Det här är ett program ...

```
cafebabe 00000031 00270700 0201000a
44726177 53717561 72650700 04010010
6a617661 2f6c616e 672f4f62 6a656374
0100063c 696e6974 3e010003 28295601
0004436f 64650a00 0300090c 00050006
01000f4c 696e654e 756d6265 72546162
6c650100 124c6f63 616c5661 72696162
6c655461 626c6501 00047468 69730100
0c4c4472 61775371 75617265 3b010004
6d61696e 01001628 5b4c6a61 76612f6c
616e672f 53747269 6e673b29 56070011
01002273 652f6c74 682f6373 2f707464
632f7769 6e646f77 2f53696d 706c6557
696e646f 77080002 0a001000 140c0005
00150100 17284949 4c6a6176 612f6c61
6e672f53 7472696e 673b2956 07001701
001c7365 2f6c7468 2f63732f 70746463
2f737175 6172652f 53717561 72650a00
1600190c 0005001a 01000628 ...
```


Det här är samma program

```
import se.lth.cs.ptdc.window.SimpleWindow;
import se.lth.cs.ptdc.square.Square;

public class DrawSquare {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(600, 600, "DrawSquare");
        Square sq = new Square(300, 300, 200);
        sq.draw(w);
    }
}
```


Datorns delar

Primärminne, där man lagrar instruktionerna i programmet och de data som programmet arbetar med.

Processor, eller CPU ("Central Processing Unit"), som är den enhet i datorn som utför instruktioner i datorns maskinspråk.

Sekundärminne, till exempel skivminne, där man kan lagra program och data så att man kan använda dem senare.

Kommunikationskanaler, som ser till att datorn kan kommunicera med andra datorer eller med andra apparater som är kopplade till datorn.

Maskinspråk och högnivåspråk

Maskinspråk (assembler, påhittat):

```
LOAD   R1, 44670 // hämta innehållet i minnescellen
                 // med adressen 44670 till register R1
LOAD   R2, 44674 // och innehållet i 44674 till R2
ADD    R1, R2    // addera, resultatet i R1
MULT   R1, #10   // multiplicera med konstanten 10
STORE  R1, 44678 // lagra resultatet i 44678
```

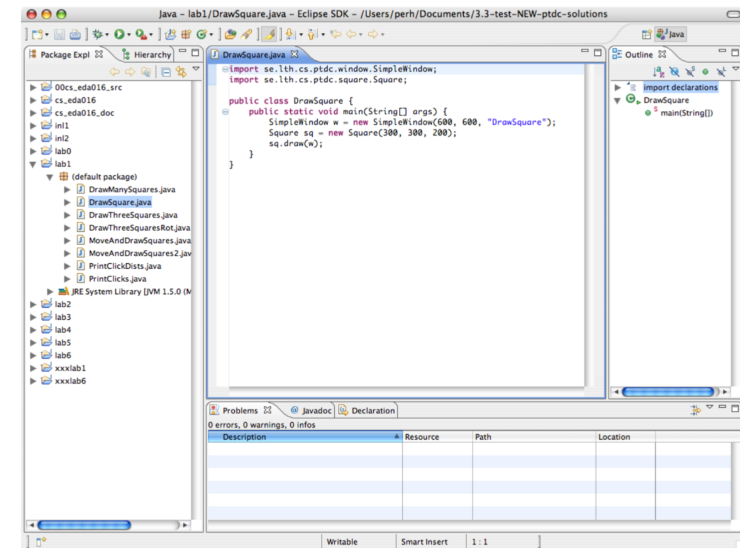
Högnivåspråk (Java):

```
sum = (x + y) * 10;
```

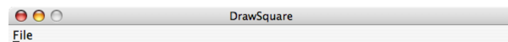
Objektorienterad programmering

- Programmering är att lösa problem och beskriva lösningen i ett program skrivet i ett programspråk.
- I "verkligheten" finns "saker" som motsvaras av *objekt* i ett program som körs. Bilar → bilobjekt, gator → gatuobjekt, hundar → hundobjekt, ...
- I programmet beskriver man objektens egenskaper i *klasser*. `class Car { ... }`, `class Road { ... }`, `class Dog { ... }` ...

