

LEKCE 06

Generika + Repository pattern

Zpětná vazba 🙏

Lekce 06

1. Generika
2. Repository pattern
3. Mockování v praxi

Generika

Což takhle rovnou příklad?

Definice generiky

- ≈ Generické typy nejsou specifické k určitému typu
- ≈ Parametrizované typy

Co je to generika?

- Generické mohou být třídy, metody, rozhraní...
- Bere ≥ 1 typových parametrů
- Nejlepší kombinace:
 - **přepoužitelnosti** - nemusíme duplikovat kód pro různé typy
 - **silného typování** - odlišné typy vyhodí kompilační chybu
 - **výkonu** - neboxujeme hodnotové typy

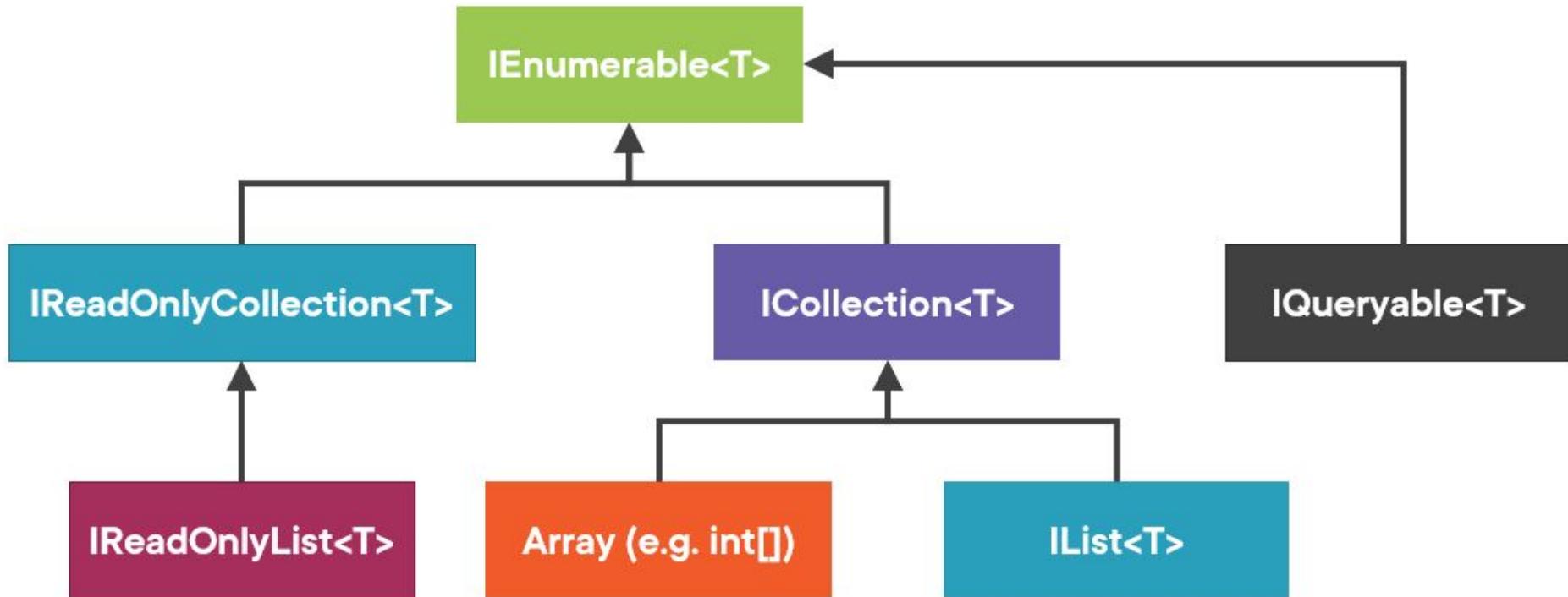
Where omezení

- Nechceme vždy povolovat všechny typy
- Silnější typování
- Můžeme využívat metody constrained typu

