

En introduksjon til objektorientert programmering

- Hvordan oppsto oop?
- Hva er oo?
- To eksempler [Horstmann 8.1-8.3]
- Klassen String [Horstmann 2.5]
- Java-biblioteket [Horstmann appendix D]
- En oppsummering

Hvordan oppnår man slik heder?

Ifi-professorene *Ole-Johan Dahl* og *Kristen Nygaard*:

2000: kommandør av St Olav

2001: ACM Turing award
(«Informatikkens
Nobel-pris»)

2002: IEEE John von
Neumann medal



Hva er oo?

Bakgrunnen i 1967 var at man i snart 20 år hadde programmert med variabler, arrayer og metoder. Det lar seg gjøre, men det er lett å miste oversikten når man skal skrive store programmer.

Løsningen

Hva om vi i stedet kan programmere akkurat det vi skal benytte i programmet?

Sagt med andre ord:

Vi kan programmere hvordan et sykkelspeedometer fungerer med variabler og metoder, men hadde det ikke vært bedre å programmere *selve sykkelspeedometeret*?

Eksempel 1: Et sykkelspeedometer

Et sykkelspeedometer er tilkopleet en føler på hjulet og kan

- vise farten
- vise kjørt distanse
- nullstilles
- sette omkretsen av hjulet



Klasser

Vi programmerer ting (både gjenstander og ideer) ved å definere **klasser**. Klassen blir da en **modell** av tingen.

```
class Sykkelspeedometer {  
    :  
}
```

Grensesnittet er det viktigste i en klasse

Grensesnittet

Vi må aller først finne ut hvordan vi skal bruke klassen, med andre ord hvilke operasjoner vi ønsker. Summen av operasjonene kalles klassens **grensesnitt** («interface»).

Grensesnitt for Sykkelspeedometer

- lese farten
- lese distansen
- nullstille distansen
- lese omkretsen av hjulet
- øke omkretsen 1 cm
- senke omkretsen 1 cm

I tillegg finnes

- signal fra sensoren på hjulet
- intern klokke som gir puls hvert millisekund

Grensesnittet er det viktigste i en klasse

Grensesnittet beskrives som metoder:

```
class Sykkelspeedometer {  
    String visFart() { ... }  
    String visDistanse() { ... }  
    void nullstill() { ... }  
  
    String visOmkrets() { ... }  
    void oekOmkrets() { ... }  
    void senkOmkrets() { ... }  
  
    void tellImpuls() { ... }  
    void tellMillisek() { ... }  
    :  
}
```

Det finnes ulike metoder

Hittil har vi brukt metoder for å slippe å gjenta så mye kode. Slike metoder er markert som **static**.

Grensesnittmetodene skal manipulere tilstanden i det enkelte objektet. De er *ikke* markert som static.

Er dette hele sannheten?

Nei; den fulle sannheten åpnebaret om to uker.

Hvordan skal vi representere alt dette?

Representasjon

Når grensesnittet er bestemt, må vi finne ut hvilke data vi må lagre i hvert objekt for at klassen skal fungere slik vi ønsker.

Her finnes det flere mulige alternativer; jeg har valgt:

```
private double omkrets = 0.0; // Hjulets omkrets
private double distanse = 0.0; // Distanse hittil
private double tid = 0.0; // Loepende tid (i ms)
private double tid1 = 0.0; // Tidspunkt siste impuls
private double tid2 = 0.0; // Tidspunkt forrige impuls
```

(Legg merke til at alle er **private**.)

Innkapsling

Når vi programmerer, er det lurt å ha så få globale navn (på variabler, metoder etc) som mulig. Med mange navn er det lettere å miste oversikten og dermed gjøre feil.

En fordel med klasser er at både grensesnittet og implementasjonen defineres *inni* klassene; de er **innkapslet** der. Dermed reduserer vi antall globale navn.

Beskyttelse

Det er mulig å bruke variablene i representasjonen som vanlige variabler, men det anbefales ikke. Det er tryggere om de brukes bare av grensesnittmetodene.