Programutveckling med Eclipse

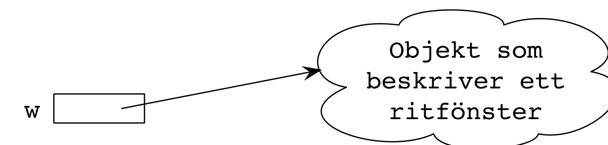


Exekvering av program

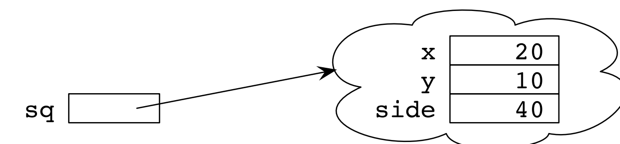


Objekt, attribut och referensvariabler

Fönsterobjekt:

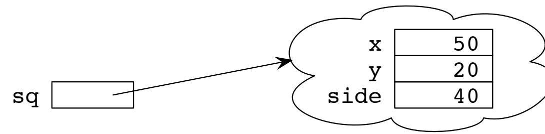


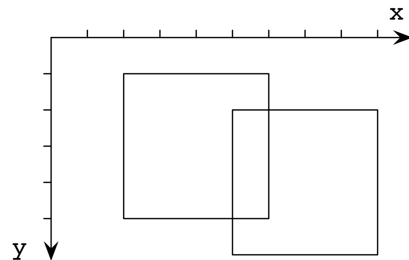
Kvadratobjekt:



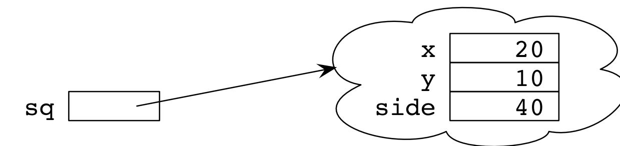
Operationer

```
sq.draw(w);
sq.move(30, 10);
sq.draw(w);
```





Objekt i datorns minne



...		
5200	47540	sq
...		
47540	20	x
47544	10	y
47548	40	side
...		

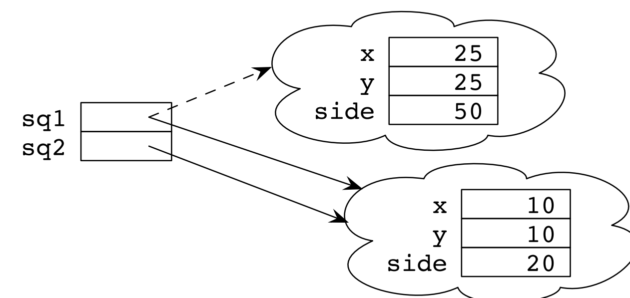
Skapa objekt

```
Square sq = new Square(20, 10, 40);
```

- `Square sq` betyder att referensvariabeln `sq` *deklareras*.
- `new Square(...)` skapar ett kvadratobjekt. Resultatet är en referens till det nyskapade objektet.
- Likhetstecknet utläser man "tilldelas". Satsen lagrar värdet av `new`-uttrycket, alltså referensen till objektet, i variabeln `sq`.
- Ordet `new` är ett reserverat ord i Java.

Referenstilldelning

```
Square sq1 = new Square(25, 25, 50);
Square sq2 = new Square(10, 10, 20);
sq1 = sq2;
```



Specifikation av kvadratklass

```
/** Skapar en kvadrat med övre vänstra hörnet i x,y
    och med sidlängden side */
Square(int x, int y, int side);

/** Ritar kvadraten i fönstret w */
void draw(SimpleWindow w);

/** Flyttar kvadraten avståndet dx i x-led,
    dy i y-led */
void move(int dx, int dy);
```

```
Square sq = new Square(20, 10, 40);
sq.draw(w);
sq.move(30, 10);
sq.draw(w);
```


Funktioner

Klassen Square, funktioner.

```
/** Tar reda på x-koordinaten för kvadratens läge */
int getX();

/** Tar reda på y-koordinaten för kvadratens läge */
int getY();

/** Tar reda på kvadratens area */
int getArea();
```

```
Square sq = new Square(20, 10, 40);
System.out.println("Kvadratens läge: " + sq.getX() +
                   ", " + sq.getY());
System.out.println("Kvadratens area: " + sq.getArea());
```


Ny klass — bankkonto

BankAccount

```
/** Skapar ett bankkonto med numret accntNbr och
    saldot noll */
BankAccount(int accntNbr);

/** Tar reda på kontonumret */
int getAccntNbr();

/** Tar reda på saldot */
int getBalance();

/** Sätter in amount kronor på kontot */
void deposit(int amount);

/** Tar ut amount kronor från kontot */
void withdraw(int amount);
```


Användning av BankAccount

```
public class BankTest {
    public static void main(String[] args) {
        BankAccount myAccount = new BankAccount(12379);
        myAccount.deposit(1500);
        myAccount.withdraw(350);
        System.out.println("På konto " +
                           myAccount.getAccntNbr());
        System.out.println("finns " +
                           myAccount.getBalance() +
                           " kr");
    }
}
```

