

# Things to consider for testable code

Frank Kleine, 27.10.2009



# The Speaker: Frank Kleine

- 1&1 Internet AG  
(Head of Web Infrastructure)
- PHP since 2000
- Lead Developer
  - Stubbles
  - XJConf for PHP
  - vfsStream
- Technical editor "PHP Design Patterns"



# Separation of Concerns

- Typical concerns in applications





Business logic



Ef@g#IT/htm?JqmbJ]kPtN/M!":-m@oFLL<7]mS74/113Cn>3-HZDW+;;\iqNHMPM]099Ef@g#IT/htm?JqmbJ]kPtN/M!":-m@oFLL<7]mS74/113Cn>3-HZDW+;;\iqNHMPM]099  
<#PU\Xrh1WDrRR=8\dfD,=Ds82fjs7XW\o)JOTrqZEir9CWcrqu'orV'jRq>^lCq>^F<j<#PU\Xrh1WDrRR=8\dfD,=Ds82fjs7XW\o)JOTrqZEir9CWcrqu'orV'jRq>^lCq>^F<j  
f-ig"O&L=o&plroAY\*nG]?UkOe);MnJ7fOnADJrWV>O\aoFcs8FD5oY"Q\*li\$+k :s,%pf-ig"O&L=o&plroAY\*nG]?UkOe);MnJ7fOnADJrWV>O\aoFcs8FD5oY"Q\*li\$+k :s,%p  
1KhfBr;ndJ'Yd(YYbe,-3&kkOQFrrW#mZN'Y)m\*i9=O)GTeli6ulp\OphXnVharg4d'p1KhfBr;ndJ'Yd(YYbe,-3&kkOQFrrW#mZN'Y)m\*i9=O)GTeli6ulp\OphXnVharg4d'p  
E\>gV;8;n\*K1GpAWCSO'l\Xo)J[tDzOnW\_tX011.4&Mp&G"7f\$X/io" 'a's8MumVuGQ\$W E\>gV;8;n\*K1GpAWCSO'l\Xo)J[tDzOnW\_tX011.4&Mp&G"7f\$X/io" 'a's8MumVuGQ\$W  
j;Wr.'<HkU1VSNRn-u\$N9gb\_sZU<mJ?bZrVm"r8En8rr;m-rqZ?grUQ34qtg9erq1EdEj;wr.'<HkU1VSNRn-u\$N9gb\_sZU<mJ?bZrVm"r8En8rr;m-rqZ?grUQ34qtg9erq1EdE  
<#urWrK1o @-fg=FL\_rpTjerr;p,q>^/oPI(8)o n'rr;<arX\ '#s81m@s72fws8Mfn5<#urWrK1o @-fg=FL\_rpTjerr;p,q>^/oPI(8)o n'rr;<arX\ '#s81m@s72fws8Mfn5  
2Y.g>\$'I=K(K'ot8Tip[kln') P%o(MF5G<o2)<],qf](Pmus8W)t'W#r;r;HZp"8Ving2Y.g>\$'I=K(K'ot8Tip[kln') P%o(MF5G<o2)<],qf](Pmus8W)t'W#r;r;HZp"8Ving  
"ORh8K?m-78UX]qEGORX\*qHfU">2Tp=6\$il6TDN&iK,ELp4do)J[toRuf6rr;Yor;F%Gq"ORh8K?m-78UX]qEGORX\*qHfU">2Tp=6\$il6TDN&iK,ELp4do)J[toRuf6rr;Yor;F%Gq  
Z]p#Q2Tppu6QoqZS01s7Pu(oDe^ep&"a]fY%"PgtU0hppf5]rYkP0rqEMARV1'nZH)k=cZ]p#Q2Tppu6QoqZS01s7Pu(oDe^ep&"a]fY%"PgtU0hppf5]rYkP0rqEMARV1'nZH)k=c  
Ire\*m;rMuWbVqu?U'jOC3+jnn<8laaA\$0Dee;mIpO\_UTKUQnFZ\_&R[j?;ZK&VTQTJlIre\*m;rMuWbVqu?U'jOC3+jnn<8laaA\$0Dee;mIpO\_UTKUQnFZ\_&R[j?;ZK&VTQTJl  
=#iKM,sDkklL5s8NZ)qr\1S)s:JoAoo7fq&0,rr32flcQU4nc&RD%CGTArf0oWGa(R1)-#iKM,sDkklL5s8NZ)qr\1S)s:JoAoo7fq&0,rr32flcQU4nc&RD%CGTArf0oWGa(R1]  
