en SOLID grund för Clean Code

#DevLin2012

15 Mars 2012

Torbjörn ”Tobbe” Gyllebring
@drunkcod

tobbe@cint.com

Översikt

- SOLID?
- **S**ingle Responsibility Principle
- **O**pen Closed Principle
- **L**iskow Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Historik ... *gäsp*

SOLID?

- Ett antal **PRINCIPER**
- Används reaktivt eller för att förklara olika krafter som verkar på ett system
- Upptäck och validera behov, spekulera inte
- Spekulation leder till ONÖDIG KOMPLEXITET!
- Föredra global över lokal enkelhet

Och jag ska bry mig därför att?

- Leder till flexibel design
- Långsiktig underhållbarhet
- Alla coola kidsen gör det redan

Du skola endast en anledning till förändring hava

Single Responsibility Principle

Single Responsibility Principle

- Härledd från studier av Tom De Marco och Meilir Page-Jones
- ”Cohesion”
  - Funktionell
  - Sekventiell
  - Kommunicerande
  - Procedurell
  - Temporal (tidsmässig)
  - Logisk
  - Sammanträffande

Single Responsiblity Principle

- Hur orsakas förändring?
  - Nya krav
  - Förändringar i något beroende
- Sträva mot en enda konceptuell anledning att förändras
- ”An axis of change is an axis of change only if the change actually occur”

Vanligaste bovarna blanda flera av

- Lagring
- Presentation
- Affärslogik

Bonusprincip!

- **S**ingle
- **L**evel
- (of) **A**bstraction
- **P**rinciple

Du skola vara öppen för utvidgning men stängd för förändring

Open Closed Principle

Open Closed Principle

- En förändring kommer sällan ensam
- Tillägg > Förändring
- Gör nästa liknande förändring ett tillägg
- Arv är inte enda vägen
  - Eller bästa vägen
  - Eller ens en speciellt bra väg alls

Hur åstadkommer vi öppenhet?

- Delegering
- ”Composability”
- Data driven design
- Arv (när vi inte har något bättre val)

När skapar vi utvidningspunkter?

- När förändring händer!
- Testning stimulerar ofta till förändring

Härledda typer skola vara fullvärdiga alternativ till sina bastyper

Liskow Substitution Principle

- *” Let q(x) be a property provable about objects of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.”*

Liskow Substitution Principle

- Typisk brott när vi gör switchar på typ
- Handlar i grunden om invarianter (preconditions & postconditions)
- Policy inte protokoll
- Akta dig för interface sjuka

Användare skola icke behöva bero på metoder de icke bruka

Interface Segregation Principle

Interface Segregation Principle

- Snävare interface ger högre återanvändbarhet
- Flera interface per klass
- Interface per klienttyp
- Generisk programmering

Dynamisk och ”Duck” typing

- Implicita interface per användare!

Beroenden skola stärfva i riktning mot det abstrakta

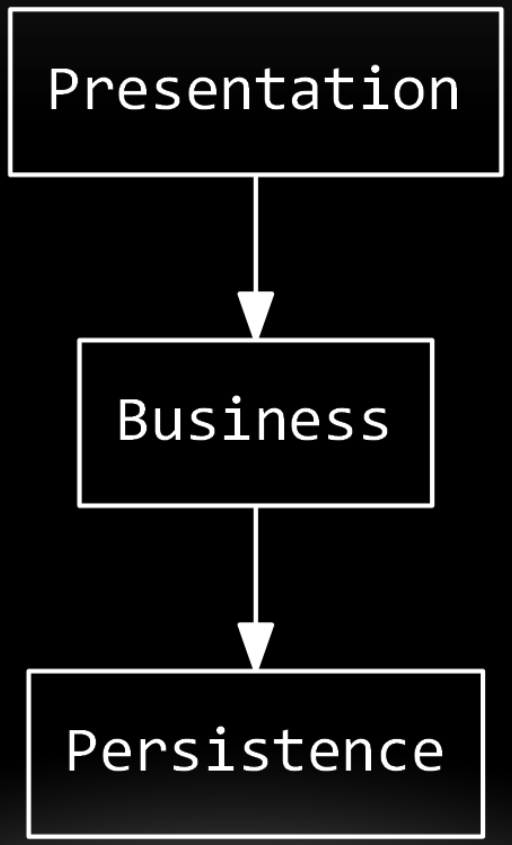
Dependency Inversion Principle

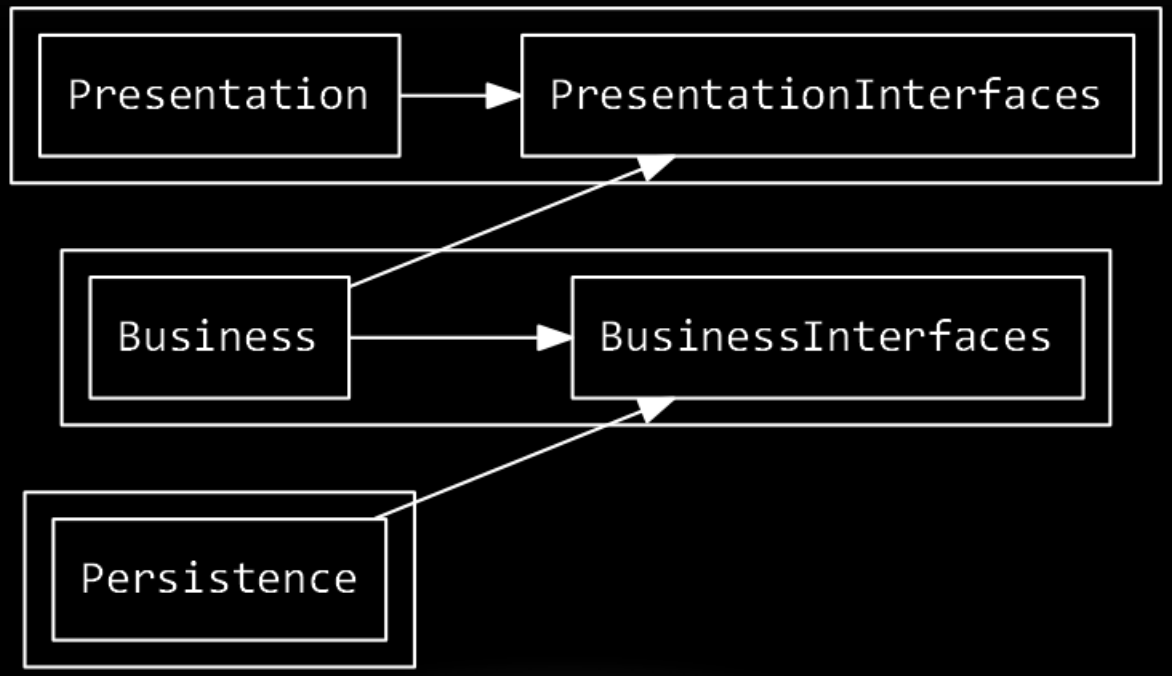
Dependency Inversion Principle

- Beroenden bör vara mot abstraktioner
- Beroenden flödar uppåt inte nedåt
- IoC containers
- ”Tell, don’t ask.”

Föredra abstrakta beroenden

- Föredra att ha referenser till interfaces eller abstrakta klasser
- Om du ärver, ärv från abstrakta typer
- Undvik att överlagra en redan existerande metod


Presentation
Business
Persistence


Presentation
PresentationInterfaces
Business
BusinessInterfaces
Persistence

För mer info

- http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod