Clean Code

Design Principles

www.everyloop.com

Vad är Clean Code?

- Kod som är lätt att läsa
- Kod som är lätt att utöka
- Kod som kan återanvändas
- Kod som inte går sönder på oväntade ställen

Vad är smutsig kod? (Code smell)

- Kod som inte är tydlig
- Kod som är onödigt komplex
- Kod som går sönder på oväntade sätt
- Kod som gör saker den inte ska

Enkla tumregler

- Namngivning är **VIKTIGT!**
  - Välj variabelnamn som beskriver dess innehåll.
  - Välj metodnamn som beskriver dess uppgift.
- Om du behöver kommentera koden du skrivit för att den ska vara läsbar. Skriv om koden så den går att förstå istället.
- Det är bättre med många små funktioner och metoder, än stora oläsliga.

Design Principles

Generella riktlinjer.

- **DRY**
  Don’t Repeat Yourself.
- **KISS**
  Keep It Simple, Stupid.
- **YAGNI**
  You Ain’t Gonna Need It.
- **SOLID**
  Samlingsnamn för 5 designprinciper:

  SRP, **O**CP, **L**SP, **I**SP, **D**IP

Design Patterns

Lösningar på specifika problem.

- **Factory Pattern**
  Skapa nya objekt.
- **Build Pattern**
  Konstruera komplexa objekt.
- **Strategy Pattern**
  Tillhandahålla olika algoritmer.
- **Dependency Injection**
  Skapa löst knutna beroenden.

SOLID

- **S**ingle **R**esponsibility **P**rinciple
- **O**pen/**C**losed **P**rinciple
- **L**iskov **S**ubstitution **P**rinciple
- **I**nterface **S**egregation **P**rinciple
- **D**ependency **I**nversion **P**rinciple

SOLID är en samling riktlinjer för att minimera konsekvenserna av framtida förändringar av din kod. I praktiken handlar det om att göra kvalificerade gissningar kring hur kravspecifikationen för din mjukvara troligast kommer förändras framöver.

Single Responsibility Principle

- “A class should have only one reason to change.”
- “A method should do only one thing, with no side effects.”
- “Gather together the things that change for the same reason. Separate those things that change for different reasons.”
- SRP handlar om bedöma hur kod bäst bryts ner i beståndsdelar, och hur man drar gränser kring inkapsling i sin kod.

Separation of Concerns

Programs bör brytas ner i väl avgränsade delar, där varje del adresserar en separat sak.