Where omezení

Constraint	Description
<code>where T : struct</code>	The type argument must be a non-nullable value type, which includes <code>record</code> <code>struct</code> types. For information about nullable value types, see Nullable value types . Because all value types have an accessible parameterless constructor, either declared or implicit, the <code>struct</code> constraint implies the <code>new()</code> constraint and can't be combined with the <code>new()</code> constraint. You can't combine the <code>struct</code> constraint with the <code>unmanaged</code> constraint.
<code>where T : class</code>	The type argument must be a reference type. This constraint applies also to any class, interface, delegate, or array type. In a nullable context, <code>T</code> must be a non-nullable reference type.
<code>where T : new()</code>	The type argument must have a public parameterless constructor. When used together with other constraints, the <code>new()</code> constraint must be specified last. The <code>new()</code> constraint can't be combined with the <code>struct</code> and <code>unmanaged</code> constraints.
<code>where T : <base class name></code>	The type argument must be or derive from the specified base class. In a nullable context, <code>T</code> must be a non-nullable reference type derived from the specified base class.
<code>where T : <interface name></code>	The type argument must be or implement the specified interface. Multiple interface constraints can be specified. The constraining interface can also be generic. In a nullable context, <code>T</code> must be a non-nullable type that implements the specified interface.
<code>where T : U</code>	The type argument supplied for <code>T</code> must be or derive from the argument supplied for <code>U</code> . In a nullable context, if <code>U</code> is a non-nullable reference type, <code>T</code> must be a non-nullable reference type. If <code>U</code> is a nullable reference type, <code>T</code> can be either nullable or non-nullable.
<code>where T : default</code>	This constraint resolves the ambiguity when you need to specify an unconstrained type parameter when you override a method or provide an explicit interface implementation. The <code>default</code> constraint implies the base method without either the <code>class</code> or <code>struct</code> constraint. For more information, see the default constraint spec proposal .

Generické kolekce

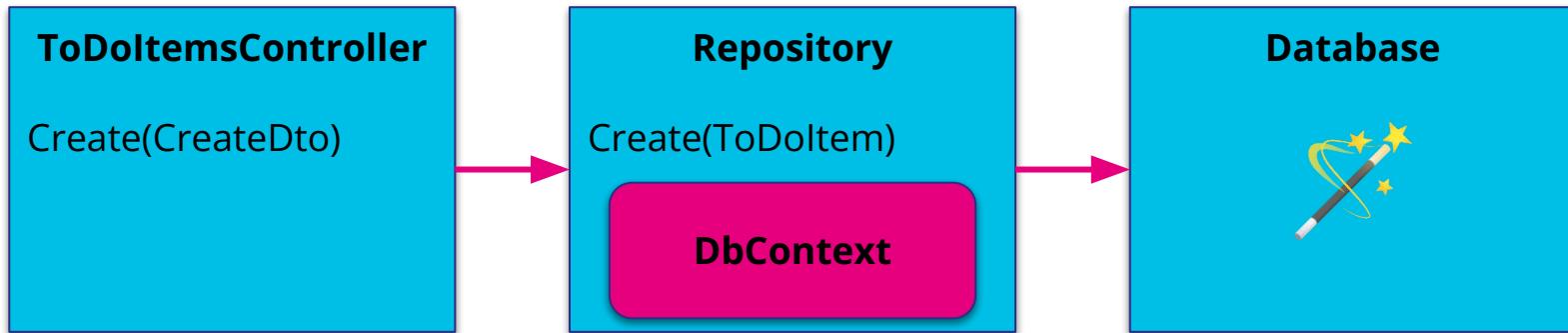
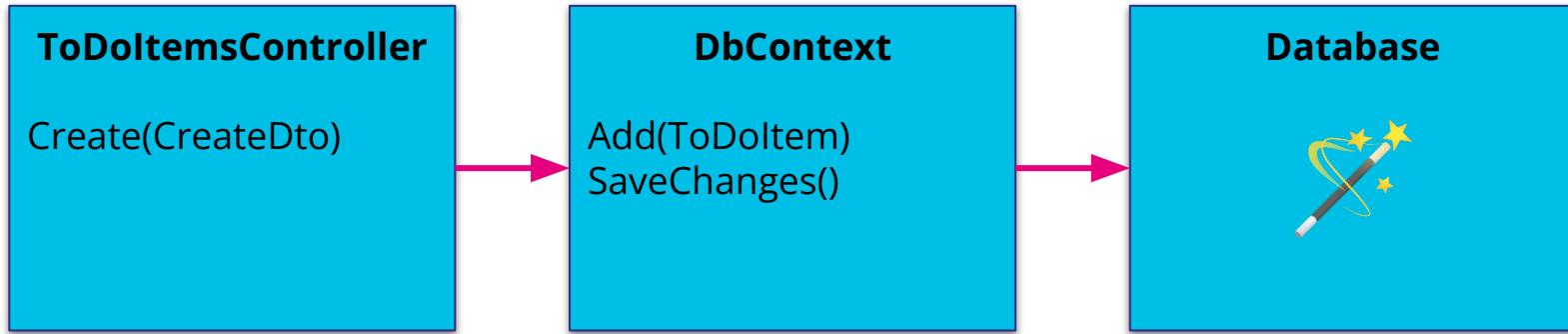