Variabler, uttryck och satser

```
int a = 100;            // a får värdet 100
int b = 5 * a + 1;      // b får värdet 5 * 100 + 1 = 501
a = a + 1;              // a får värdet 100 + 1 = 101,
                        // utläses "a ökas med 1"
b = 2 * (a - 100) + b; // b får värdet 2*(101-100)+501
                        // = 503
```

	Efter `a = 100; b = 5 * a + 1`	Efter `a = a + 1`	Efter `b = 2 * (a - 100) + b`
a	100	101	101
b	501	501	503

if-satser

```
int a = ...;
int b = ...;

if (a >= b + 1) {
    a = a + 1;          // utförs om a >= b + 1
} else {
    a = a - 1;          // utförs om a < b + 1
}

if (a == 1 && b == 1) { // && betyder "och"
    a = 0;              // båda dessa satser utförs
    b = 0;              // om a = 1 och b = 1
}
```


while-satser och for-satser

Repetera ett obestämt antal gånger:

```
Square sq = new Square(100, 200, 25);
while (sq.getX() > 0) {
    sq.draw(w);
    sq.move(-10, -10);
}
```

Repetera ett bestämt antal gånger:

```
Square sq = new Square(100, 200, 25);
for (int i = 0; i < 5; i++) {
    sq.draw(w);
    sq.move(-10, -10);
}
```


Implementering av klass

Utgå från specifikationen:

- Skriv `public class { ... }` runtom.
- Deklarera attribut.
- Ge attributen startvärden i konstruktorn.
- Implementera metoderna.
- Skriv `public` på metoderna, `private` på attributen.

Specifikation av Square

```
/** Skapar en kvadrat med övre vänstra hörnet i x,y
    och med sidlängden side  */
Square(int x, int y, int side);

/** Ritar kvadraten i fönstret w */
void draw(SimpleWindow w);

/** Flyttar kvadraten avståndet dx i x-led,
    dy i y-led */
void move(int dx, int dy);

/** Tar reda på x-koordinaten för kvadratens läge */
int getX();

/** Tar reda på y-koordinaten för kvadratens läge */
int getY();

/** Tar reda på kvadratens area */
int getArea();
```


Implementering av Square, 1

```
public class Square {
    private int x;    // x- och y-koordinat för
    private int y;    // övre vänstra hörnet
    private int side; // sidlängd

    /** Skapar en kvadrat med övre vänstra hörnet i x,y
        och med sidlängden side */
    public Square(int x, int y, int side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }
```


Implementering av Square, 2

```
    /** Ritar kvadraten i fönstret w */
    public void draw(SimpleWindow w) {
        w.moveTo(x, y);
        w.lineTo(x, y + side);
        w.lineTo(x + side, y + side);
        w.lineTo(x + side, y);
        w.lineTo(x, y);
    }

    /** Flyttar kvadraten avståndet dx i x-led,
        dy i y-led */
    public void move(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
```


Implementering av Square, 3

```
    /** Tar reda på x-koordinaten för kvadratens läge */
    public int getX() {
        return x;
    }

    /** Tar reda på y-koordinaten för kvadratens läge */
    public int getY() {
        return y;
    }

    /** Tar reda på kvadratens area */
    public int getArea() {
        return side * side;
    }
}
```

Specifikation av BankAccount

BankAccount

```
/** Skapar ett bankkonto med numret accntNbr och
    saldot noll */
BankAccount(int accntNbr);

/** Tar reda på kontonumret */
int getAccntNbr();

/** Tar reda på saldot */
int getBalance();

/** Sätter in amount kronor på kontot */
void deposit(int amount);

/** Tar ut amount kronor från kontot */
void withdraw(int amount);
```


Implementering av BankAccount, 1

```
public class BankAccount {
    private int accntNbr; // kontonummer
    private int balance;  // saldo

    /** Skapar ett bankkonto med numret accntNbr och
        saldot noll */
    public BankAccount(int accntNbr) {
        this.accntNbr = accntNbr;
        this.balance = 0;
    }

    /** Tar reda på kontonumret */
    public int getAccntNbr() {
        return accntNbr;
    }
```


Implementering av BankAccount, 2

```
    /** Tar reda på saldot */
    public int getBalance() {
        return balance;
    }

    /** Sätter in amount kronor på kontot */
    public void deposit(int amount) {
        balance = balance + amount;
    }

    /** Tar ut amount kronor från kontot */
    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```


Konstruktorer

Konstruktorn ska initiera (ge startvärden till) alla attributen. Man skiljer på parametern och attributet genom att skriva `this.` före attributnamnet.