'#XqoZMMp\t.Hk4VM<D0b45pAacpc272\*b/"#PZ1@nhlMpMUqPwi(^%M-smC1d&mu)Vmd'XqoZMMp\t.Hk4VM<D0b45pAacpc272\*b/"#PZ1@nhlMpMUqPwi(^%M-smC1d&mu)Vmd  
2)@[SN'3E\b4UA^Rg+rVm\$!r:m2g8N&s#ke#mrU?'7rqcZn#6+>qplk@Ks9N&us8E#ss2)@[SN'3E\b4UA^Rg+rVm\$!r:m2g8N&s#ke#mrU?'7rqcZn#6+>qplk@Ks9N&us8E#ss  
<dm6rV16b\*+<,p5s68^%YPn(jrUZ]ls7H<j0\_\|VYa(s0)(Z1()Ys8&1&q>r:g3OM'mI) <dm6rV16b\*+<,p5s68^%YPn(jrUZ]ls7H<j0\_\|VYa(s0)(Z1()Ys8&1&q>r:g3OM'mI)  
>/4eOD/Fb\c3\*3qtTmPpYtGo^33aLs&A:p%N[fs8W#tr1+m;rr2j\$R:9gOr:'CY"- (Uhp/4eOD/Fb\c3\*3qtTmPpYtGo^33aLs&A:p%N[fs8W#tr1+m;rr2j\$R:9gOr:'CY"- (Uhp  
7(Ar[m^?gAAQPaLHJDM19NKoi[aWh]'e+AshC'potsp&+OL[+R2o]i@A1' /s8E&tr(Ar[m^?gAAQPaLHJDM19NKoi[aWh]'e+AshC'potsp&+OL[+R2o]i@A1' /s8E&tr  
23mqrquflrr)lJrIsnDp&=derq+IUqtg?hrqc5SmIU>UrgG'Ap%JC\kK=qmrVWq/p\XmU  
&@i3mdTiIoDPrNm.' "kc1g?!n+ZhLV8)u\$07E2Ao)\*Etps[1\_>j\$@>0d-1p:tInq8M4Qr  
'4E]o" 'L\_rXeI![%caOrU^SgmB60Wq>1-krW2Gbs6gX#c"d4QqjR9Wb!0\SF\T74q"iX\_r4E]o" 'L\_rXeI![%caOrU^SgmB60Wq>1-krW2Gbs6gX#c"d4QqjR9Wb!0\SF\T74q"iX  
fK+r;50-Q?8^qn,E:crW:3^oDednN8jX'XP3^R\$Mj\_rrV1VHR-sbBs8EbP;;InXK^a3/afK+r;50-Q?8^qn,E:crW:3^oDednN8jX'XP3^R\$Mj\_rrV1VHR-sbBs8EbP;;InXK^a3/af  
'6\*Ja<;WH2mdBrVca#rVZGerV1'przDusrWd1UN9(\$<rr!#mrq;TBrVulss8W'rvZQ\_f'6\*Ja<;WH2mdBrVca#rVZGerV1'przDusrWd1UN9(\$<rr!#mrq;TBrVulss8W'rvZQ\_f  
l6b#laf%q=ji,Y4)>f"lBjDs6ogart"o)qIWeV';0?2p@MT>q>1-k#OKLQN'1;Nq>^Esnl6b#laf%q=ji,Y4)>f"lBjDs6ogart"o)qIWeV';0?2p@MT>q>1-k#OKLQN'1;Nq>^Esn  
ZfsrOMq2nb'1KK)^<.neFlHpjC(+;YKcds8W)s'W,o9#Q=Ytf=(S?pc>j<c/AKtm+T5R^ZfsrOMq2nb'1KK)^<.neFlHpjC(+;YKcds8W)s'W,o9#Q=Ytf=(S?pc>j<c/AKtm+T5R  
-1%NkP53=);AmPt/oZ]S"0>D[:Bj^nfufqXX!Br:]mahRE(BM'gfErVc^#rF%.eo)A[fi-1%NkP53=);AmPt/oZ]S"0>D[:Bj^nfufqXX!Br:]mahRE(BM'gfErVc^#rF%.eo)A[fi  
0[frr;pMq>U0gd]l:cp&=X[ratm's7ZEgcF39@p%e-Jch@>0J?&5:p\rrRpoA4KiV]P>>0[frr;pMq>U0gd]l:cp&=X[ratm's7ZEgcF39@p%e-Jch@>0J?&5:p\rrRpoA4KiV]P>>  
0i6R,7H-ocW"Vp@4]prT'9mp%eUihlitags#Q(h<phOm(kZ%N)s<Z0:'Xhh6BtrKQ[Br0i6R,7H-ocW"Vp@4]prT'9mp%eUihlitags#Q(h<phOm(kZ%N)s<Z0:'Xhh6BtrKQ[Br  
