Architektura aplikací

Pavel Mička

Obsah

1. Architektura aplikací
2. Vrstevnatá architektura
3. Hexagonální architektura
4. Rich domain model
5. Data transfer objects

Architektura aplikací

- Vždy se volí s ohledem na aplikaci
- Základní pilíře
  - **Abstraction** — Aplikace zakrývá nepodstatné detaily a její funcionalita volí patřičnou míru zobecnění problému
  - **Refinement** — Design je tvořen od nejvyšších bloků postupným zjemňováním až na úroveň příkazů programovacího jazyka
  - **Modularity** — Software se skládá z nahraditelných modulů
  - **Information hiding** — Každý modul o sobě prozradí pouze nezbytné množství informací

Architektura aplikací

- Vždy se volí s ohledem na aplikaci
- Základní pilíře
  - **Abstraction** — Aplikace zakrývá nepodstatné detaily a její funcionalita volí patřičnou míru zobecnění problému
  - **Refinement** — Design je tvořen od nejvyšších bloků postupným zjemňováním až na úroveň příkazů programovacího jazyka
  - **Modularity** — Software se skládá z nahraditelných modulů
  - **Information hiding** — Každý modul o sobě prozradí pouze nezbytné množství informací

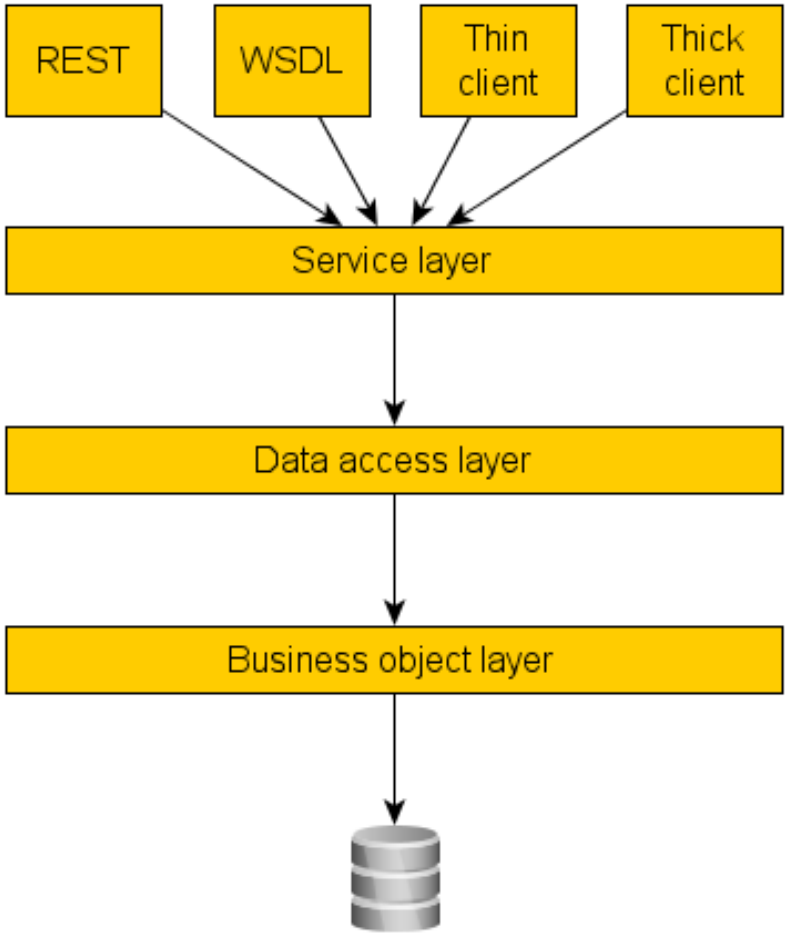
Architektura aplikací

- Vždy se volí s ohledem na aplikaci
- Základní pilíře
  - **Abstraction** — Aplikace zakrývá nepodstatné detaily a její funcionalita volí patřičnou míru zobecnění problému
  - **Refinement** — Design je tvořen od nejvyšších bloků postupným zjemňováním až na úroveň příkazů programovacího jazyka
  - **Modularity** — Software se skládá z nahraditelných modulů
  - **Information hiding** — Každý modul o sobě prozradí pouze nezbytné množství informací

Architektura aplikací

- Vždy se volí s ohledem na aplikaci
- Základní pilíře
  - **Abstraction** — Aplikace zakrývá nepodstatné detaily a její funcionalita volí patřičnou míru zobecnění problému
  - **Refinement** — Design je tvořen od nejvyšších bloků postupným zjemňováním až na úroveň příkazů programovacího jazyka
  - **Modularity** — Software se skládá z nahraditelných modulů
  - **Information hiding** — Každý modul o sobě prozradí pouze nezbytné množství informací

Vrstevnatá architektura I.



Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

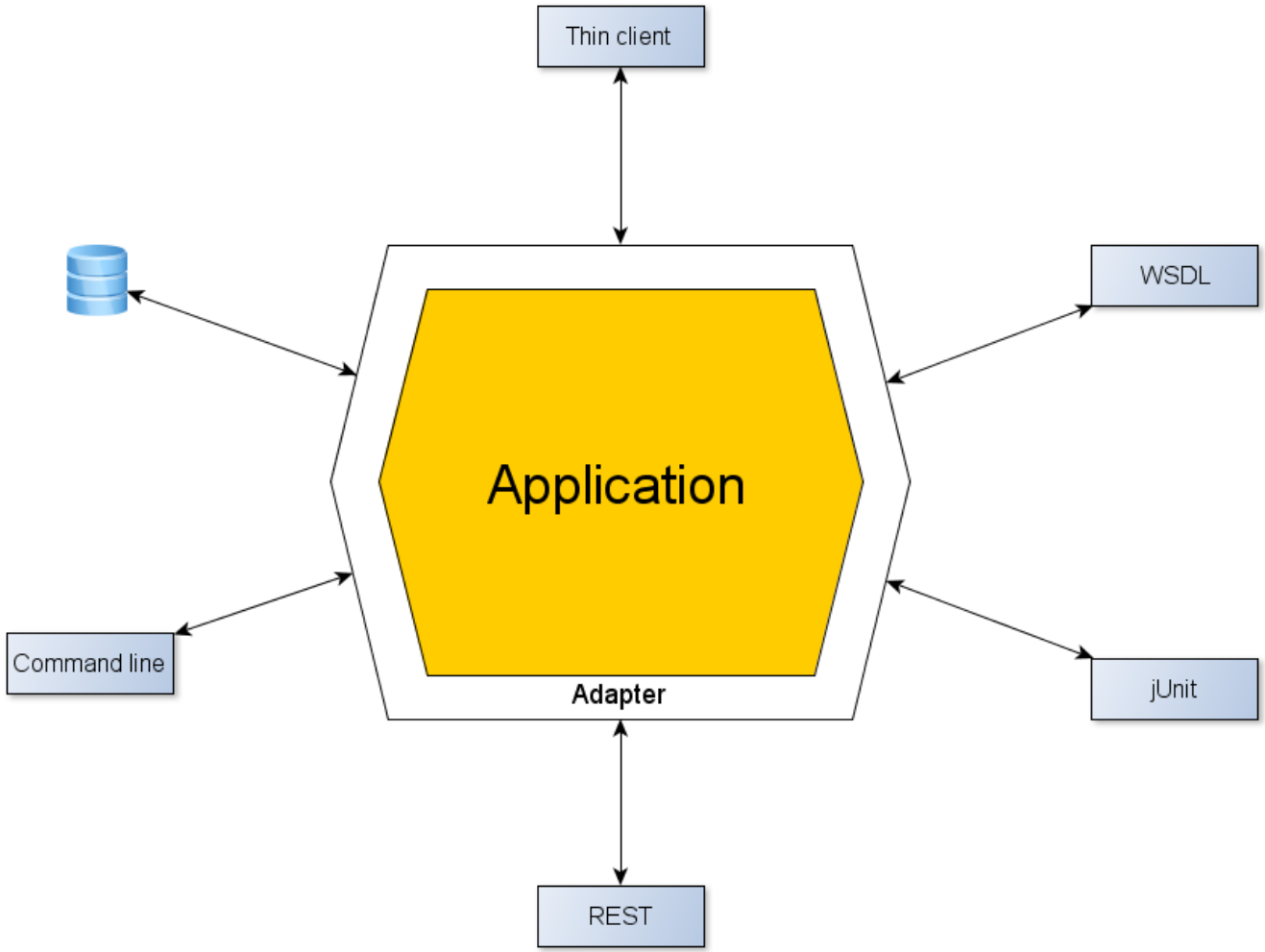
Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Vrstevnatá architektura II.