ArrayList

Všeho s mírou!

Repository

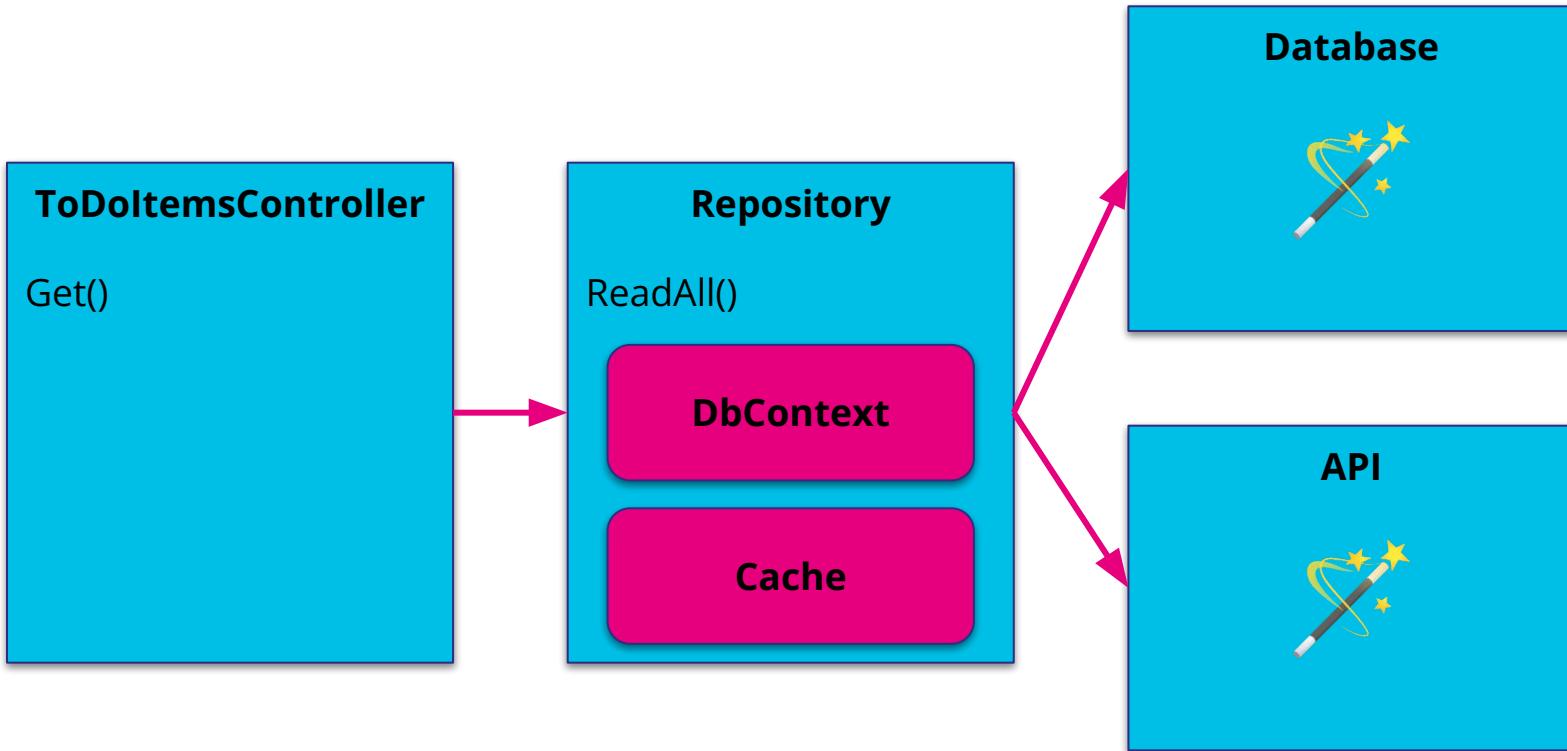
Repository pattern



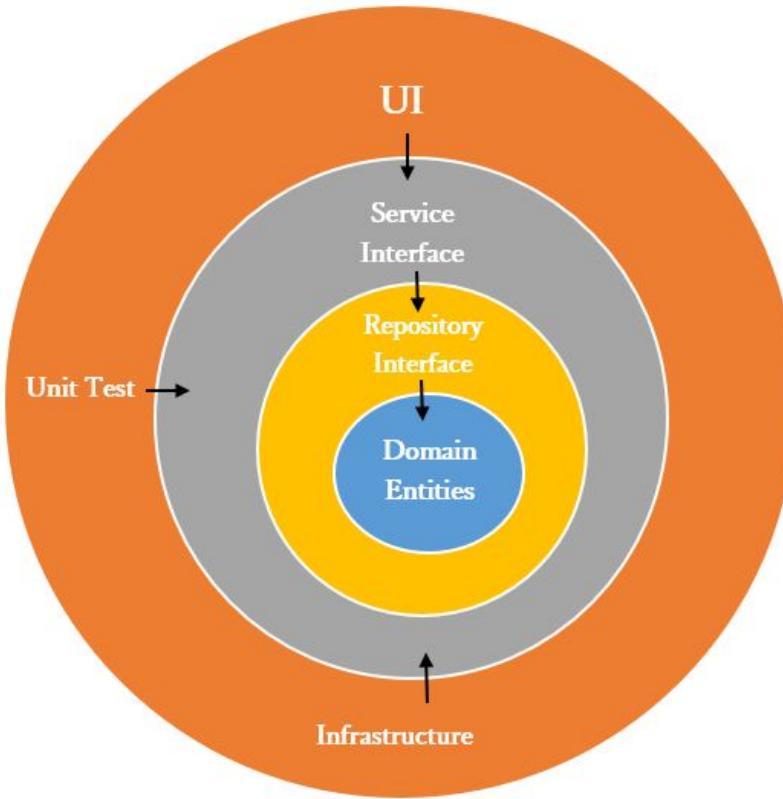
K čemu Repository pattern?

- Přepoužitelnost
- Centralizovaná logika
- Lepší testování
- “Oddělená” datová vrstva
- Může reprezentovat několik datových zdrojů
- Jednodušší vyměnit datový zdroj

Repository pro několik zdrojů



Cibulová architektura



IRepository<T>

Mocking

Co je to mockování?