50erUU!'s7cQ&lMpkbrq-6j(?Maso'h);s7Z2ep[6"<@Z@frd.mV(s8F)\*priJlRpb^(50erUU!'s7cQ&lMpkbrq-6j(?Maso'h);s7Z2ep[6"<@Z@frd.mV(s8F)\*priJlRpb^(  
&lls#R0\_uKG\NT:'3W;Q\p!;g'f'Cr?&eboF>q<jZ/P0UVf^7'lgkl:8hqWD[0mf(,Uj&lls#R0\_uKG\NT:'3W;Q\p!;g'f'Cr?&eboF>q<jZ/P0UVf^7'lgkl:8hqWD[0mf(,U  
cZnr;Bhrr2j-rq1]ifr4)<s6Mopr:U!)ir8rTrs/PlronItq>U3irVcrsrpTjVs8E,mscZnr;Bhrr2j-rq1]ifr4)<s6Mopr:U!)ir8rTrs/PlronItq>U3irVcrsrpTjVs8E,ms  
W/mq8cS!rr\*\*!m^Di]rVuWkrse;GAVLk\*s8DflGSp,Fs7cf-2<W\_NrUp0h"n0S5o'+jnrW/mq8cS!rr\*\*!m^Di]rVuWkrse;GAVLk\*s8DflGSp,Fs7cf-2<W\_NrUp0h"n0S5o'+jnr  
Vs"jRZPI:9FQI#k.,H;F+FXq>UBns8L.?s8;rsrr<#ts?7KY]"uMzrX??+K6+Euq<k6]Vs"jRZPI:9FQI#k.,H;F+FXq>UBns8L.?s8;rsrr<#ts?7KY]"uMzrX??+K6+Euq<k6]  
kKSW8k<P,OAE[IB<]| "RLO845S,F-^X?n,N@shKemer;#ZGp<B'#qu\$EkrYb[Gf\_tF@kKSW8k<P,OAE[IB<]| "RLO845S,F-^X?n,N@shKemer;#ZGp<B'#qu\$EkrYb[Gf\_tF@  
Xi'kPkGzrV1Qjc-#nYs8EK!rnEn?qu?KioXVm3p](3trM/[r;D\ls83)srQhSER:'ac%Xi'kPkGzrV1Qjc-#nYs8EK!rnEn?qu?KioXVm3p](3trM/[r;D\ls83)srQhSER:'ac%  
fRTMS];jp>>]Fs7@o)[Z/H,rV=f[p\$pmmg0fEoURf-XW;YA+F'- \$Q1M^>Fr1N\*o' \*6<fRTMS];jp>>]Fs7@o)[Z/H,rV=f[p\$pmmg0fEoURf-XW;YA+F'- \$Q1M^>Fr1N\*o' \*6<  
C<unBoP?Sa-4;s8E<\$nCjc)s7ZKjs8On'T'<Dcs71KYp6b[Pl'YPBPuPwq#90Ue!-]EcC<unBoP?Sa-4;s8E<\$nCjc)s7ZKjs8On'T'<Dcs71KYp6b[Pl'YPBPuPwq#90Ue!-]Ec  
3n?cDp?g#C?icH>S0g8WU\*rW\$lrWi2sp[E]nG3+'SpWE:YGet'S6DmBYXTH\$NL+k-3n?cDp?g#C?icH>S0g8WU\*rW\$lrWi2sp[E]nG3+'SpWE:YGet'S6DmBYXTH\$NL+k-  
[(0s8N,ogZ\$QsoVh'grW<,urr;g(r1<7)qtg<grr)Wh :S\cr;R\$Spa=B%q=ssbr'&h1[(0s8N,ogZ\$QsoVh'grW<,urr;g(r1<7)qtg<grr)Wh :S\cr;R\$Spa=B%q=ssbr'&h1  
sO #4VBaq#:!Ss8EH\$Z7'H^oDAQ]^WQNs8O>@p\$H4OJ>]Hn1M^bYHkGT?kgY4CnBkn2rsO #4VBaq#:!Ss8EH\$Z7'H^oDAQ]^WQNs8O>@p\$H4OJ>]Hn1M^bYHkGT?kgY4CnBkn2r  
"J2s8)Z&rt>>&qt\*1Z??>Y-1E"2d>%74\$rr2otrlG->rr2otrr2j#m [|Ail^mGepV]+T"J2s8)Z&rt>>&qt\*1Z??>Y-1E"2d>%74\$rr2otrlG->rr2otrr2j#m [|Ail^mGepV]+T  
eeFrp]:M[#\*]=o)7=rW3-'aC?)Up\#LhuA9\*Z2X8;Mgf6Hicr<'8k07pAmqYU3hrVd?>eeFrp]:M[#\*]=o)7=rW3-'aC?)Up\#LhuA9\*Z2X8;Mgf6Hicr<'8k07pAmqYU3hrVd?>

