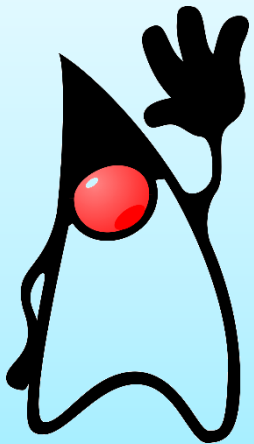


TDDD78, TDDE30, 729A85

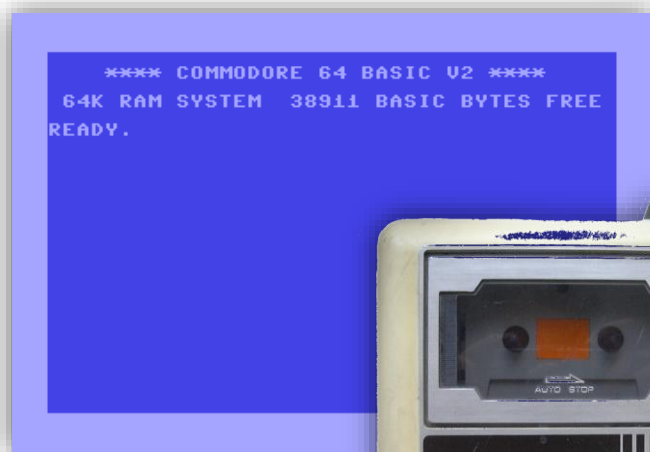
Objektorienterad programmering och Java



Kursinformation



Examinator, kursledare: Jonas Kvarnström



Fråga – kommentera – avbryt!





Objektorientering på två minuter

Utan OO har vi datastrukturer för lagring, som "manipuleras utifrån" av funktioner

```
daysSinceToday(date)  
isLeapYear(date)  
add(date, days)
```

Date-struktur

```
year: 2021  
month: 12  
day: 31
```

Med OO har objekt både datalagring och egen funktionalitet, egna metoder

- ➔ Fundamental princip:
"Objektet bestämmer över sig själv"
(ingen manipulerar utifrån)
- ➔ Varför? Grunden för begrepp som
ärvning, overriding, polymorfism, inkapsling,
...

Date-objekt

```
year: 2021  
month: 12  
day: 31
```

```
daysSinceToday()  
isLeapYear()  
add(days)
```

En klass är en datatyp, t.ex. Date, som anger:
Vilken *information* lagras i objekt av typ Date?
Vad kan objekt av den typen göra?

Mål och medel

**Objektorienterad
programmering**

**Konkret
OO-språk:
Java**



**Generella
begrepp:**

**Stark typning,
pekare,
kontrakt,
...**

ERT MÅL (?)

*Kunskap och färdigheter
inom
programutveckling*

**Generella
färdigheter:**

**Verktyg för
utveckling**

Programmeringsvana!

Vad kommer först?

Ibland föreläsning först
➔ läs + experimentera

Ibland labba först
➔ sätt i ett sammanhang
på föreläsningar

Ofta ”fram och tillbaka”!

Lyssna

Föreläsningar: Hur OO fungerar,
intressanta begrepp

*Ramar för förståelsen,
övergripande tankar och idéer*



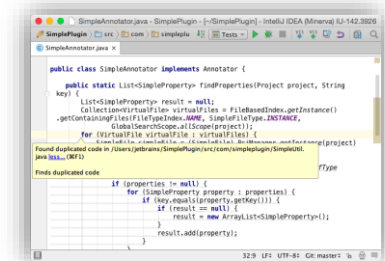
Läsa

Kursbok, instruktioner,
material på nätet, ...

*Mer detaljer – viktiga,
och läses bäst i egen takt*

MEDEL:

Hur ska ni uppnå era mål?



Utföra

Labbar och projekt

*Erfarenhet,
förståelse för vad man kan –
och vad man behöver öva på*

Föreläsningar

Introduktion

Kursintro, Java för Python-programmerare, typning, ...
Utan objektorientering: Ändå mycket nytt...

Objektorienteringens grunder

Principer och begrepp: Klasser, metoder, fält, konstruktorer, ...
Konkret användning av OO i Java

Viktiga begrepp, med och utan OO

Kontrakt, åtkomst och inkapsling, lagring och livstid,
pekare och komposition, identitet och likhet, ...

Hierarkier och arv

Typhierarkier med gränssnitt (*interface*), polymorfism,
statisk/dynamisk bindning, abstrakta klasser, *overriding*, ...

Grafiska gränssnitt

med *komponenter* och
händelsehantering

...

Projektinfo

Att välja
projekt, ...

- Föreläsningar ska alltid utgå från **förkunskapskraven**

Vad kan ni redan?

Grundläggande begrepp inom programmering

*Konkreta programmeringskunskaper i Python, motsvarande
t.ex. kurserna Funktionell och imperativ programmering del 1 och del 2.*



- Har du **mycket erfarenhet** av Java?
 - Skumma gärna genom bilderna på förhand – avgör vad som är **mest intressant**
 - Men missa inte det viktiga!

Förutsättningar för föreläsningar (2)

Vad bygger (närmast) på Java/OO?

OO

Begrepp

Java

Mer programmeringsvana!

U:

TDDD80 (11 hp)

Mobila och sociala applikationer / Java

D, U:

TDDD86 (11 hp)

Datastrukturer, algoritmer och programmeringsparadigm / C++

- Två alternativ:

Få föreläsningsbilder, prata fritt från kortfattade punkter

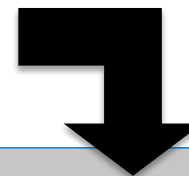
- (+) Känns ”spontant”**
- (+) Ingen går i förväg**
- (-) Lätt att missa vad som sägs**
- (-) Lätt att glömma, måste anteckna**
- (-) Föreläsaren kan missa detaljer**

Många föreläsningsbilder, skriv ner alla relevanta fakta

- (+) Bra att både höra och läsa**
- (+) Bra referensmaterial senare**
- (-) Kan kännas som att läsa innantill, repetitivt**



- Intressant för ett fåtal, men inte centralt?
- Rena fakta – tillgängliga metoder, ...?



The screenshot shows a web browser window with the address bar displaying <https://www.ida.liu.se/~TDDD78/labs/2017>. The page title is "Vanliga lösningar". The main heading of the page is "Ganska vanliga lösningar på ganska vanliga uppgifter". The content discusses a course with a central core of knowledge and skills, and a large "periferi" (periphery) of tasks that recur over time. It mentions that each group only benefits from a small part of this under the course, and that it is not clear which part they will need. It also states that the course collects solutions to these "ganska vanliga" (fairly common) tasks, instead of having all students participate in extra lectures where nothing is central to the course goals and only a small part is necessary for the project. A note in parentheses says: "(Sedan kan ju en del av det ändå vara *intressant* för många, men skulle vi ta upp allt som är intressant skulle kursen ta flera år...)" (Since some of it could still be *interesting* for many, but if we took up everything that is interesting, the course would take several years...). It concludes by saying that it also includes some solutions that *many* will benefit from, but which are more like pure **fakta** (facts) than concepts, so it is better to look them up when they are needed.

Jag vill...

- Göra något med strängar
- Göra något med listor
- Skapa en mappning, som *dict*
- Skapa menyer
- Fråga användaren något
- Reagera på tangenttryckningar
- Stänga ett fönster
- Rita ut en bitmap-bild
- Måla med texturer
- Måla genomskinligt
- Måla med kantutjämning
- Hitta typsnitt
- Upptäcka kollisioner på skärmen
- Spela upp ett ljud
- Spara eller skicka hela objekt

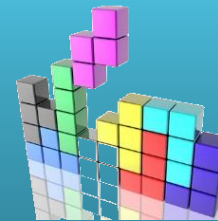
Labbar och projekt

Steg 1: Era första(?) Java-program

Fokus: Lär in rätt sätt, inte fel (svårt att bli av med)

Steg 2: Miniprojekt

Fortfarande ganska mycket vägledning,
för att strukturera ett större program



Steg 3: Eget projekt!

Helt fria händer
Använd er kunskapstörst,
lär er så mycket ni kan!

Instruktioner och
detaljstyrning minskar



"Varsågod, lär dig simma!"



*...eller ska man öva
"handgreppen" först?*

Fokus: Lär in rätt sätt, inte fel (svårt att bli av med)

Medel: Styr till bra lösningar
Tutorial-form: Steg-för-steg-instruktioner

Bara ett förberedelsesteg!

Mindre erfarenhet → effektiv metod för "rätt programmeringsvanor"

Mer erfarenhet → snabbt avklarat + kan få nya insikter / idéer...

Resultatet är inte kod, utan kunskap!

Tänk, reflektera, förstå varför vi programmerar på ett visst sätt

GOOD CODERS...



... KNOW WHAT THEY'RE DOING

**Utvecklingsmiljön
(IntelliJ IDEA)!**

Feedback!

Varningar för möjliga problem (600+ "inspektioner")

```
try {  
    new FileInputStream(f).read(data, 0, len);  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Result of 'FileInputStream.read()' is ignored [more...](#) (Ctrl+F1)

```
IncMor incMor = new IncMor(expectedSize);  
Map<Node, SNode> iNodeToSNode = new HashMap<>(expectedSize);  
iNodeToSNode.put(iNode, sNode);
```

Contents of collection 'iNodeToSNode' are updated, but never queried [more...](#) (Ctrl+F1)

```
// This means the we first see if the wait should be converted by the unco  
final int x = -conditioning.outNegCon.value; // this is the x in Morri  
if (x < 0) {
```

Dereference of 'conditioning.outNegCon' may produce 'java.lang.NullPointerException' [more...](#) (Ctrl+F1)

Har kraftigt minskat antal *rutinproblem* som ger komplettering

Fixa buggar, undvik kompletteringar, lär er direkt från varningarna!

Om du inte (är säker på att du) förstår varningen

1. Förklaringar från IDEA
2. Förklaringar på kurswebsidorna

```
new FileInputStream(f).read(data, 0, len);  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Result of 'FileInputStream.read()' is ignored [more...](#) (Ctrl+F1)

Vanliga varningar från IDEA | TDD... <https://www.ida.liu.se/~TDD78/la>

- **Chain of 'instanceof' checks** – Koden testar explicit vilken typ ett objekt har, t.ex. om det är en Circle, Rectangle eller Square, och gör sedan olika saker beroende på typen. Detta är oftast ett misstag eftersom man istället vill hantera detta via t.ex. subtypspolymorfism eller i vissa fall ad-hoc-polymorfism.
- **Class explicitly extends a Collection class** – När man skapar en subclass till en Collection-klass (t.ex. `class Foo extends List`) öppnar man samtidigt upp för att andra kan manipulera den nya klassens objekt genom alla metoder som man ärver ner från Collection-klassen. Detta är ett typiskt fall när delegering brukar vara ett bättre val än ärvning – då bestämmer man själv vilken av Collection-klassens funktionalitet som ska exponeras utåt.
- **Class without package statement / Class '...' lacks a package statement** – Som vi har sagt tidigare ska all kod ligga i paket.
- **Collection declared by class, not interface** – Man bör normalt deklarera collection-variabler eller fält med en gränssnittstyp, som `List`, istället för en konkret klass, som `ArrayList`. Detta hjälper till att undvika onödiga beroenden på exakt vilken konkret implementation av gränssnittet som används och gör det därmed lättare att byta implementation senare (till exempel att byta till `LinkedList` istället för `ArrayList` utan att byta ut deklarationstypen).

3. Fråga handledaren eller examinatorn

Handledare på labbar – Fråga!

Be om feedback även *när allt verkar fungera*:
Största problemen är de man inte vet om...
Låt inte handledarna vila!

Utvecklingsmiljön:

Analyserar koden, visar
problem / möjliga förbättringar,
ger *inlärningsmöjligheter*

Inlämningar:

Kompletteringar
ger också en chans
att lära sig

Feedback!

Kursdeltagare:

Fritt fram att
diskutera, så länge
som *lösningen*
är er egen

Men ni ska bli självständiga ingenjörer!
Tveka inte att fråga,
men handledarna ska ge ledtrådar, inte färdiga svar.

- Under flera tidigare år:
 - Ofta schemalagt på kvällar – för att salarna ska räcka till
 - **Många** outnyttjade platser i labbsalar – ”hemarbete”, ...

- I år:
 - Boka *något färre* platser (t.ex. 30 för 36 personer)
 - Bättre schema trots fler studenter
 - *Kan* bli överfullt någon enstaka gång
 - Säg till handledaren – kanske finns någon som inte hör hemma i salen
 - Oftast finns alternativa platser

- Extra alternativ:
 - Mjukvara också installerad i PC-PUL

Tidplan och schema

2018-01-15
2018-01-16
2018-01-17
2018-01-18
2018-01-19
2018-01-20
2018-01-21
2018-01-22
2018-01-23
2018-01-24
2018-01-25
2018-01-26
2018-01-27
2018-01-28
2018-01-29
2018-01-30
2018-01-31
2018-02-01
2018-02-02
2018-02-03
2018-02-04
2018-02-05
2018-02-06
2018-02-07
2018-02-08
2018-02-09
2018-02-10
2018-02-11
2018-02-12
2018-02-13
2018-02-14
2018-02-15
2018-02-16
2018-02-17
2018-02-18
2018-02-19
2018-02-20
2018-02-21
2018-02-22
2018-02-23
2018-02-24
2018-02-25
2018-02-26
2018-02-27
2018-02-28
2018-03-01
2018-03-02
2018-03-03
2018-03-04
2018-03-05
2018-03-06
2018-03-07

4 labbar, enskilt

**Miniprojekt,
enskilt**



**Skriv egen projektbeskrivning
(grupper om 2 rekommenderas)**

**3 hp
Godkänt /
komplettera**

**D1, kogvet:
period VT I**

2018-03-19
2018-03-20
2018-03-21
2018-03-22
2018-03-23
2018-03-24
2018-03-25
2018-03-26
2018-03-27
2018-03-28
2018-03-29
2018-03-30
2018-03-31
2018-04-01
2018-04-02
2018-04-03
2018-04-04
2018-04-05
2018-04-06
2018-04-07
2018-04-08
2018-04-09
2018-04-10
2018-04-11
2018-04-12
2018-04-13
2018-04-14
2018-04-15
2018-04-16
2018-04-17
2018-04-18
2018-04-19
2018-04-20
2018-04-21
2018-04-22
2018-04-23
2018-04-24
2018-04-25
2018-04-26
2018-04-27
2018-04-28
2018-04-29
2018-04-30
2018-05-01
2018-05-02
2018-05-03
2018-05-04
2018-05-05
2018-05-06
2018-05-07
2018-05-08
2018-05-09

**Projektarbete,
rapportskrivande**

**3 hp
Godkänt /
3 / 4 / 5**

**D1, kogvet:
period VT2**

2018-05-10
2018-05-11
2018-05-12
2018-05-13
2018-05-14
2018-05-15
2018-05-16
2018-05-17
2018-05-18
2018-05-19
2018-05-20
2018-05-21
2018-05-22
2018-05-23

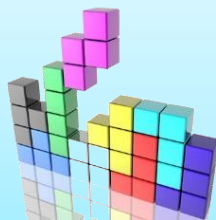
**D1: Rapport
Inlämning,
feedback
språkgranskning
med mera**

**1 hp
Godkänt / komplettera**

2018-01-15
2018-01-16
2018-01-17
2018-01-18
2018-01-19
2018-01-20
2018-01-21
2018-01-22
2018-01-23
2018-01-24
2018-01-25
2018-01-26
2018-01-27
2018-01-28
2018-01-29
2018-01-30
2018-01-31
2018-02-01
2018-02-02
2018-02-03
2018-02-04
2018-02-05
2018-02-06
2018-02-07
2018-02-08
2018-02-09
2018-02-10
2018-02-11
2018-02-12
2018-02-13
2018-02-14
2018-02-15
2018-02-16
2018-02-17
2018-02-18
2018-02-19
2018-02-20
2018-02-21
2018-02-22
2018-02-23
2018-02-24
2018-02-25
2018-02-26
2018-02-27
2018-02-28
2018-03-01
2018-03-02
2018-03-03
2018-03-04
2018-03-05
2018-03-06
2018-03-07

4 labbar, enskilt

**Miniprojekt,
enskilt**



**Friare projekt,
men generella förslag finns**

Grupper om 2 rekommenderas

**3 hp
Godkänt /
komplettera**

**3 hp
3 / 4 / 5 /
komplettera**

**U1: Kurs över
1 period
(men mindre att
göra parallellt)**

**Ingen
språkgranskning i
denna kurs**

	Måndag	Tisdag	Onsdag	Torsdag	Fredag
8-10	Block 2	Block 1	Block 4	Block 3	Block 2
10-12	Block 3	Block 2	Block 1		Block 1
13-15	Block 4	Block 3	Block 2	Block 1	Block 4
15-17	Block 4	Block 3	Block 2	Block 1	Block 3
17-19	1/2	1/2	1/2	1/2	
19-21	grupp	grupp	grupp	grupp	



Begränsad möjlighet för oss att välja tider

**Programmering:
Vad är vårt fokus?**

När är ett program (en inlämning) bra?

När körningen "ger rätt resultat"?

Nej!

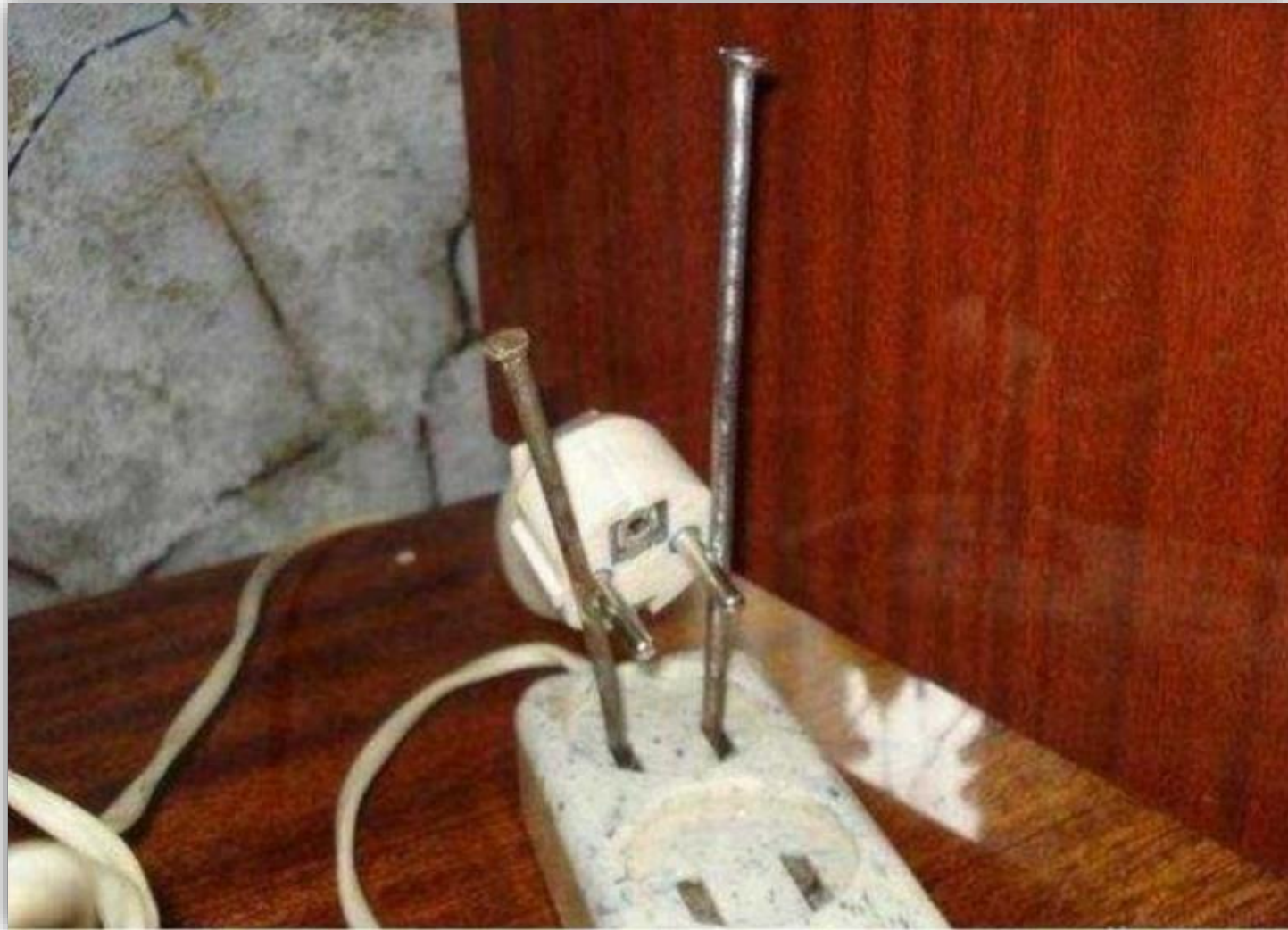
När programkoden är tydlig, välstrukturerad och
visar att ni vet vad ni gör!