Ved å angi at representasjonsvariablene er **private**, vil Java-kompilatoren hjelpe oss med dette.

Da er klassen ferdig!

Til sist kan vi skrive kode i grensesnittmetodene:

Sykkelspeedometer.java

```
class Sykkelspeedometer {
    private double omkrets = 0.0; // Hjulets omkrets
    private double distanse = 0.0; // Distanse hittil
    private double tid = 0.0; // Loepende tid (i ms)
    private double tid1 = 0.0; // Tidspunkt siste impuls
    private double tid2 = 0.0; // Tidspunkt forrige impuls

    void nullstill() {
        distanse = 0.0;
    }

    void tellMillisek() {
        tid++; // Samme som tid = tid+1;
    }
}
```

Da er klassen ferdig!

```
void tellImpuls() {
    distanse += omkrets;
    // Samme som distanse = distanse+omkrets;
    tid2 = tid1;
    tid1 = tid;
}

void oekOmkrets() {
    omkrets += 0.01;
}

void senkOmkrets() {
    omkrets -= 0.01;
}
```



Da er klassen ferdig!

```
String visDistanse() {  
    return "" + distanse;  
}  
  
String visFart() {  
    double tid = (tid2-tid1) / 1000.0;  
    return "" + distanse/tid/3.6;  
}  
  
String visOmkrets() {  
    return "" + omkrets;  
}  
}
```

Til slutt kan vi lage objekter av klassene våre

NB!

- En **klasse** er bare en mal (dvs en arbeidstegning eller produksjonsbeskrivelse).
- Med **new** lager vi **objekter** (også kalt **instanser**) av klassene. Vi kan lage vilkårlig mange.
- Vi benytter variabler til å holde orden på objektene og utføre metodene.

```
Sykkelspeedometer s = new Sykkelspeedometer();
```

```
s.nullstill();  
s.tellImpuls();
```

TestSykkelspeedometer.java

```
class TestSykkelspeedometer {
    public static void main(String[] arg) {
        Sykkelspeedometer s = new Sykkelspeedometer();

        // ...
        System.out.println("Jeg har syklet " +
            s.visDistanse() + " km.");
        System.out.println("Farten er " +
            s.visFart() + " km/t.");
    }
}
```


Hva er et telleverk?

Eksempel 2: Et telleverk [Horstmann 8.2]

Et telleverk gjør det enkelt å telle ting, for eksempel personer.



Hvordan lager man en klasse?

- 1 Hvilke ting/begreper skal vi modellere?
(Eksamenshint: Hvilke substantiver er brukt i oppgaven?)
- 2 Hva er grensesnittet?
Hvilke operasjoner trenger vi overfor objektene?
(Eksamenshint: Hvilke verb brukes i forbindelse med klassene nevnt i forrige punkt?)
- 3 Hvordan skal klassen implementeres?
Hvilke objektvariabler trenger vi for å lagre nok om tilstanden?
- 4 Lag objekter av klassen for testing.

- ① Hva skal vi modellere?
Svaret er enkelt: et telleverk.

```
class Counter {  
    :  
}
```

2 Hva er grensesnittet?

- Vi vil klikke for å telle.
- Vi ønsker å lese av verdien.

```
class Counter {  
    void count() { ... }  
    int getValue() { ... }  
    :  
}
```

- ③ Hva er representasjonen?
En enkelt teller (en int) er nok.

```
class Counter {  
    private int value;  
  
    void count() { ... }  
    int getValue() { ... }  
    :  
}
```

Den ferdige klassen

Counter.java

```
class Counter {  
    private int value;  
  
    void count() {  
        value++;  
    }  
  
    int getValue() {  
        return value;  
    }  
}
```

Testing

TestCounter.java

```
class TestCounter {
    public static void main (String[] arg) {
        Counter concertCounter = new Counter();
        Counter boardingCounter = new Counter();

        boardingCounter.count();
        boardingCounter.count();
        boardingCounter.count();
        System.out.println("Concert: " +
                           concertCounter.getValue());
        System.out.println("Boarding: " +
                           boardingCounter.getValue());
    }
}
```

```
$ java TestCounter
Concert: 0
Boarding: 3
```



Hvilke typer har vi nevnt hittil?