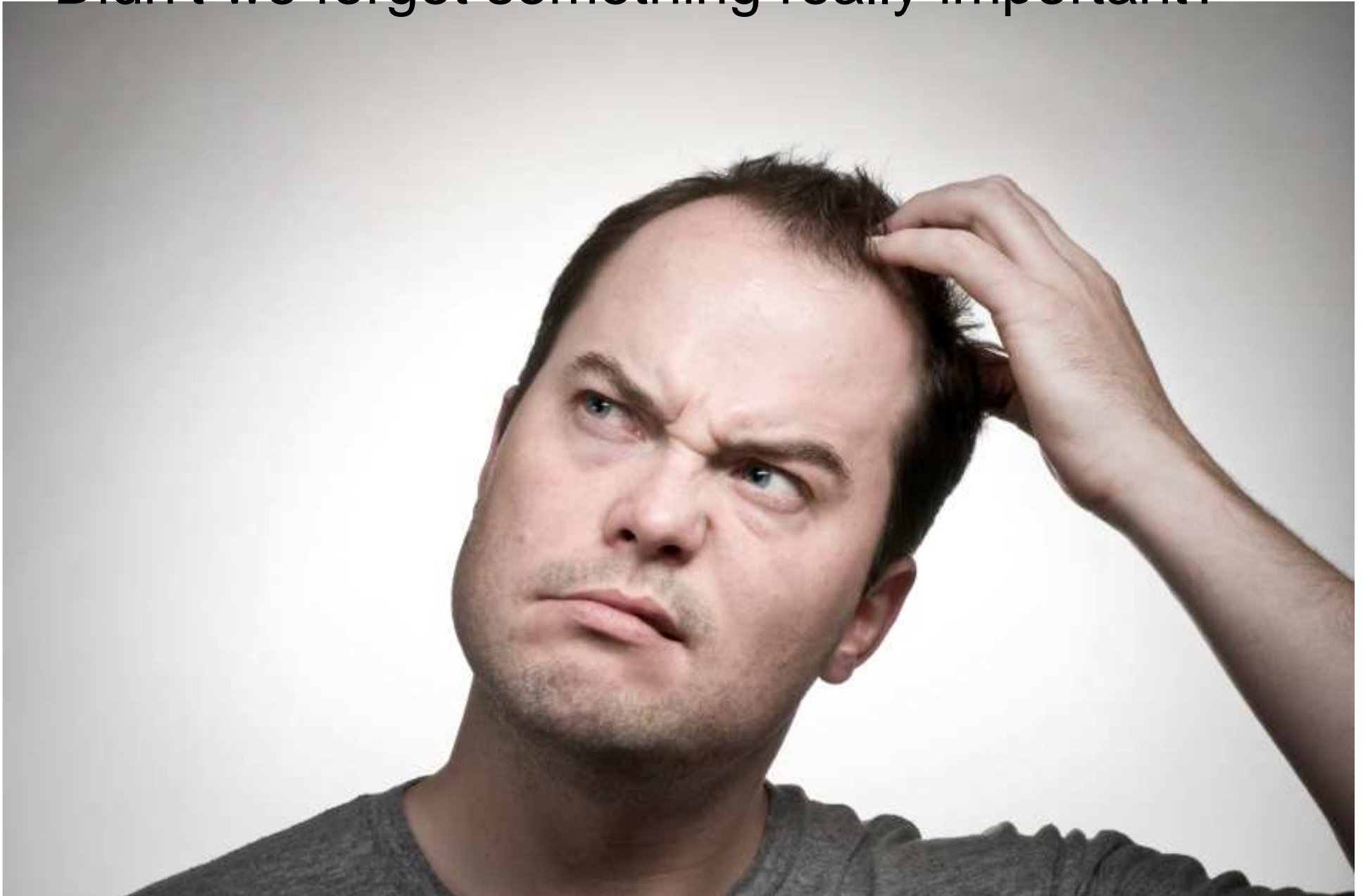


# Error handling





Didn't we forget something really important?





# Construction of Objects!



# Construction of objects

- new is a very powerful keyword



# Construction of objects

- `new` is a very powerful keyword
- Binds usage to a concrete implementation



# Construction of objects

- `new` is a very powerful keyword
- Binds usage to a concrete implementation
- Golden rule of `new`:
  - OK for domain classes, but not for services





# Construction of objects

- `new` is a very powerful keyword
- Binds usage to a concrete implementation
- Golden rule of `new`:
  - OK for domain classes, but not for services
  - OK in tests and specialised construction classes, but not in business logic



# Construction of objects II

- Bad:

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```



# Construction of objects II

- Bad:

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```

Work in  
constructor  
(Anti-Pattern)



# Construction of objects II

- Bad:

Hard to test

Work in  
constructor  
(Anti-Pattern)

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```





# Construction of objects II

- **Bad:**

Hard to test

Work in  
constructor  
(Anti-Pattern)

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```

- **Good:**

```
class Car {  
    public function __construct(Engine $eng, Tire $tire) {  
        $this->engine = $eng;  
        $this->tire    = $tire;  
    }  
}
```



# Construction of objects II

- **Bad:**

Hard to test

Work in  
constructor  
(Anti-Pattern)

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```

Piece of  
cake to test

- **Good:**

```
class Car {  
    public function __construct(Engine $eng, Tire $tire) {  
        $this->engine = $eng;  
        $this->tire    = $tire;  
    }  
}
```



# Construction of objects II

- **Bad:**