- *Layers* pattern
- Rozdělení odpovědností (*Separation of concerns*)
- Snížení závislostí v software
- Vyšší vrstva provolává pouze vrstvu pod sebou
- V relaxované verzi může vyšší vrstva provolávat kteroukoliv z nižších vrstev
- Asymetrie shora dolů
- Výsledná architektura silně zavání transakčním skriptem
- Proč striktně oddělovat v okamžiku, kdy používáme rozhraní a dependency injection container?

Hexagonální architektura I.



Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura II.

- *Ports and adapters* pattern
- Vzor vymyslel *Alistair Cockburn*
- Pohled na aplikaci jako modul
- Protected variations ve velkém
- Systém msuí být stejně dobře řiditelný strojem jako člověkem (např. testování)
- Velmi podobné vzoru *Service layer* (P of EAA)

Hexagonální architektura III.

- Modul je stíněn svým rozhraním (sada portů)
- Vnější technologie/aplikace přistupují k portům pomocí adaptérů
- Asymetrie mezi vnitřkem a vnějškem aplikace
- Aplikace se vnitřně může skládat z mnoha dalších hexagonů

Hexagonální architektura III.

- Modul je stíněn svým rozhraním (sada portů)
- Vnější technologie/aplikace přistupují k portům pomocí adaptérů
- Asymetrie mezi vnitřkem a vnějškem aplikace
- Aplikace se vnitřně může skládat z mnoha dalších hexagonů

Hexagonální architektura III.

- Modul je stíněn svým rozhraním (sada portů)
- Vnější technologie/aplikace přistupují k portům pomocí adaptérů
- Asymetrie mezi vnitřkem a vnějškem aplikace
- Aplikace se vnitřně může skládat z mnoha dalších hexagonů

Hexagonální architektura III.

- Modul je stíněn svým rozhraním (sada portů)
- Vnější technologie/aplikace přistupují k portům pomocí adaptérů
- Asymetrie mezi vnitřkem a vnějškem aplikace
- Aplikace se vnitřně může skládat z mnoha dalších hexagonů

Rich domain model

- Rich domain je opositum k antipatternu *Anemic domain model*
- Anemický datový model nemá v entitách žádné skutečné metody (pouze gettery a settery)
- Anemický datový model odporuje objektově orientovanému programování (protože *objekt == data + operace*)
- Anemický datový model vede na transakční skript — programování s objekty
- **Bohatý datový model by měl být samozřejmostí**, bohužel tomu tak není kvůli nepochopení objektových principů, mizerné podpoře injekce do entit ze strany některých frameworků a vyšší náročnosti na implementaci

Rich domain model

- Rich domain je opositum k antipatternu *Anemic domain model*
- Anemický datový model nemá v entitách žádné skutečné metody (pouze gettery a settery)
- Anemický datový model odporuje objektově orientovanému programování (protože *objekt == data + operace*)
- Anemický datový model vede na transakční skript — programování s objekty
- **Bohatý datový model by měl být samozřejmostí**, bohužel tomu tak není kvůli nepochopení objektových principů, mizerné podpoře injekce do entit ze strany některých frameworků a vyšší náročnosti na implementaci

Rich domain model

- Rich domain je opositum k antipatternu *Anemic domain model*
- Anemický datový model nemá v entitách žádné skutečné metody (pouze gettery a settery)
- Anemický datový model odporuje objektově orientovanému programování (protože *objekt* == *data* + *operace*)
- Anemický datový model vede na transakční skript — programování s objekty
- **Bohatý datový model by měl být samozřejmostí**, bohužel tomu tak není kvůli nepochopení objektových principů, mizerné podpoře injekce do entit ze strany některých frameworků a vyšší náročnosti na implementaci

Rich domain model

- Rich domain je opositum k antipatternu *Anemic domain model*
- Anemický datový model nemá v entitách žádné skutečné metody (pouze gettery a settery)
- Anemický datový model odporuje objektově orientovanému programování (protože *objekt* == *data* + *operace*)
- Anemický datový model vede na transakční skript — programování s objekty
- **Bohatý datový model by měl být samozřejmostí**, bohužel tomu tak není kvůli nepochopení objektových principů, mizerné podpoře injekce do entit ze strany některých frameworků a vyšší náročnosti na implementaci

Rich domain model

- Rich domain je opositum k antipatternu *Anemic domain model*
- Anemický datový model nemá v entitách žádné skutečné metody (pouze gettery a settery)
- Anemický datový model odporuje objektově orientovanému programování (protože *objekt* == *data* + *operace*)
- Anemický datový model vede na transakční skript — programování s objekty
- **Bohatý datový model by měl být samozřejmostí**, bohužel tomu tak není kvůli nepochopení objektových principů, mizerné podpoře injekce do entit ze strany některých frameworků a vyšší náročnosti na implementaci

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects I.

- Data transfer objekty (DTO) odpovídají na otázku, jakým způsobem přenášet data mezi aplikací a jejími externími součástmi (např. service layer → user interface)
- Pro tento účel nelze používat entity (business objekty)
  - Transakční kontext končí na servisní vrstvě
  - Entity mohou obsahovat citlivá data (hash hesla, salt, rodné číslo)
  - Zabezpečení volání metod je taktéž na servisní vrstvě
  - Došlo by k odhalení vnitřního rozhraní aplikace (tj. porušení protected variations)
  - Entity nemohou být přeneseny do jiné VM — případné injektované závislosti neprojdou serializací a deserializací (*null pointer exception*)

Data transfer objects II.

- Pro přenos proto používáme jednoúčelové přepravky DTO (pouze gettery a settery)
- DTO v případě referencí obsahují pouze identifikátor
  - Nemohou snadno obsahovat jiné DTO, protože by bylo třeba vyřešit, jak až daleko lze traverzovat
- Kód DTO lze bez obav poskytnout třetím stranám pro usnadnění tvorby third-party aplikací
- Nevýhodou může být značná duplikace kódu mezi DTO a Entitami (lze vyřešit generováním kódu a poloautomatickým mapováním)

Data transfer objects II.

- Pro přenos proto používáme jednoúčelové přepravky DTO (pouze gettery a settery)
- DTO v případě referencí obsahují pouze identifikátor
  - Nemohou snadno obsahovat jiné DTO, protože by bylo třeba vyřešit, jak až daleko lze traverzovat
- Kód DTO lze bez obav poskytnout třetím stranám pro usnadnění tvorby third-party aplikací
- Nevýhodou může být značná duplikace kódu mezi DTO a Entitami (lze vyřešit generováním kódu a poloautomatickým mapováním)

Data transfer objects II.

- Pro přenos proto používáme jednoúčelové přepravky DTO (pouze gettery a settery)
- DTO v případě referencí obsahují pouze identifikátor
  - Nemohou snadno obsahovat jiné DTO, protože by bylo třeba vyřešit, jak až daleko lze traverzovat
- Kód DTO lze bez obav poskytnout třetím stranám pro usnadnění tvorby third-party aplikací
- Nevýhodou může být značná duplikace kódu mezi DTO a Entitami (lze vyřešit generováním kódu a poloautomatickým mapováním)

Data transfer objects II.

- Pro přenos proto používáme jednoúčelové přepravky DTO (pouze gettery a settery)
- DTO v případě referencí obsahují pouze identifikátor
  - Nemohou snadno obsahovat jiné DTO, protože by bylo třeba vyřešit, jak až daleko lze traverzovat
- Kód DTO lze bez obav poskytnout třetím stranám pro usnadnění tvorby third-party aplikací
- Nevýhodou může být značná duplikace kódu mezi DTO a Entitami (lze vyřešit generováním kódu a poloautomatickým mapováním)

Data transfer objects II.

- Pro přenos proto používáme jednoúčelové přepravky DTO (pouze gettery a settery)
- DTO v případě referencí obsahují pouze identifikátor
  - Nemohou snadno obsahovat jiné DTO, protože by bylo třeba vyřešit, jak až daleko lze traverzovat
- Kód DTO lze bez obav poskytnout třetím stranám pro usnadnění tvorby third-party aplikací
- Nevýhodou může být značná duplikace kódu mezi DTO a Entitami (lze vyřešit generováním kódu a poloautomatickým mapováním)

Zdroje a literatura

- Layered architecture

  `http://msdn.microsoft.com/en-us/library/ff650258.aspx`
- Hexagonal architecture

  `http://alistair.cockburn.us/Hexagonal+architecture`
- Service layer

  `http://martinfowler.com/eaaCatalog/serviceLayer.html`
- Anemic domain model

  `http://martinfowler.com/bliki/AnemicDomainModel.html`