Datatyper

De viktigste er:

int	et heltall	<code>int n = 5;</code>
long	et stort heltall	<code>long v = 0L;</code>
double	et flyt-tall	<code>double pi = 3.14;</code>
char	et tegn	<code>char c = '?';</code>
boolean	sann/usann	<code>boolean t = true;</code>

Skal vi jobbe med ett eller mange tegn?

Datotypen char

- En **String** kan lagre tekster med vilkårlig antall tegn.
- En **char** kan lagre nøyaktig ett tegn.

Datotypen boolean¹

Vi har brukt tester som er sanne eller usanne i løkker og valgsetninger:

```
if (alder >= 18) ...  
while (n > 0 && m > 0) ...
```

Slike tester må gi et resultat **false** eller **true**.

Vi kan lagre slike verdier i **boolean**-variabler:

```
boolean myndig = alder >= 18;  
  
if (myndig) ...
```

¹George Boole var en engelsk/irsk matematiker.

Klassen String

String er en *nesten* vanlig klasse; den er en del av Javas klassebibliotek så vi kan bruke den til å jobbe med tekster.

Grensesnitt (det viktigste)

- **int compareTo(String other)**
gjør en sammenligning
- **boolean equals(String other)**
tester likhet
- **int length()**
finner tekstens lengde
- **String substring(int begin, int end)**
plukker ut en del av teksten



Det er lett å lage nye String-objekter

Å lage String-objekter

- **new String()**
gir et nytt String-objekt med tomt innhold.
- **new String(s)**
gir et nytt String-objekt der innholdet er en kopi av en annen string s.

NB!

Spesielt for akkurat String:

- **"Abc..."** er et String-objekt.
- **s1 + s2** gir et nytt String-objekt som er de to tekstene skjøtt sammen.

Hvor lang er en tekst?

- **int length()**
forteller oss hvor mange tegn det er i teksten.

Et eksempel

TestStringLength.java

```
class TestStringLength {  
    public static void main (String[] arg) {  
        String g = new String("Grunnkurs");  
        String o = "objektorientert";  
  
        System.out.println("L1: " + g.length());  
        System.out.println("L2: " + "i".length());  
        System.out.println("L3: " + o.length());  
    }  
}
```

```
$ javac TestStringLength.java  
$ java TestStringLength  
L1: 9  
L2: 1  
L3: 15
```



Sammenligning av tekster

- **boolean equals(String other)**
sammenligner to tekster og gir **sann** om de er helt like.
- **int compareTo(String other)**
forteller om den andre teksten kommer før eller etter i en alfabetisk sortering:

$$a.compareTo(b) \text{ gir } \begin{cases} < 0 & \text{om } a \text{ kommer } \textit{før} \text{ } b \\ = 0 & \text{om } a \text{ og } b \text{ er } \textit{like} \\ > 0 & \text{om } a \text{ kommer } \textit{etter} \text{ } b \end{cases}$$

(Dette fungerer egentlig bare for engelsk; for norsk med æøå må man benytte andre teknikker.)

Et eksempel

TestStringSammenligning.java

```
class TestStringSammenligning {
    public static void main(String[] arg) {
        System.out.println("Per vs Per: " + "Per".equals("Per"));
        System.out.println("Per vs per: " + "Per".equals("per"));
        System.out.println("Per vs Ida: " + "Per".compareTo("Ida"));
    }
}
```

```
$ javac TestStringSammenligning.java
$ java TestStringSammenligning
Per vs Per: true
Per vs per: false
Per vs Ida: 7
```


Det er mulig å hente ut deler av tekster

Å hente ut deler av en tekst

- **String substring(a, b)**

lager et nytt String-objekt med innhold fra deler av teksten *fra og med a til (men ikke med) b*.