```
/** Skapar en kvadrat med övre vänstra hörnet i x,y
    och med sidlängden side */
public Square(int x, int y, int side) {
    this.x = x;
    this.y = y;
    this.side = side;
}

/** Skapar ett bankkonto med numret accntNbr och
    saldot noll */
public BankAccount(int accntNbr) {
    this.accntNbr = accntNbr;
    this.balance = 0; // eller balance = 0
}
```

Bankkonto med ränta

- Attributet `interestRate` anger räntesatsen i procent för kontot.
- Den ackumulerade räntan (attributet `interest`) ökas genom anrop av metoden `computeInterest` varje gång som saldot förändras genom insättning eller uttag.
- Attributet `lastInterestDay` anger numret på den dag (med början på nummer 1 för den 1 januari) då räntan senast beräknades.
- Vi förutsätter att vi har tillgång till en variabel `Date.today` som ger numret på aktuell dag.
- I slutet av ett år läggs den ackumulerade räntan till saldot när någon "utifrån" utför operationen `newYearActions`.

Implementering av BankAccountWithInterest, 1

```
public class BankAccountWithInterest {
    private int accntNbr;          // kontonummer
    private int balance;           // saldo
    private double interestRate; // räntesats i procent
    private double interest;       // ackumulerad ränta
                                   // under året
    private int lastInterestDay; // dagnummer för senaste
                                   // ränteberäkning

    ...

    public void deposit(int amount) {
        computeInterest();
        balance = balance + amount;
    }
```


Implementering av BankAccountWithInterest, 2

```
    /** Adderar årets ränta till saldot. Ska utföras
        vid årsskifte */
    public void newYearActions() {
        computeInterest();
        balance = balance + (int) Math.round(interest);
        interest = 0;
        lastInterestDay = 1;
    }

    /** Adderar räntan sedan föregående insättning
        eller uttag */
    private void computeInterest() {
        interest = interest + interestRate / 100.0 *
                   (Date.today - lastInterestDay) /
                   360 * balance;
        lastInterestDay = Date.today;
    }
}
```


Representation av tillstånd

Läget hos en kvadrat kan representeras av ett `Point`-objekt i stället för av koordinaterna `x` och `y`:

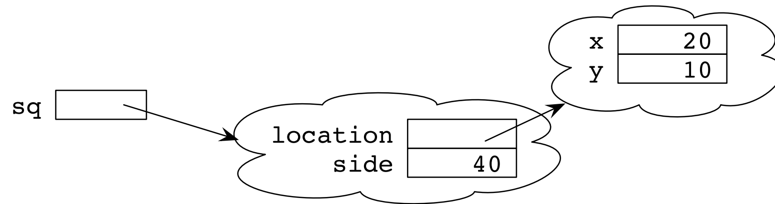
```
private Point location; // övre vänstra hörnet
private int side;       // sidlängd
```

```
/** Skapar en punkt med koordinaterna x, y */
Point(int x, int y);

/** Tar reda på x-koordinaten */
int getX();

/** Tar reda på y-koordinaten */
int getY();

/** Flyttar punkten avståndet dx i x-led,
    dy i y-led */
void move(int dx, int dy);
```

Ny Square med Point-objekt som anger läget

Square-objekten används på samma sätt som tidigare, men de ser annorlunda ut "inuti".

```
Square sq = new Square(20, 10, 40);
```



Implementering av ny Square, 1

```
/** Skapar en kvadrat med övre vänstra hörnet i x,y
    och med sidlängden side */
public Square(int x, int y, int side) {
    this.location = new Point(x, y);
    this.side = side;
}

public void draw(SimpleWindow w) {
    w.moveTo(location.getX(), location.getY());
    w.lineTo(location.getX(), location.getY() + side);
    w.lineTo(location.getX() + side,
             location.getY() + side);
    w.lineTo(location.getX() + side, location.getY());
    w.lineTo(location.getX(), location.getY());
}
```


Implementering av ny Square, 2

draw-variant med lokala variabler x och y:

```
public void draw(SimpleWindow w) {
    int x = location.getX();
    int y = location.getY();
    w.moveTo(x, y);
    w.lineTo(x, y + side);
    w.lineTo(x + side, y + side);
    w.lineTo(x + side, y);
    w.lineTo(x, y);
}
```


Implementering av ny Square, 3

Inte svåra:

```
public int getX() {
    return location.getX();
}

public int getY() {
    return location.getY();
}
```

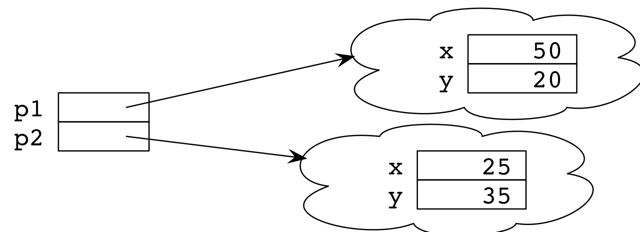
I move måste man utnyttja move-operationen i Point för att flytta kvadraten:

```
public void move(int dx, int dy) {
    location.move(dx, dy);
}
```

Objekt som parameter

```
/** Beräknar avståndet mellan denna punkt och
    punkten p */
double getDistanceTo(Point p);
```

```
Point p1 = new Point(50, 20);
Point p2 = new Point(25, 35);
double dist = p1.getDistanceTo(p2);
System.out.println("Avståndet mellan punkterna är " +
                   dist);
```



Implementering av getDistanceTo

Använd bara metoderna i Point:

```
public double getDistanceTo(Point p) {
    int xDist = getX() - p.getX();
    int yDist = getY() - p.getY();
    return Math.sqrt(xDist * xDist + yDist * yDist);
}
```

Utnyttja kunskapen om att Point har attributen x och y:

```
public double getDistanceTo(Point p) {
    int xDist = x - p.x;
    int yDist = y - p.y;
    return Math.sqrt(xDist * xDist + yDist * yDist);
}
```


Delade objekt

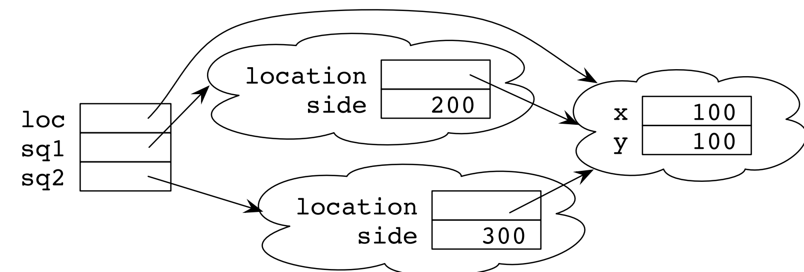
Point-objektet som definierar kvadratens läge kan skapas utanför objektet och skickas med som en parameter till en ny konstruktor:

```
public Square(Point location, int side) {
    this.location = location;
    this.side = side;
}
```

Nu kan flera kvadrater dela samma läge:

```
Point loc = new Point(100, 100);
Square sq1 = new Square(loc, 200);
Square sq2 = new Square(loc, 300);
```


Delade objekt — var försiktig!



Nu kan man flytta båda kvadraterna med loc.move(25,30). Men observera att sq1.move(25,30) också medför att båda kvadraterna flyttas!

Objekt som funktionsresultat

Metoder för att ta reda på kvadratens läge:

```
public int getX() {
    return location.getX();
}

public int getY() {
    return location.getY();
}
```

Man kan skriva en metod som ger `Point`-objektet som resultat, men då ger man användaren tillgång till en implementeringsdetalj så det brukar man inte göra:

```
public Point getLocation() {
    return location;
}
```


Algoritmexempel: summering

Summera ett antal värden:

```
sum = 0;
för alla termer {
    term = "nästa term";
    sum = sum + term;
}
```

Exempel 1: läs `n`-värde, läs `n` tal, summera och skriv ut.

```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt(); // läs antalet tal
int sum = 0;
for (int i = 0; i < n; i++) {
    int term = scan.nextInt(); // läs nästa tal
    sum = sum + term;
}
System.out.println(sum);
```


Mera summering

Exempel 2: läs tills negativt tal påträffas, summera.

```
Scanner scan = new Scanner(System.in);
int sum = 0;
int nbr = scan.nextInt();
while (nbr >= 0) {
    sum = sum + nbr;
    nbr = scan.nextInt();
}
```

Exempel 3: Beräkna summan $\sum_{i=1}^{100} \frac{1}{i * i}$.

```
double sum = 0;
for (int i = 1; i <= 100; i++) {
    sum = sum + 1.0 / (i * i);
}
```


Statiska attribut och metoder

- "Vanligt" attribut: varje objekt har sin egen uppsättning av attributen som definieras i klassen.
- Statiskt attribut: finns bara i *en* upplaga för varje klass.
- Man kommer åt en statisk storhet med `Klassnamn.namn`, *inte* med `referens.namn`.

Exempel: `Math.PI`, `Math.sin(x)`.