Hard to test

Work in  
constructor  
(Anti-Pattern)

```
class Car {  
    public function __construct() {  
        $this->engine = new Engine();  
        $this->tire    = TireFactory::createTire();  
    }  
}
```

Piece of  
cake to test

- **Good:**

```
class Car {  
    public function __construct(Engine $eng, Tire $tire) {  
        $this->engine = $eng;  
        $this->tire    = $tire;  
    }  
}
```

Dependency  
Injection



# Dependency Injection

- Pile of new (startup phase)
  - Create objects and stack them together
  - "Boilerplate", but automatable with DI framework
- Pile of objects (runtime phase)
  - Business logic, error handling, etc.
- Factories or DI instances for runtime object creation







**Dependency Injection is viral**

# Law of Demeter

- Bad:

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->engine->stop();  
    }  
}
```



# Law of Demeter

- Bad:

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->engine->stop();  
    }  
}
```



Driver  
coupled to  
Engine



# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->engine->stop();  
    }  
}
```

Hard to test: test  
always needs  
Engine or a mock  
of it

Driver  
coupled to  
Engine



# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->  
        $this->vehicle->drive(  
        $this->vehicle->engine-  
    }  
}
```

Hard to test: test  
always needs  
Engine or a mock  
of it

Driver  
coupled to  
Engine

Internal state  
of Vehicle  
revealed



# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->  
        $this->vehicle->drive(  
        $this->vehicle->engine-  
    }  
}
```

Hard to test: test  
always needs  
Engine or a mock  
of it

Driver  
coupled to  
Engine

Internal state  
of Vehicle  
revealed

- **Good:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->stop();  
    }  
}
```



# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->  
        $this->vehicle->drive(  
        $this->vehicle->engine-  
    }  
}
```

Hard to test: test  
always needs  
Engine or a mock  
of it

Driver  
coupled to  
Engine

Internal state  
of Vehicle  
revealed

- **Good:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->stop();  
    }  
}
```

Driver not coupled  
to Engine: simpler  
to maintain





# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->  
        $this->vehicle->drive(  
        $this->vehicle->engine-  
    }  
}
```

Hard to test: test  
always needs  
Engine or a mock  
of it

Driver  
coupled to  
Engine

Internal state  
of Vehicle  
revealed

- **Good:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->stop();  
    }  
}
```

Piece of  
cake to test

Driver not coupled  
to Engine: simpler  
to maintain



# Law of Demeter

- **Bad:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->engine->  
        $this->vehicle->drive(  
        $this->vehicle->engine-  
    }  
}
```

Hard to test: test always needs Engine or a mock of it

Driver coupled to Engine

Internal state of Vehicle revealed

- **Good:**

```
class Driver {  
    public function drive($miles) {  
        $this->vehicle->start();  
        $this->vehicle->drive($miles);  
        $this->vehicle->stop();  
    }  
}
```

Piece of cake to test

Driver not coupled to Engine: simpler to maintain

Less prone to errors



# Global state

- Same operation with same inputs should yield same results.
- Not necessarily true with global state.



# Global state

- Same operation with same inputs should yield same results.
- Not necessarily true with global state.
- Hidden global state
  - Singleton
  - static methods
  - `$_GET`, `$_POST`, `$_SESSION`, `$_...`
  - Registry



# Global state: Singletons

- Never implement any
- Required most likely due to bad design of your framework



# Global state: Singletons

- Never implement any
- Required most likely due to bad design of your framework
- Use a DI framework with support for Singleton scope
  - Gives full power of Singletons
  - Code using the Singleton stays simple



# Singleton with DI framework

```
$binder->bind('Session')  
    ->to('PhpSession')
```





# Singleton with network

Configure  
the binding

```
$binder->bind('Session')  
->to('PhpSession')
```



# Singleton with DI

Configure  
the binding

Enforce  
singletoness: DI  
framework will only  
create one instance  
of PhpSession and  
inject always this  
same instance

```
$binder->bind('Session')  
->to('PhpSession')  
->in(stubBindingScopes::$SINGLETON);
```



# Singleton with DI

Configure the binding

Enforce singletonness: DI framework will only create one instance of PhpSession and inject always this same instance

```
$binder->bind('Session')  
    ->to('PhpSession')  
    ->in(stubBindingScopes::$SINGLETON);
```

```
class Processor {  
    protected $session;  
    /**  
     * @Inject  
     */  
    public function __construct(Session $session) {  
        $this->session = $session;  
    }  
    ...  
}
```



# Singleton with DI

Configure the binding

Enforce singletoness: DI framework will only create one instance of PhpSession and inject always this same instance

```
$binder->bind('Session')  
    ->to('PhpSession')  
    ->in(stubBindingScopes::$SINGLETON);
```

```
class Processor {  
    protected $session;  
    /**  
     * @Inject  
     */  
    public function __construct(Session $session) {  
        $this->session = $session;  
    }  
    ...  
}
```

Dependency Injection



# Singleton with DI

Configure the binding

Enforce singletonness: DI framework will only create one instance of PhpSession and inject always this same instance

```
$binder->bind('Session')  
->to('PhpSession')  
->in(stubBindingScopes::$SINGLETON);
```

Tell DI framework to inject required parameters on creation of Processor

Dependency Injection

```
class Processor {  
    protected $session;  
    /**  
     * @Inject  
     */  
    public function __construct(Session $session) {  
        $this->session = $session;  
    }  
    ...  
}
```



# Singleton with DI

Configure the binding

Enforce singletoness: DI framework will only create one instance of PhpSession and inject always this same instance

```
$binder->bind('Session')  
->to('PhpSession')  
->in(stubBindingScopes::$SINGLETON);
```

Tell DI framework to inject required parameters on creation of Processor

Dependency Injection

