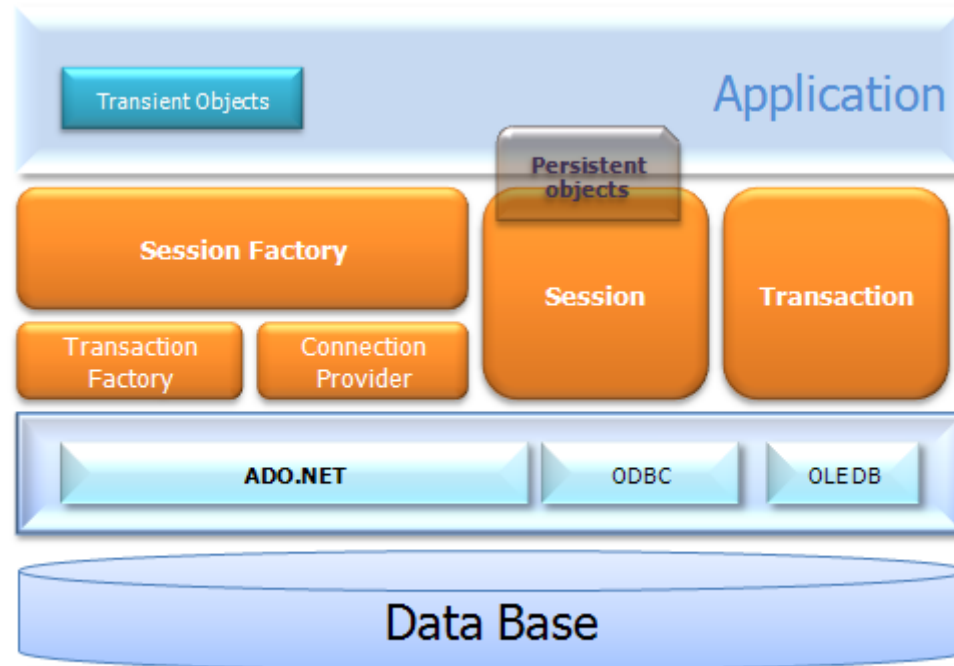
# Agenda

- Architektura
- Trwałość przezroczysta
- Konfiguracja, konfiguracja mapowania
- Dziedziczenie klas
- Kolekcje
- Asocjacje
- Cykl życia obiektów
- Trwałość przechodnia
- Strategie sprowadzania danych
- Pobieranie obiektów
- Cache system
- Stateless session
- Debugging

# Zaczynamy...

- ...od przykładu
  - HelloWorld

# Architektura



# Architektura

- Podstawowe interfejsy
  - ISession
  - ISessionFactory
    - ISession vs. ISessionFactory
  - Configuration
  - Transaction
  - IQuery
  - ICriteria

# Trwałość przezroczysta

- Polega na tym, że utrwalane klasy nie są świadome utrwalania
  - Kontrwzorzec: ActiveRecord
- NHibernate jest (prawie) przezroczysty
  - Nie trzeba nic implementować ani nic dziedziczyć
  - Za utrwalanie odpowiada menedżer trwałości: interfejsy Session i Query
  - Gdzie jest to prawie?
    - Pola muszą być virtual
    - Czasami typy muszą pochodzić z np. Iesi.Collection

# Konfiguracja

- Mamy dwa główne typy konfiguracji
  - Dostęp do źródła danych
    - Plik hibernate.cfg.xml
    - Pliki mapowania pomiędzy obiektami i tabelkami

# Konfiguracja

- Plik hibernate.cfg.xml
  - Można wstawić zawartość bezpośrednio do web.config
  - Można też utworzyć zewnętrzny plik o innej nazwie i dać referencję z web.config

```
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2" >  
  <session-factory>  
    <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>  
    <property name="connection.connection_string">Server=.;initial catalog=Test;Integrated Security=SSPI</property>  
    <property name="show_sql">>false</property>  
    <property name="dialect">NHibernate.Dialect.MsSql2008Dialect</property>  
  </session-factory>  
</hibernate-configuration>
```



# Konfiguracja

- Mapowanie pomiędzy obiektami i tabelkami
    - W zewnętrznych plikach XML
    - Poprzez atrybuty .NET
    - Za pomocą biblioteki Fluent
    - Poprzez Mapping By Code
  - Narzędzia wspierające potrafiące wygenerować
    - Pliki odwzorowań na podstawie bazy danych
    - Schemat bazy danych na podstawie opisu metadanych
    - Pliki klas na podstawie plików odwzorowań
    - Pliki klas na podstawie bazy danych
- Jednak jak i kiedy tego używać?

# Konfiguracja mapowania

- Podstawowe elementy
  - Atrybuty
    - Nazwy właściwości, kolumny, typy, operacje (update, insert)
    - Jeśli wielkość litera ma znaczenie → apostrofy
  - Właściwości wyliczane
    - Definiujemy podając atrybut formuła
    - Wartość atrybutu to wyrażenie SQL
    - Dla takiej właściwości nie jest tworzona kolumna w BD

# Konfiguracja mapowania

- Podstawowe elementy
  - Tożsamość obiektów
    - Klucz może być jednokolumnowy lub wielokolumnowy
    - Generatory identyfikatorów
      - identity
        - Generuje identyfikatory typów long, short, int
        - Wspiera m.in. DB2, MySQL, MS SQL Server, Sybase
      - sequence
        - Generuje identyfikatory typów long, short, int
        - Wykorzystuje sekwencje m.in. w DB2, PostgreSQL, Oracle, SAP DB

# Konfiguracja mapowania

- Generatory identyfikatorów c.d.
  - hilo
    - Generuje identyfikatory typów long, short, int
    - Bazuje na dwóch wartościach hi i lo, dostarczy pakiety Idków
  - guid.comb
    - Generuje GUIDy
    - Sortowanie daje sekwencję dodawania
  - native
    - wybiera jeden z generatorów identity, sequence i hilo w zależności od możliwości bazy danych
- Dobry identyfikator
  - Syntetyczny
  - Uporządkowany
  - Non-insertable

# Konfiguracja mapowania

- Podstawowe elementy
  - Komponenty
    - Kilka klas po stronie modelu mapujemy na jedną tabelę po stronie bazy danych
  - Enumeracje
    - Pozwala na przechowywanie czytelnych napisowych wartości po stronie bazy danych

# Przykłady

- FactoryCreatingCost
- AttributeMapping
- FluentMappings
- MappingByCodeExample
- UsingComponents
- UsingEnumerations
- CompositeKeys1
- CompositeKeys2
- IdentityMap

# Dziedziczenie klas

- Tabela na klasę
- Tabela na hierarchie klas
- Tabela na podklasę

# Dziedziczenie klas

- Tabela na klasę
  - Wszystkie właściwości klasy są w tabeli odpowiadającej tej klasie
    - łącznie z właściwościami dziedziczonymi)
  - Zaleta: jeśli zapytanie dotyczy jednej klasy, wykona się szybko i łatwo je skonstruować
  - Wady
    - Modyfikacja jednego pola implikuje zmiany we wszystkich tabelach powiązanych
    - Problem jest także operacjach polimorficznych



# Dziedziczenie klas

- Tabela na hierarchie klas
  - Rozwiązanie polega zastosowaniu jednej tabeli dla hierarchii klas powiązanych relacją dziedziczenia
  - Dodatkowo jest kolumna dyskryminatora, który określa jakiego typu jest dany wpis w tabeli
  - Zaleta: Łatwo zadawać zapytania zwykłe jak i te oparte na polimorfizmie
  - Wady: Zaburzona normalizacja, w każdym wierszu sporo wartości pustych, problem z NULLami

# Dziedziczenie klas

- Tabela na podklasę
  - W tym modelu każdy byt (klasy, w tym abstrakcyjne, interfejsy) mają swoje tabele
  - W tabelach tych są tylko właściwości zdefiniowane w danej klasie lub interfejsie
  - Jeżeli klasa ma podklasę, wtedy jej klucz główny jest jednocześnie obcym do nadklasy i tam znajduje się reszta danych danego obiektu

# Dziedziczenie klas

- Tabela na podklasę
  - Zalety
    - Pełna normalizacja schematu w bazie danych
    - Wsparcie dla polimorfizmu
  - Wady
    - Przede wszystkim jedna: przy większej strukturze, duża złożoność obsługi (trzeba wykonywać dużo złączeń)

# Dziedziczenie klas

- Wybór strategii
  - W zasadzie wybór jest pomiędzy opcjami 2 i 3
  - Kiedy co wybrać?