```
public class Math {
    public static final double
        PI = 3.14159265358979323846;

    public static double sin(double x) {
        // beräkning av sinus för x
    }
}
```

Överlagring av metoder och konstruktorer, 1

```
public class BankAccount {
    private int accntNbr;
    private int balance;

    /** Skapar ett bankkonto med numret accntNbr och
        saldot noll */
    public BankAccount(int accntNbr) {
        this.accntNbr = accntNbr;
        this.balance = 0;
    }

    /** Skapar ett bankkonto med numret accntNbr och
        saldot balance */
    public BankAccount(int accntNbr, int balance) {
        this.accntNbr = accntNbr;
        this.balance = balance;
    }
```

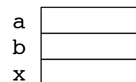

Överlagring av metoder och konstruktorer, 2

```
    /** Skapar ett bankkonto med samma kontonummer och
        saldo som kontot orig */
    public BankAccount(BankAccount orig) {
        this.accntNbr = orig.accntNbr;
        this.balance = orig.balance;
    }
}

BankAccount accnt1 = new BankAccount(47733);
BankAccount accnt2 = new BankAccount(36622, 1000);
BankAccount accnt3 = new BankAccount(accnt1);
```


Lokala variabler och parametrar

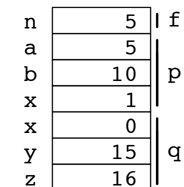
Vid varje anrop av en metod reserveras plats för metodens parametrar och för metodens lokala variabler i en *aktiveringspost* i minnet. När metoden lämnas stryks aktiveringsposten.

```
public static void p(int a, int b) {
    int x = 1;
    q(x, a + b);
}
```



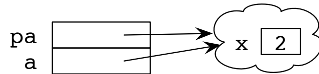
Aktiveringsposter, större exempel

```
public static void f() {
    int n = 5;
    p(n, 2 * n);
}

public static void p(int a, int b) {
    int x = 1;
    q(x, a + b);
}

public static void q(int x, int y) {
    int z = x + y;
    x = 0;
    ...
}
```



Aktiveringsposter, referensparametrar

```
public static void f() {
    A pa = new A(1);
    g(pa);
}

public static void g(A a) {
    a.setX(2);
}

public class A {
    private int x;
    public A(int x) {
        this.x = x;
    }
    public void setX(int newX) {
        x = newX;
    }
}
```



Program, namn, indragningar, ...

```
import se.lth.cs.ptdc.window.SimpleWindow;
import se.lth.cs.ptdc.square.Square;

public class DrawSquares {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(600, 600, "DrawSquares");
        Square sq = new Square(100, 200, 25);
        for (int i = 0; i < 5; i++) {
            sq.draw(w);
            sq.move(-10, -10);
        }
    }
}
```


Datatyper

Datatyper i Java:

Typ	Betydelse
`byte`, `short`, `int`, `long`	heltal
`float`, `double`	reellt tal (tal med decimaldel)
`boolean`	logiskt värde (sant eller falskt)
`char`	tecken, till exempel bokstav, siffra, specialtecken
`referens`	referens till objekt

Heltalstypernas storlek:

Typ	Bitar	Min	Max
`byte`	8	-128	$+127$
`short`	16	-32768	$+32767$
`int`	32	-2147483648	$+2147483647$
`long`	64	$\approx -9 \cdot 10^{18}$	$\approx +9 \cdot 10^{18}$

Deklarationer

```
typ namn = startvärde;
```

- Om startvärdet utelämnas blir lokala variabler odefinierade, attribut får implicit startvärde (0, 0.0, ...).
- Om det i en klass finns flera förekomster av samma namn så gäller den ”närmaste” deklarationen:

```
public class A {
    private int x; // attribut

    public void p() {
        int x = 0; // lokal variabel i metoden p
        // alla förekomster av "x" avser här den lokala
        // variabeln x. Om man vill komma åt attributet
        // x skriver man this.x
    }
}
```

Lokala variabler, konstanter

- Deklarera variabler så sent som möjligt, omedelbart innan de används första gången.
- Bara de storheter som behövs för att beskriva tillståndet hos objekt ska vara attribut.
- Deklaration av konstanter:

```
public class CardGame {
    private final static int MAX_PLAYERS = 10;
    ...
}
```


Tilldelningssatser

```
variabel = nytt värde;
```

- Det nya värdet är ett uttryck. Variabeln och uttrycket ska ha samma (eller kompatibel) typ.
- Om variabeln är "större" än uttryckets värde konverteras det nya värdet automatiskt till variabelns typ.
- Om variabeln är "mindre" måste man konvertera explicit:

```
int i = 100;
double d = 314.61;
short s = (short) i; // värdet av i konverteras
                     // till short
i = (int) d;         // värdet av d konverteras
                     // till int, 314.61 -> 314
```


Aritmetiska uttryck

Heltalsuttryck:

```
int a = 0;
int b = 12;
int c = 20;
a = 2 * (b + c) + 4; // a = 68
b = a / 10;          // b = 6 (6.8, decimalerna stryks)
c = a % 10;          // c = 8 (68/10 = 6 + 8/10,
                     // 8 är resten)
```

Reella uttryck:

```
double x = 1.4;
double y = 1 + 2 * (x + 1); // y = 5.8
double z = x * x + y * y;   // z = 35.60
z = z / 10;                 // z = 3.56
int a = 5;
x = 1 + (double) a / 2;     // x = 3.5
```


Minsta och största värde, standardfunktioner

```
short smin = Short.MIN_VALUE;   // smin = -32768
int imax = Integer.MAX_VALUE;   // imax = 2147483647
double dmin = Double.MIN_VALUE; // dmin = 4.9E-324
double dmax = Double.MAX_VALUE; // dmax = 1.8E308
```

`long round(double x);`	avrundning
`int abs(int x);`	$\lvert x \rvert$
`int max(int x, int y);`	maximum
`int min(int x, int y);`	minimum
`double hypot(double x, double y);`	$\sqrt{x^2 + y^2}$
`double sin(double x);`	$\sin x$, också cos, tan
`double asin(double x);`	$\arcsin x, \ldots$
`double atan2(double y, double x);`	$\arctan y/x$
`double exp(double x);`	e^x
`double log(double x);`	$\ln x$
`double sqrt(double x);`	$\sqrt{x}$
`double pow(double x, double y);`	x^y

Logiska uttryck

Logiska uttryck kan kopplas samman med operatorerna && ("och"), || ("eller"). ! betyder "icke".

```
int a = 3;
int b = 10;
if (a > 1 && b > 1) ...      // true
if (a < 0 || a > 10) ...     // false
if (! (a > 5)) ...           // true (a <= 5)
```

Sanningstabell för logiska operatorer:

p	q	p && q	p || q	! p
true	true	true	true	false
true	false	false	true	
false	true	false	true	true
false	false	false	false	

Logiska variabler har datatypen `boolean`.

Teckenuttryck

- En variabel av typ char har ett värde som består av *ett* tecken.
- Teckenkonstanter: 'A', '<', '7', ' '.
- Internt i datorn representeras varje tecken av ett nummer enligt Unicode-tabellen.
- Man behöver inte kunna tabellen utantill — man kan direkt jämföra tecken med varandra:

```
char ch = 'A';
if (ch < 'B') ...  // true
if (ch >= 'a') ... // false
```

- Ett teckenvärde konverteras automatiskt till heltal när det behövs. I andra riktningen måste man konvertera explicit:

```
char ch = ...;
if (ch >= 'a' && ch <= 'z') {
    ch = (char) (ch - 'a' + 'A');
}
```


Unicode

	002	003	004	005	006	007	00C	00D	00E	00F
0	sp	0	@	P	`	p	À		à	
1	!	1	A	Q	a	q	Á		á	
2	"	2	B	R	b	r				
3	#	3	C	S	c	s				
4	$	4	D	T	d	t	Ä		ä	
5	%	5	E	U	e	u	Å		å	
6	&	6	F	V	f	v	Æ	Ö	æ	ö
7	'	7	G	W	g	w				
8	(	8	H	X	h	x	È	Ø	è	ø
9	)	9	I	Y	i	y	É		é	
A	*	:	J	Z	j	z				
B	+	;	K	[	k	{				
C	,	<	L	\	l	|		Ü		ü
D	−	=	M	]	m	}				
E	.	>	N	^	n	~				
F	/	?	O	_	o					

Stränguttryck

En sträng är en följd av tecken. Strängar är objekt av standardklassen `String`, som vi behandlar senare. Strängkonstanter:

```
String s = "Rubrik";
System.out.println(s);

System.out.println("Detta är en lång text som inte " +
                   "ryms på en rad");
```

Objektuttryck

- new-uttryck, till exempel `new Square(20, 10, 40)`
- `null`
- funktionsanrop, till exempel `sq.getLocation()`
- `this`, som är en referens till det aktuella objektet
- `==` jämför referenser, *inte* innehållet i objekt:

```
public class Point {
    private int x;
    private int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

Point p1 = new Point(10, 20);
Point p2 = new Point(10, 20);
if (p1 != null) ... // true
if (p1 == p2) ...   // false
```


Villkorsuttryck, ? :