```
class Processor {  
    protected $session;  
    /**  
     * @Inject  
     */  
    public function __construct(Session $session) {  
        $this->session = $session;  
    }  
    ...  
}
```

Piece of cake to test:  
independent of  
PhpSession



# Global state: `static` methods

- Not always bad
  - Simple operations without dependencies
- Always bad if state is involved
  - Might return different results with same input.





# Global state: `$_GET`, `$_POST`, ...

- Ugly to test



# Global state: `$_GET`, `$_POST`, ...

- Ugly to test
- Rule: never access those vars directly in business logic



# Global state: `$_GET`, `$_POST`, ...

- Ugly to test
- Rule: never access those vars directly in business logic
- Abstract access to globals with classes
  - Remember: Singleton is evil



# Global state: `$_GET`, `$_POST`, ...

- Ugly to test
- Rule: never access those vars directly in business logic
- Abstract access to globals with classes
  - Remember: Singleton is evil
- Even better: interfaces



# Global state: \$\_GET, \$\_POST, ...

- Ugly to test
- Rule: never access those vars directly in business logic
- Abstract access to globals with classes
  - Remember: Singleton is evil
- Even better: interfaces
- Security: use a Request class which enforces filtering/validating input



# Global state: Registry

- Using a DI framework: no registry required



# Global state: Registry

- Using a DI framework: no registry required
- Without DI framework: use Registry for config values only





# Global state: Registry

- Using a DI framework: no registry required
- Without DI framework: use Registry for config values only
- Did I already mentioned that Singletons are evil?



# Global state: Registry

- Using a DI framework: no registry required
- Without DI framework: use Registry for config values only
- Did I already mentioned that Singletons are evil?
  - static methods are enough





A close-up photograph of a large pile of golden-brown, triangular samosas. The samosas are scattered across a light-colored surface, with some showing signs of being broken or crushed. In the center of the pile, a small, rectangular, light-colored sign is placed at an angle. The sign has the text "Change is coming." written on it in a dark, sans-serif font. The word "Modify" is overlaid in a large, blue, sans-serif font in the upper-middle part of the image.

**Modify**

Change is coming.

A close-up photograph of a large pile of golden-brown, triangular samosas. The samosas are scattered across a light-colored surface, with some showing signs of being broken or crushed. A white banner is placed diagonally across the middle of the pile, featuring the text "Change is coming." in a black, sans-serif font. The lighting is bright, highlighting the texture of the fried dough.

**Simplify**

**Modify**

Change is coming.

A close-up photograph of a large pile of golden-brown, triangular samosas. The samosas are scattered across a light-colored surface, with some showing signs of being broken or crushed. In the center of the pile, a small, light-colored rectangular sign is placed, tilted slightly. The sign has the text "Change is coming." written on it in a dark, sans-serif font. The background is a soft, out-of-focus light color, emphasizing the texture and color of the samosas.

**Simplify  
Improve  
Modify**

Change is coming.

# Finally...

- If you remember only one little thing of this presentation it should be...





**Singletons are really, really  
(and I mean really)**

**DANGER**

EXTREMELY FLAMMABLE  
VAPORIZATION OF SPRAY MAY BE HARMFUL  
CONTAINER MAY EXPLODE IF HEATED

**CAUTION**





**Singletons are really, really  
(and I mean really)**

**DANGER**

EXTREMELY FLAMMABLE  
VAPORIZATION OF SPRAY MAY BE HARMFUL  
CONTAINER MAY EXPLODE IF HEATED

# The End

- Questions and comments?



# The End

- Questions and comments?
  
- Thank you.



# Commercial break

- Stephan Schmidt: PHP Design Patterns



# Commercial break

- Stephan Schmidt: PHP Design Patterns



- Clean code talks: [http://www.youtube.com/view\\_play\\_list?p=79645C72EA595A91](http://www.youtube.com/view_play_list?p=79645C72EA595A91)



# Commercial break

- Stephan Schmidt: PHP Design Patterns



- Clean code talks: [http://www.youtube.com/view\\_play\\_list?p=79645C72EA595A91](http://www.youtube.com/view_play_list?p=79645C72EA595A91)
- Stubbles: [www.stubbles.net](http://www.stubbles.net)