"smiler".substring(1,4)

s	m	i	l	e	r
---	---	---	---	---	---

 ⇒ "mil"

0 1 2 3 4 5

Parameteren til main er en array med String-er som inneholder parametre oppgitt når vi kjører programmet.

TestStringInitialer.java

```
class TestStringInitialer {  
    public static void main(String[] arg) {  
        String hun = arg[0];  
        String han = arg[1];  
  
        System.out.println(hun.substring(0,1) + " + " +  
                           han.substring(0,1) + " = SANT");  
    }  
}
```

```
$ javac TestStringInitialer.java  
$ java TestStringInitialer Kari Ola  
K + O = SANT
```

Hvor finner jeg nyttige klasser?

Java-biblioteket

Java har et bibliotek med hundrevis av klasser. Heldigvis er det godt beskrevet:

- Appendix D i læreboken
- <http://docs.oracle.com/javase/7/docs/api/>

Hva er grensenettet til String?

The screenshot shows a Mozilla Firefox browser window displaying the Java Platform SE 7 documentation for the `String` class. The browser address bar shows `docs.oracle.com/javase/7/docs/api`. The page title is "String (Java Platform SE 7)".

The left sidebar shows a navigation pane with "All Classes" and "Packages" sections. The "All Classes" section lists various Java classes, including `String`. The "Packages" section lists various Java packages, including `java.lang`.

The main content area displays the following information:

- Overview**: Package, **Class**, Use, Tree, Deprecated, Index, Help.
- Prev Class**, **Next Class**, **Frames**, **No Frames**.
- Summary**: Nested | Field | Const | Method | Detail | Field | Const | Method.
- Class String**
- java.lang Object**
 - java.lang.String
- All Implemented Interfaces:**
 - Serializable, CharSequence, Comparable<String>
- public final class String**
 - extends Object
 - implements Serializable, Comparable<String>, CharSequence
- The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
- Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:


```
String str = "abc";
```
- is equivalent to:


```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```
- Here are some more examples of how strings can be used:


```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```
- The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.
- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String conversions are implemented through the method toString, defined by Object and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.
- Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.
- A String represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs (see the section Unicode Character Representations in the Character class for more information). Index values refer to char code units, so a supplementary character uses two positions in a String.
- The String class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., char values).
- Since:**
 - 1.0
 - 1.1
 - 1.2
 - 1.3
 - 1.4
 - 1.5
 - 1.6
 - 1.7
 - 1.8
 - 1.9
 - 10
 - 11
 - 12
 - 13
 - 14
 - 15
 - 16
 - 17
 - 18
 - 19
 - 20
 - 21
 - 22
 - 23
 - 24
 - 25
 - 26
 - 27
 - 28
 - 29
 - 30
 - 31
 - 32
 - 33
 - 34
 - 35
 - 36
 - 37
 - 38
 - 39
 - 40
 - 41
 - 42
 - 43
 - 44
 - 45
 - 46
 - 47
 - 48
 - 49
 - 50
 - 51
 - 52
 - 53
 - 54
 - 55
 - 56
 - 57
 - 58
 - 59
 - 60
 - 61
 - 62
 - 63
 - 64
 - 65
 - 66
 - 67
 - 68
 - 69
 - 70
 - 71
 - 72
 - 73
 - 74
 - 75
 - 76
 - 77
 - 78
 - 79
 - 80
 - 81
 - 82
 - 83
 - 84
 - 85
 - 86
 - 87
 - 88
 - 89
 - 90
 - 91
 - 92
 - 93
 - 94
 - 95
 - 96
 - 97
 - 98
 - 99
 - 100
 - 101
 - 102
 - 103
 - 104
 - 105
 - 106
 - 107
 - 108
 - 109
 - 110
 - 111
 - 112
 - 113
 - 114
 - 115
 - 116
 - 117
 - 118
 - 119
 - 120
 - 121
 - 122
 - 123
 - 124
 - 125
 - 126
 - 127
 - 128
 - 129
 - 130
 - 131
 - 132
 - 133
 - 134
 - 135
 - 136
 - 137
 - 138
 - 139
 - 140
 - 141
 - 142
 - 143
 - 144
 - 145
 - 146
 - 147
 - 148
 - 149
 - 150
 - 151
 - 152
 - 153
 - 154
 - 155
 - 156
 - 157
 - 158
 - 159
 - 160
 - 161
 - 162
 - 163
 - 164
 - 165
 - 166
 - 167
 - 168
 - 169
 - 170
 - 171
 - 172
 - 173
 - 174
 - 175
 - 176
 - 177
 - 178
 - 179
 - 180
 - 181
 - 182
 - 183
 - 184
 - 185
 - 186
 - 187
 - 188
 - 189
 - 190
 - 191
 - 192
 - 193
 - 194
 - 195
 - 196
 - 197
 - 198
 - 199
 - 200
 - 201
 - 202
 - 203
 - 204
 - 205
 - 206
 - 207
 - 208
 - 209
 - 210
 - 211
 - 212
 - 213
 - 214
 - 215
 - 216
 - 217
 - 218
 - 219
 - 220
 - 221
 - 222
 - 223
 - 224
 - 225
 - 226
 - 227
 - 228
 - 229
 - 230
 - 231
 - 232
 - 233
 - 234
 - 235
 - 236
 - 237
 - 238
 - 239
 - 240
 - 241
 - 242
 - 243
 - 244
 - 245
 - 246
 - 247
 - 248
 - 249
 - 250
 - 251
 - 252
 - 253
 - 254
 - 255
 - 256
 - 257
 - 258
 - 259
 - 260
 - 261
 - 262
 - 263
 - 264
 - 265
 - 266
 - 267
 - 268
 - 269
 - 270
 - 271
 - 272
 - 273
 - 274
 - 275
 - 276
 - 277
 - 278
 - 279
 - 280
 - 281
 - 282
 - 283
 - 284
 - 285
 - 286
 - 287
 - 288
 - 289
 - 290
 - 291
 - 292
 - 293
 - 294
 - 295
 - 296
 - 297
 - 298
 - 299
 - 300
 - 301
 - 302
 - 303
 - 304
 - 305
 - 306
 - 307
 - 308
 - 309
 - 310
 - 311
 - 312
 - 313