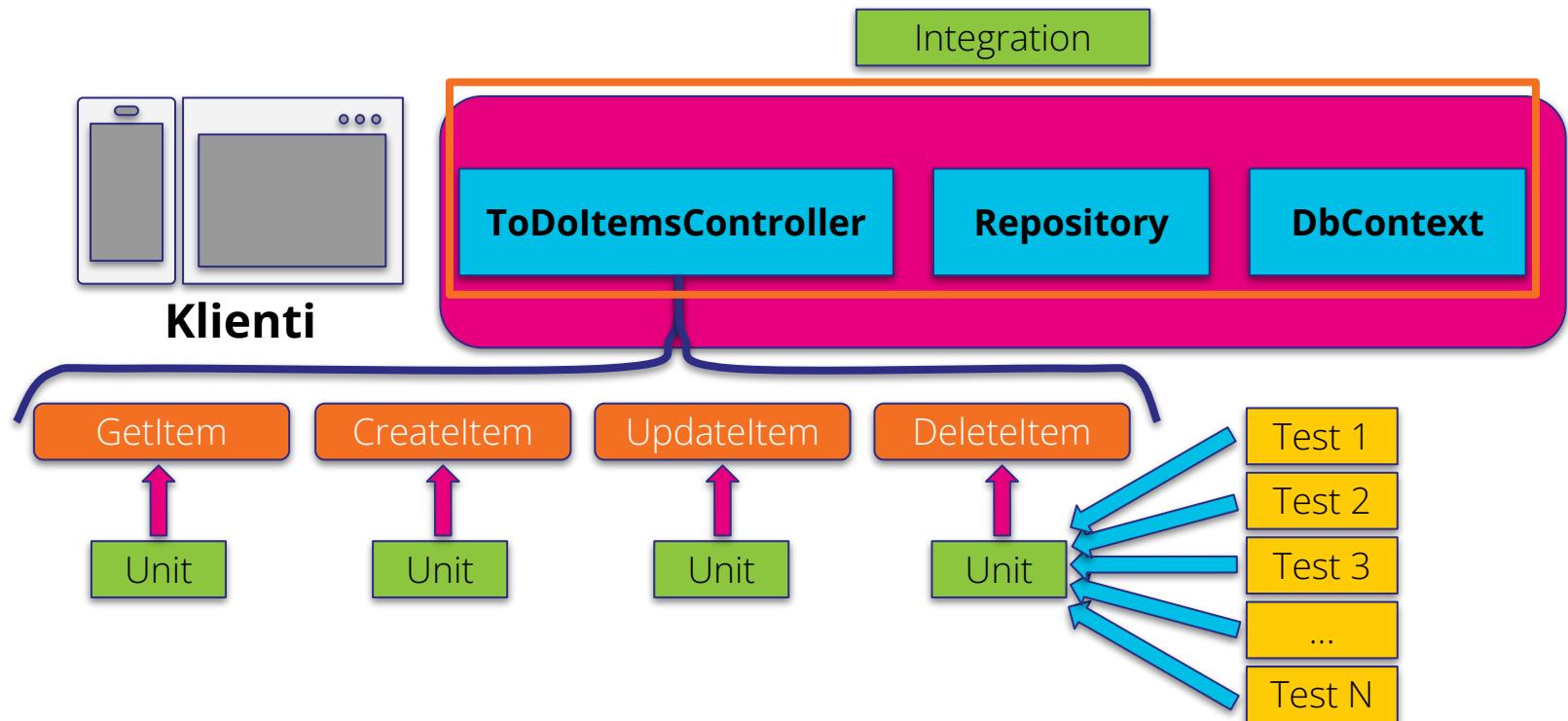
Mockování je proces vytváření objektů (Mocků), které simulují chování reálných objektů.

Tyto Mocky lze v testovacím prostředí upravovat a nastavovat jejich chování.

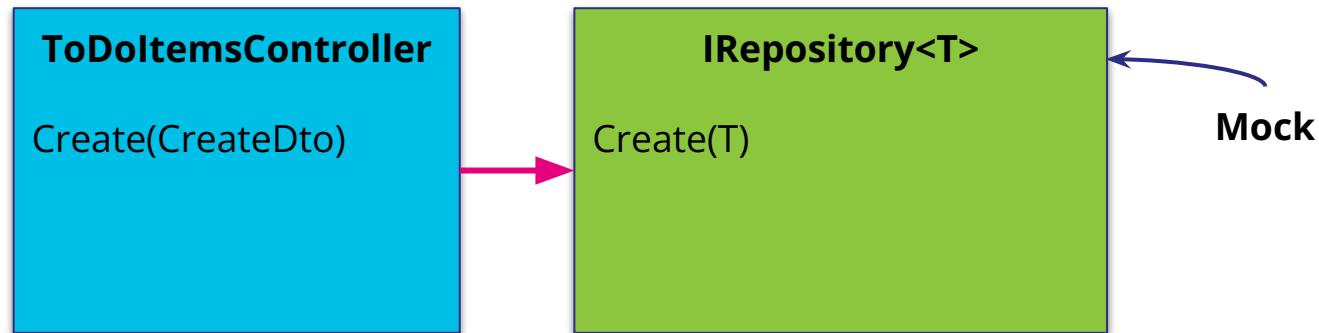
Výhody mockování

- Nemusíme testovat DB, jen naší logiku
- DB nám nerozbije testy
- Testy nespadnou při paralelním běhu

Unit Testing?



Mockování



Side effects