Tight Coupling

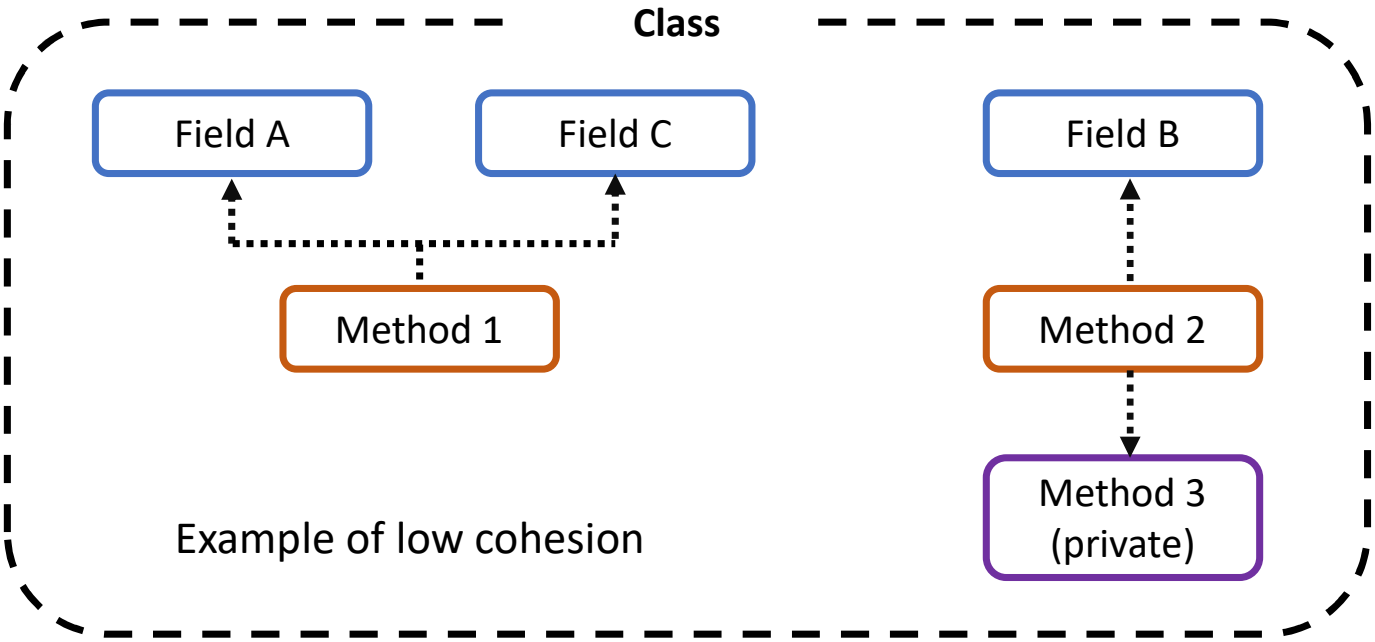
Binder ihop två, eller fler, detaljer på ett sätt så att de är svåra att separera.

Loose Coupling

Tillhandahåller ett modulärt sätt att välja vilka detaljer som är involverade I en given operation.

Class Cohesion

Class elements that belong together are cohesive.



Open/Closed Principle

- “Software entities (classes, modules, functions) should be open for extension, but closed for modification.” – Bertrand Meyer
- “You should be able to extend the behavior of a system without having to modify that system.” – Bob Martin
- OCP handlar om att bedöma vilka delar av ditt system som bör tillhandahålla konkret funktionalitet (som du själv implementerat), och vilka delar som är abstrakta konstruktioner där den som använder systemet själv kan lägga in konkret funktionalitet.

Avvägning mellan abstrakt och konkret

```
public class TotallyAbstract<TArg, TRes>
{
   private Func<TArg, TRes> f;

   public TotallyAbstract(Func<TArg, TRes> f)
   {
      this.f = f;
   }

   public TRes Apply(TArg a) {
      return f(a);
   }
}
```

```
public class TotallyConcrete
{

   public int Apply()
   {
      return 2 + 2;
   }

}
```