# Przykłady

- Inheritance1TablePerClass
- Inheritance2TablePerClassHierarchy
- Inheritance3TablePerSubclass

# Kolekcje

- Podstawowe rodzaje kolekcji:
  - Set – zbiór
  - Bag – wielozbiór
  - Map – struktura asocjacyjna
  - List – struktura indeksowana, czyli „kolejność ma znaczenie”
- Jakie typy po stronie .NET?
  - Znaczenie biblioteki `System.Collections`

# Przykłady

- Collections1
- Collections2
- Collections3
- Collections4
- Collections5

# Asocjacje

- Reprezentują związki między obiektami, które mają swoje odbicie w związkach między tabelami
- Często są dosyć kłopotliwym elementem do obsłużenia
- Asocjacje mogą być
  - Jednokierunkowe
  - Dwukierunkowe



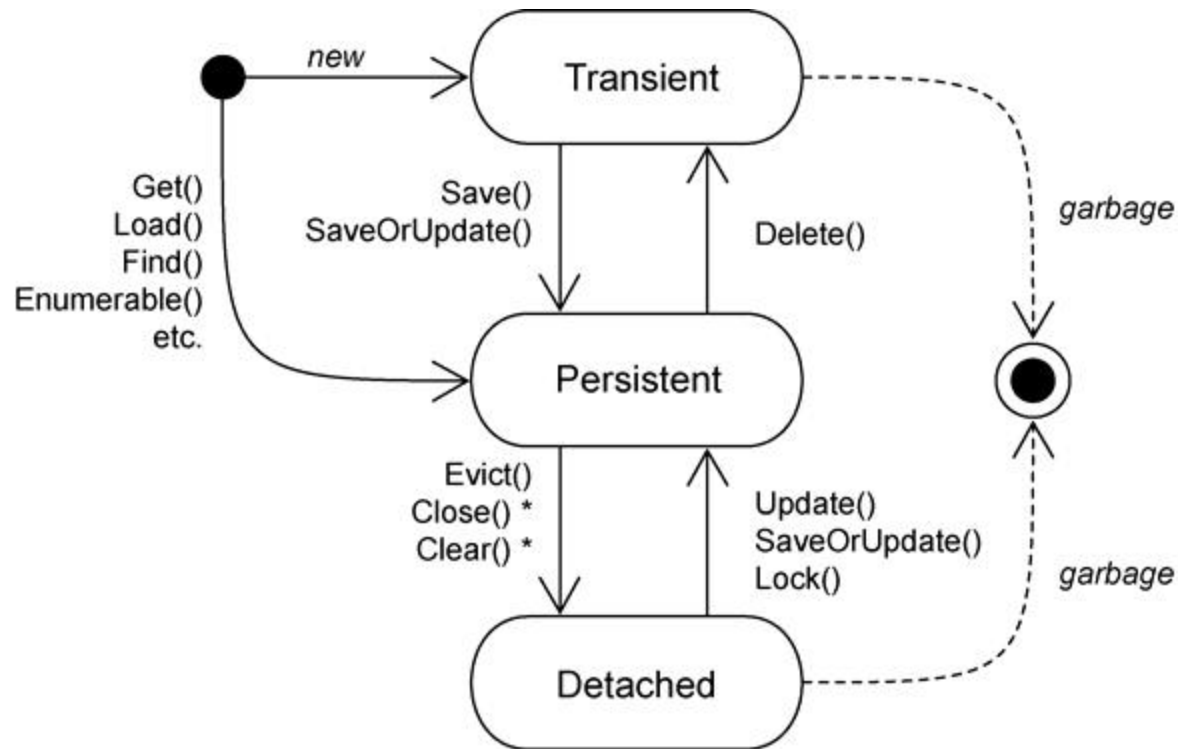
# Asocjacje

- Rodzaje asocjacji
  - jeden-do-jednego
  - wiele-do-jednego
  - jeden-do-wielu
  - wiele-do-wielu

# Przykłady

- Associations1ManyToOne
- Associations2ManyToMany

# Cykl życia obiektów



\* affects all instances in a Session

# Przykład

---

- UsingObjects

# Trwałość przechodnia

- Realizowana przez opcję cascade
- Bazuje na asocjacjach
- Dostępne wartości
  - none, save-update, delete, all, delete-orphan, all-delete-orphan
- Wartości można łączyć
  - Np. cascade="save-update, delete,,

# Przykład

---

- PersistenceByReachability

# Strategie sprowadzania danych

- NHibernate implementuje następujące strategie sprowadzania danych:
  - Sprowadzanie natychmiastowe, ang. *immediate fetching*
  - Sprowadzanie leniwe, ang. *lazy fetching*
  - Sprowadzanie wyprzedzające, ang. *eager (outer join) fetching*
  - Sprowadzanie wsadowe, ang. *batch fetching*
- Do określenia strategii mamy atrybuty w XML:
  - *lazy, extra, fetch, batch-size*
- Ciekawe rozważania:
  - <http://ayende.com/blog/3943/nhibernate-mapping-set>