 - 314
 - 315
 - 316
 - 317
 - 318
 - 319
 - 320
 - 321
 - 322
 - 323
 - 324
 - 325
 - 326
 - 327
 - 328
 - 329
 - 330
 - 331
 - 332
 - 333
 - 334
 - 335
 - 336
 - 337
 - 338
 - 339
 - 340
 - 341
 - 342
 - 343
 - 344
 - 345
 - 346
 - 347
 - 348
 - 349
 - 350
 - 351
 - 352
 - 353
 - 354
 - 355
 - 356
 - 357
 - 358
 - 359
 - 360
 - 361
 - 362
 - 363
 - 364
 - 365
 - 366
 - 367
 - 368
 - 369
 - 370
 - 371
 - 372
 - 373
 - 374
 - 375
 - 376
 - 377
 - 378
 - 379
 - 380
 - 381
 - 382
 - 383
 - 384
 - 385
 - 386
 - 387
 - 388
 - 389
 - 390
 - 391
 - 392
 - 393
 - 394
 - 395
 - 396
 - 397
 - 398
 - 399
 - 400
 - 401
 - 402
 - 403
 - 404
 - 405
 - 406
 - 407
 - 408
 - 409
 - 410
 - 411
 - 412
 - 413
 - 414
 - 415
 - 416
 - 417
 - 418
 - 419
 - 420
 - 421
 - 422
 - 423
 - 424
 - 425
 - 426
 - 427
 - 428
 - 429
 - 430
 - 431
 - 432
 - 433
 - 434
 - 435
 - 436
 - 437
 - 438
 - 439
 - 440
 - 441
 - 442
 - 443
 - 444
 - 445
 - 446
 - 447
 - 448
 - 449
 - 450
 - 451
 - 452
 - 453
 - 454
 - 455
 - 456
 - 457
 - 458
 - 459
 - 460
 - 461
 - 462
 - 463
 - 464
 - 465
 - 466
 - 467
 - 468
 - 469
 - 470
 - 471
 - 472
 - 473
 - 474
 - 475
 - 476
 - 477
 - 478
 - 479
 - 480
 - 481
 - 482
 - 483
 - 484
 - 485
 - 486
 - 487
 - 488
 - 489
 - 490
 - 491
 - 492
 - 493
 - 494
 - 495
 - 496
 - 497
 - 498
 - 499
 - 500
 - 501
 - 502
 - 503
 - 504
 - 505
 - 506
 - 507
 - 508
 - 509
 - 510
 - 511
 - 512
 - 513
 - 514
 - 515
 - 516
 - 517
 - 518
 - 519
 - 520
 - 521
 - 522
 - 523
 - 524
 - 525
 - 526
 - 527
 - 528
 - 529
 - 530
 - 531
 - 532
 - 533
 - 534
 - 535
 - 536
 - 537
 - 538
 - 539
 - 540
 - 541
 - 542
 - 543
 - 544
 - 545
 - 546
 - 547
 - 548
 - 549
 - 550
 - 551
 - 552
 - 553
 - 554
 - 555
 - 556
 - 557
 - 558
 - 559
 - 560
 - 561
 - 562
 - 563
 - 564
 - 565
 - 566
 - 567
 - 568
 - 569
 - 570
 - 571
 - 572
 - 573
 - 574
 - 575
 - 576
 - 577
 - 578
 - 579
 - 580
 - 581
 - 582
 - 583
 - 584
 - 585
 - 586
 - 587
 - 588
 - 589
 - 590
 - 591
 - 592
 - 593
 - 594
 - 595
 - 596
 - 597
 - 598
 - 599
 - 600
 - 601
 - 602
 - 603
 - 604
 - 605
 - 606
 - 607
 - 608
 - 609
 - 610
 - 611
 - 612
 - 613
 - 614
 - 615
 - 616
 - 617
 - 618
 - 619
 - 620
 - 621
 - 622
 - 623
 - 624
 - 625
 - 626
 - 627
 - 628
 - 629
 - 630
 - 631
 - 632
 - 633
 - 634
 - 635
 - 636
 - 637
 - 638
 - 639
 - 640
 - 641
 - 642
 - 643
 - 644
 - 645
 - 646
 - 647
 - 648
 - 649
 - 650
 - 651
 - 652
 - 653
 - 654
 - 655
 - 656
 - 657
 - 658
 - 659
 - 660
 - 661
 - 662
 - 663
 - 664
 - 665
 - 666
 - 667
 - 668
 - 669
 - 670
 - 671
 - 672
 - 673
 - 674
 - 675
 - 676
 - 677
 - 678
 - 679
 - 680
 - 681
 - 682
 - 683
 - 684
 - 685
 - 686
 - 687
 - 688
 - 689
 - 690
 - 691
 - 692
 - 693
 - 694
 - 695
 - 696
 - 697
 - 698
 - 699
 - 700
 - 701
 - 702
 - 703
 - 704
 - 705
 - 706
 - 707
 - 708
 - 709
 - 710
 - 711
 - 712
 - 713
 - 714
 - 715
 - 716
 - 717
 - 718
 - 719