Liskov Substitution Principle

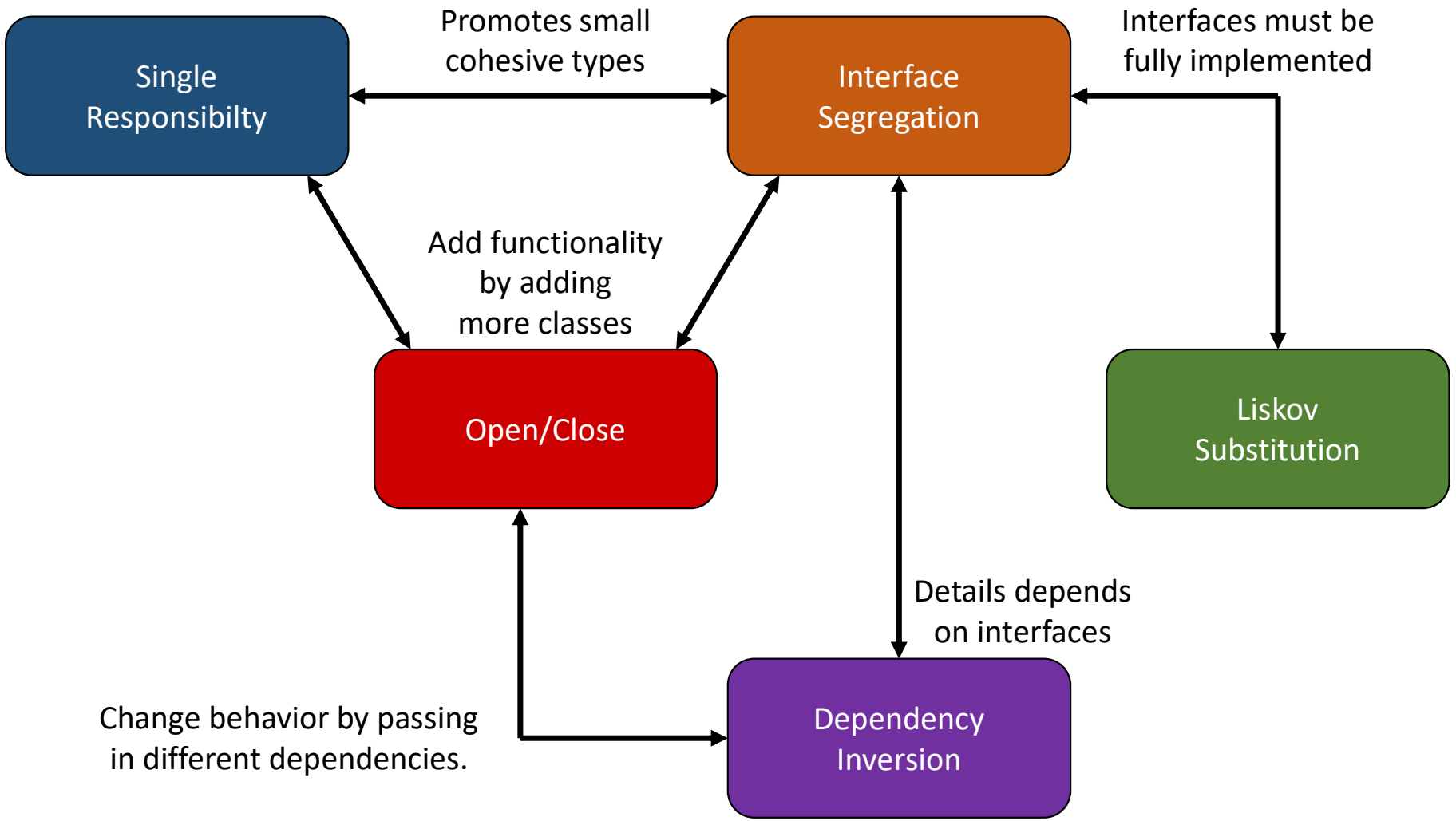
- “Methods that use references to base classes must be able to use objects of derived classes without knowing it.”
- “A basetype must be replaceable with any of its subtypes without breaking the functionality or logic of the program.”
- “Interfaces must be fully implemented.”
- Principen hjälper dig att undvika oväntade buggar och fel när du använder arv och polymorfism i din kod.

Interface Segregation Principle

- “Clients should not be forced to depend upon methods that they do not use.”
- “Prefer small, cohesive interfaces, to large, expansive ones.”
- Problemet med stora interface är att samtliga medlemmar måste implementeras för att inte bryta mot Liskov Substitution Principle; även dem som vi egentligen inte behöver.
- Principen hjälper dig att skapa mer modulär, oberoende kod, som går att återanvända för fler syften.

Dependency Inversion Principle

- “High-level modules should not depend on low-level modules. Both should depend on abstractions.”
- “Abstractions should not depend on details. Details should depend on abstractions.”
- Principen hjälper dig att skapa löst kopplade beroenden och modulär struktur, där högnivåkod inte är direkt beroende av detaljer på låg nivå.


Single Responsibilty
Promotes small cohesive types
Interface Segregation
Interfaces must be fully implemented
Add functionality by adding more classes
Open/Close
Liskov Substitution
Details depends on interfaces
Change behavior by passing in different dependencies.
Dependency Inversion