# Strategie sprawadzania danych

- Sprawadzanie leniwe
  - Najczęściej stosowane
  - Ryzyko wiąże się z koniecznością jednoczesnego pobrania, znane jako N+1 antipattern

```
// SELECT * FROM Posts
foreach (Post post in session.CreateQuery("from Post").List())
{
    //lazy loading of comments list causes:
    // SELECT * FROM Comments where PostId = @p0
    foreach (Comment comment in post.Comments)
    {
        //print comment...
    }
}
```

Warto zajrzeć:

<http://www.hibernateingrhinos.com/products/nhprof/learn/alert/selectnplusone>



# Strategie sprowadzania danych

- Sprowadzanie wyprzedzające
  - Do pobrania obiektów wykorzystywany jest OUTER JOIN
  - W pewnych scenariuszach jest bardziej optymalne od sprowadzania leniwego
  - Można włączyć na poziomie transakcji, chociaż częściej określa się w pliku mapującym
  - Parametr *max\_fetch\_depth* w pliku hibernate.cfg.xml określamy, ile tabel może być maksymalnie złączanych
    - domyślnie: 1

# Strategie sprowadzania danych

- Sprowadzanie wsadowe
  - Nie jest to osobna strategia sprowadzania, tylko mechanizm przyspieszający działanie sprowadzania leniwego i natychmiastowego.
  - Zamiast podawać w klauzuli WHERE pojedynczy identyfikator, NHibernate zbierze ich więcej i poda raz cały zestaw identyfikatorów

# Przykład

---

- FetchingStrategies

# Pobieranie obiektów

- Poprzez identyfikator
- Hibernate Query Language (HQL)
- Query By Criteria (QBC)
- Query By Example (QBE)
- LINQ to NHibernate
- Natywny SQL

# Pobieranie obiektów

- Poprzez identyfikator
  - Get
    - `var user = session.Get<User>( userID )`
    - Jeśli nie ma obiektu, zwraca NULL
    - Zwraca zawsze obiekt (nie proxy), nie działa LAZY
  - Load
    - `var user = session.Load<User>( userID )`
    - Jeśli nie ma obiektu, rzuca wyjątek
    - Współpracuje z LAZY

# Pobieranie obiektów

- Hibernate Query Language
  - Obiektowy język zapytań podobny do SQL-a
  - Oparty o napisy (ma swoje wady i zalety)
  - Tłumaczony do SQL-a

```
IQuery q = session.createQuery("from User u where u.firstname = :fname");  
q.setString("fname", "Arnold");  
IList<User> result = q.list<User>();
```

# Pobieranie obiektów

- Query By Criteria

- Obiektowy język zapytań, kompilowany

```
ICriteria criteria = session.CreateCriteria( typeof( User ) );  
criteria.Add( Expression.Like("firstname", "Alfred") );  
IList<User> result = criteria.List<User>();
```

- Query By Example

- Pozwala na podanie przykładu jako wzorca do wyszukiwania

```
User exampleUser = new User { FirstName = "Alfred" };  
ICriteria criteria = session.CreateCriteria( typeof( User ) );  
criteria.Add( Example.Create(exampleUser) );  
IList<User> result = criteria.List<User>();
```

# Pobieranie obiektów

## ■ Natywny SQL

- `IList<Produkt> produkty =  
    session.CreateSQLQuery("select {p.*} from PRODUKT {p}")  
    .AddEntity("p", typeof(Produkt)).List<Produkt>();`

## ■ LINQ to NHibernate

- `from k in session.Query<Koszyk>()  
    select k;`

## ■ LINQ to NHibernate

- `session.QueryOver<Koszyk>().List();`



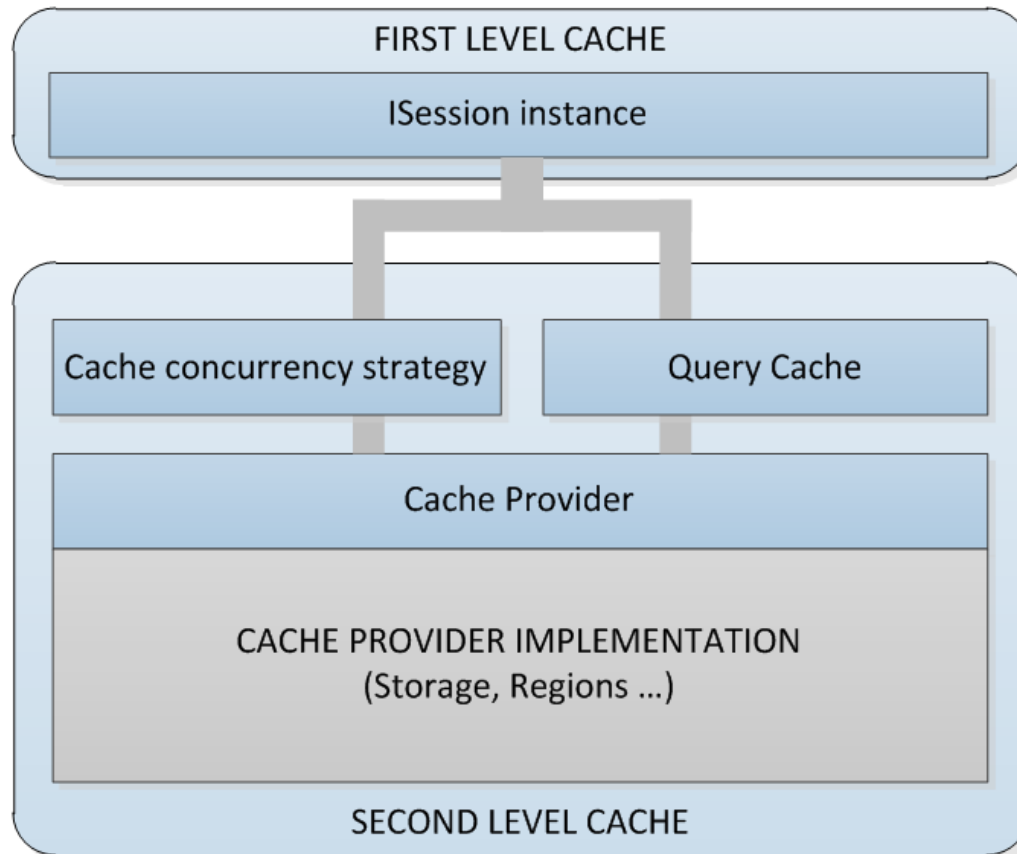
# Przykład

---

- QueryingDatastore

# Cache system

- Architektura



# Cache system

- Cache poziomu 1
  - Powiązany z obiektem ISession
  - Obiekt znajdzie się w cache po wywołaniu metod:
    - Get, Load, Save, Update, Delete, SaveOrUpdate  
Uwaga: zapytania HQL, itd. Nie wstawiają obiektów do cache
  - Cykl życia związany z transakcją oraz następującymi metodami:
    - Session.Clear, Session.Flush, Session.Evict, ...

# Cache system

- Cache poziomu 2
  - Powiązany z SessionFactory i dane współdzielone przez wszystkie sesje
  - Dostępnych jest wielu providerów dla cache 2 poziomu:
    - Proces: HashTable, Prevalence, SysCache2
    - Rozproszony: NCache, Velocity
  - Dane pamiętane są w postaci słownika
    - Tutaj mogą być cachowane zapytania, ale tylko ID
  - Mechanizm śledzenia timestampów, generalnie są polityki cache expiration
  - Niektóre providera wspierają SQL Server notifications
  - Cache może być
    - Read-write, Read, Nonstrict-read-write, transactional

# Cache system

- Cache poziomu 2, regiony
  - Pozwala na definiowanie obszarów dla zbioru klas
  - Możemy stosować osobne różne ustawienia, np. expiration policy
- Warto poczytać:
  - <http://blog.raffaeu.com/archive/2011/12/29/nhibernate-cache-system-part-1.aspx>
  - <http://blog.raffaeu.com/archive/2011/12/29/nhibernate-cache-system-part-2.aspx>
  - <http://ayende.com/blog/3112/nhibernate-and-the-second-level-cache-tips>

# Stateless session

- Stosowane przy masowych operacjach update, delete, save
- Bardziej bezpośredni dostęp do bazy
  - Bez cache 1 i 2 poziomu
  - Nie ma lazy loading
  - Nie ma kaskad
  - Bez śledzenia zmian, transakcje bardzo ograniczone

# Debugging

- Drukowanie na konsolę (opcja `show_sql`)
- Logging (NLog)
- SQL Profiler
- NHibernate Profiler