Kvalitet (1)

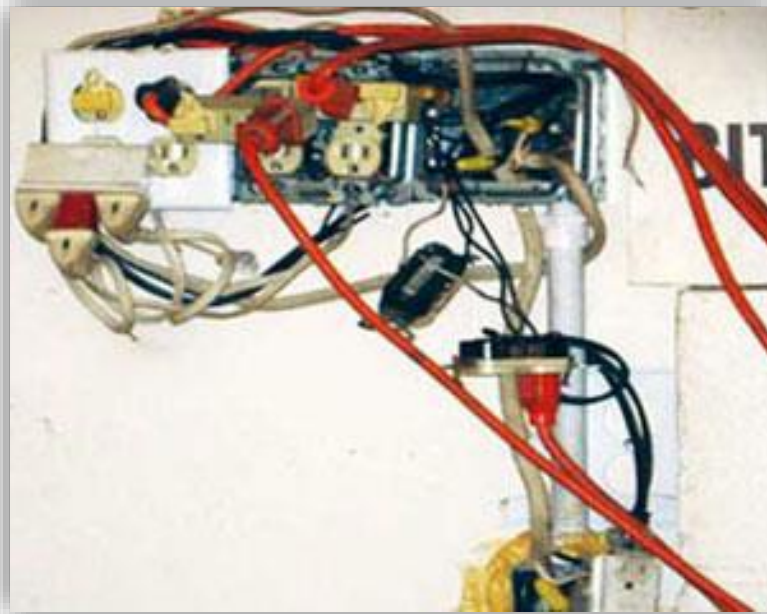
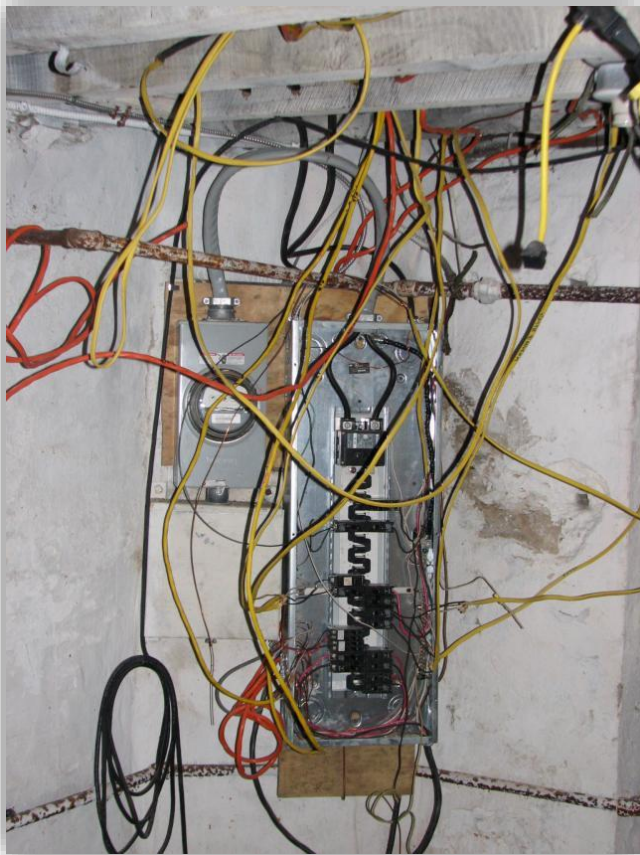
- Ser det ut så här,
blir vi misstänksamma...



- Har ni kopplat så här, är det farligt...



- Ser det ut så här, undrar vi vad ni egentligen har gjort
 - **Även** om ni faktiskt fick lampan att lysa

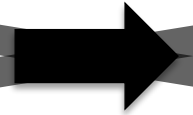


■ Ett extremt exempel:

```
■ class Div{static $1 $_;class $1{void _$(String $_){System.//  
out.print($_);}$1(){_();}void _(){int _,$_,$$,__,ä=(1<<5),  
å=100,ö=12,åö=ä*ö;å-=1<<4;while(åö>0){for($$=_$_=_=__=$=(int)  
å;_$_>($$-(1<<2));_$((""+(char)(_$_)),_$_-=1<<1)for(_=$=__-9;_>(  
$_-6);_-=1<<1,_$((""+(char)(_$_!=å?_:ä)));char S$=(char)(å+ö+1)  
;_$(("te"+S$+(char)(ö+(int)S$));åö--;}}}Div(){$_=new $1();}  
public static void main(String []$){Div å=new Div();}}
```

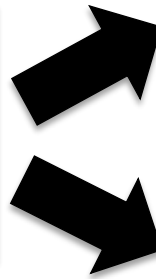
Skriver ut "TIGERteam" 384 gånger

Programmet fungerar
(ger rätt utmatning, ...)



Jag kan programmera!

Jag kan programmera!



Programmet fungerar
(ger rätt utmatning, ...)

Programmet är välskrivet,
lätt att förstå,
välstrukturerat, ...

Jag förstår varför
jag gör på ett visst sätt

- Skriv **bra** kod!
 - Lättläst
 - Bra namngivning
 - Strukturerad, modulär
 - Vældokumenterad och kommenterad – där det behövs mest!
- Extremt viktigt för:
 - Hur programmet kan utökas i framtiden
 - Hur andra kan förstå programmet
 - Hur robust det är när fel uppstår
 - ...

Utveckling är ofta 20% av arbetet...

Underhåll är 80%!

Hur ska man skriva kod?

- **Tips:**

- **Always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live.**



- *"I usually maintain my own code, so the as-if is true."*
– <http://c2.com/cgi/wiki?CodeForTheMaintainer>

Några sista ord om kursen

■ Nytt detta år:

■ **Mer omarbetning av föreläsningar**

- **Många** avsnitt uppdaterade, förbättrade, omorganiserade
- Några delar borttagna – tydligare *kursfokus*, mer tid för det viktigaste

■ **Alternativ implementation av Tetris**

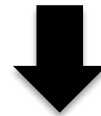
- Följ detaljinstruktionerna, eller...
- Följ de *målinriktade* instruktionerna (kommer snart)

■ **Justeringar i projektkrav**

- Tydligare bedömningsfokus (uppdateras snart)
- Siktat på färre kompletteringar



Början och slutet...

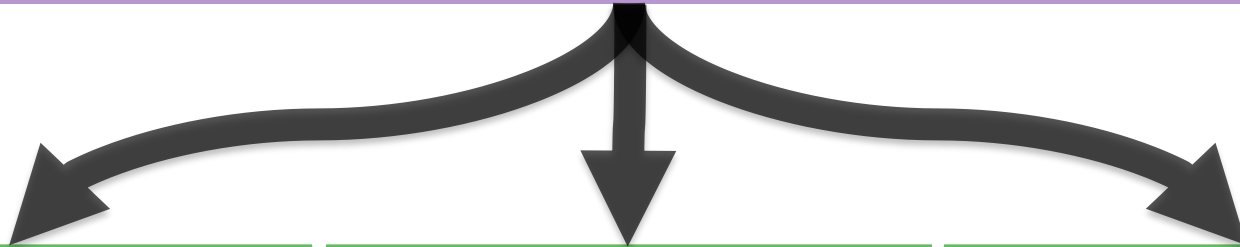


Nästa steg...

Java för Python-programmerare

Varifrån kommer Java? Varför använda det?

Jämförelser: Att starta ett program, syntax, uttryck, operatorer, villkorssatser, loopar (iteration), ...



Föreläsning

**Önskemål från
tidigare studenter!**

Läs / skumma själv

**Om det passar dig
bättre!**

Hoppa över...

**Om du vet
att du kan allt!**