 - 720
 - 721
 - 722
 - 723
 - 724
 - 725
 - 726
 - 727
 - 728
 - 729
 - 730
 - 731
 - 732
 - 733
 - 734
 - 735
 - 736
 - 737
 - 738
 - 739
 - 740
 - 741
 - 742
 - 743
 - 744
 - 745
 - 746
 - 747
 - 748
 - 749
 - 750
 - 751
 - 752
 - 753
 - 754
 - 755
 - 756
 - 757
 - 758
 - 759
 - 760
 - 761
 - 762
 - 763
 - 764
 - 765
 - 766
 - 767
 - 768
 - 769
 - 770
 - 771
 - 772
 - 773
 - 774
 - 775
 - 776
 - 777
 - 778
 - 779
 - 780
 - 781
 - 782
 - 783
 - 784
 - 785
 - 786
 - 787
 - 788
 - 789
 - 790
 - 791
 - 792
 - 793
 - 794
 - 795
 - 796
 - 797
 - 798
 - 799
 - 800
 - 801
 - 802
 - 803
 - 804
 - 805
 - 806
 - 807
 - 808
 - 809
 - 810
 - 811
 - 812
 - 813
 - 814
 - 815
 - 816
 - 817
 - 818
 - 819
 - 820
 - 821
 - 822
 - 823
 - 824
 - 825
 - 826
 - 827
 - 828
 - 829
 - 830
 - 831
 - 832
 - 833
 - 834
 - 835
 - 836
 - 837
 - 838
 - 839
 - 840
 - 841
 - 842
 - 843
 - 844
 - 845
 - 846
 - 847
 - 848
 - 849
 - 850
 - 851
 - 852
 - 853
 - 854
 - 855
 - 856
 - 857
 - 858
 - 859
 - 860
 - 861
 - 862
 - 863
 - 864
 - 865
 - 866
 - 867
 - 868
 - 869
 - 870
 - 871
 - 872
 - 873
 - 874
 - 875
 - 876
 - 877
 - 878
 - 879
 - 880
 - 881
 - 882
 - 883
 - 884
 - 885
 - 886
 - 887
 - 888
 - 889
 - 890
 - 891
 - 892
 - 893
 - 894
 - 895
 - 896
 - 897
 - 898
 - 899
 - 900
 - 901
 - 902
 - 903
 - 904
 - 905
 - 906
 - 907
 - 908
 - 909
 - 910
 - 911
 - 912
 - 913
 - 914
 - 915
 - 916
 - 917
 - 918
 - 919
 - 920
 - 921
 - 922
 - 923
 - 924
 - 925
 - 926
 - 927
 - 928
 - 929
 - 930
 - 931
 - 932
 - 933
 - 934
 - 935
 - 936
 - 937
 - 938
 - 939
 - 940
 - 941
 - 942
 - 943
 - 944
 - 945
 - 946
 - 947
 - 948
 - 949
 - 950
 - 951
 - 952
 - 953
 - 954
 - 955
 - 956
 - 957
 - 958
 - 959
 - 960
 - 961
 - 962
 - 963
 - 964
 - 965
 - 966
 - 967
 - 968
 - 969
 - 970
 - 971
 - 972
 - 973
 - 974
 - 975
 - 976
 - 977
 - 978
 - 979
 - 980
 - 981
 - 982
 - 983
 - 984
 - 985
 - 986
 - 987
 - 988
 - 989
 - 990
 - 991
 - 992
 - 993
 - 994
 - 995
 - 996
 - 997
 - 998
 - 999
 - 1000

Husk at Java-biblioteket er stort!

Hint:

Ikke gå dere vill i biblioteket!

Java-biblioteket er fascinerende lesing, men

- Slå opp når dere trenger svar på konkrete spørsmål (som «Hva angir 2. parameter til `String.substring`?»).
- Unngå leting à la «Jeg vet ikke helt hva jeg trenger, men kanskje jeg finner noe jeg kan bruke?»
- Let mer en gang dere har bedre tid.

Hva er det viktigste dere skal ha lært i dag?

Hovedpunkter

- Med **klasser** kan man **modellere** både virkelige ting og nyttige begreper.
- En klasse er en mal/konstruksjonstegning av det som skal modelleres.
- **Grensesnittet** til klassen angis med metoder.
- Vi må finne ut hvilken **representasjon** (dvs hvilke objektvariabler) som er nødvendig.
- Med **new** kan vi lage **objekter** basert på klassene.
- En variabel kan peke på et objekt; ved hjelp av den kan vi utføre metodene i grensesnittet.