```
logiskt uttryck ? uttryck1 : uttryck2
```

Exempel:

```
int i = 1;
int j = 2;
int result = (i == 1) ? i + 5 : j + 5; // result = 6
return (i > j) ? i : j;                // 2 returneras
```

Man kan klara sig utan villkorsuttryck:

```
int result;
if (i == 1) {
    result = i + 5;
} else {
    result = j + 5;
}
```


Slumptal

Slumptal får man med hjälp av standardklassen `java.util.Random`:

```
/** En slumptalsgenerator med slumptalsfröet seed */
Random(long seed);

/** En slumptalsgenerator med ett slumpmässigt
    slumptalsfrö */
Random();

/** Slumpmässigt heltal i intervallet [0,n) */
int nextInt(int n);

/** Slumpmässigt reellt tal i intervallet [0,1.0) */
double nextDouble();
```


Användning av Random

10 slumpmässiga heltal i intervallet `[1, 6]`, 10 reella tal i intervallet `[5.0, 15.0)`:

```
import java.util.Random;

public class RandomExample {
    public static void main(String[] args) {
        Random rand = new Random();
        for (int i = 0; i < 10; i++) {
            int iRand = 1 + rand.nextInt(6);
            System.out.println(iRand);
        }
        for (int i = 0; i < 10; i++) {
            double dRand = 5 + 10 * rand.nextDouble();
            System.out.println(dRand);
        }
    }
}
```

Programexempel: Tärningsspel, 1

Ett tärningsspel med två spelare ska simuleras i ett program. Spelet går till på följande sätt: den förste spelaren kastar tärningen och räknar antalet kast tills det i två kast i följd blir samma antal prickar på tärningen. Därefter kastar den andre spelaren tärningen på samma sätt. Den av spelarna som gjort minst antal kast vinner spelet. Om båda spelarna gjort samma antal kast kastar de båda igen tills någon av dem vunnit. När spelet är klart ska namnet på vinnaren skrivas ut.

Klasser: `Die` (tärning), `Player` (spelare), `DiceGame` (genomför ett spel).

Klassen `Die`, en tärning

```
import java.util.Random;
public class Die {
    private static Random rand = new Random();
    private int pips;

    /** Skapar en tärning */
    public Die() {
        roll(); // så att pips får ett värde 1..6
    }

    /** Kastar tärningen */
    public void roll() {
        pips = 1 + rand.nextInt(6);
    }

    /** Tar reda på resultatet av det senaste kastet */
    public int getResult() {
        return pips;
    }
}
```


Specifikation av `Player`, en spelare

Båda spelarna ska spela med samma tärning, så de måste få reda på "utifrån" vilken tärning de ska använda:

Player

```
/** Skapar en spelare med namnet name */
Player(String name);

/** Spelaren kastar tärningen die tills det blir
    två lika i följd, returnerar antalet kast */
int play(Die die);

/** Tar reda på spelarens namn */
String getName();
```


Klassen `DiceGame`, en spelomgång, 1

```
public class DiceGame {
    private Player player1;
    private Player player2;
    private Die die;

    /** Skapar ett spel som spelas mellan spelarna
        player1 och player2 */
    public DiceGame(Player player1, Player player2) {
        this.player1 = player1;
        this.player2 = player2;
        die = new Die();
    }

    /** Genomför en spelomgång, returnerar vinnaren */
    public Player play() {
        ... nästa bild
    }
}
```

Klassen DiceGame, en spelomgång, 2

```
public class DiceGame {
    ...

    /** Genomför en spelomgång, returnerar vinnaren */
    public Player play() {
        int p1Rolls = player1.play(die);
        int p2Rolls = player2.play(die);
        while (p1Rolls == p2Rolls) {
            p1Rolls = player1.play(die);
            p2Rolls = player2.play(die);
        }
        return (p1Rolls < p2Rolls) ?
               player1 : player2;
    }
}
```


main-metod som genomför spelet

```
public class PlayGame {
    public static void main(String[] args) {
        Player p1 = new Player("Alice");
        Player p2 = new Player("Bertil");
        DiceGame game = new DiceGame(p1, p2);
        Player winner = game.play();
        System.out.println(winner.getName() +
                           " vann");
    }
}
```


Implementering av Player.play

```
public class Player {
    ...

    /** Spelaren kastar tärningen die tills det blir
        två lika i följd, returnerar antalet kast */
    public int play(Die die) {
        die.roll();
        int prevResult = die.getResult();
        die.roll();
        int result = die.getResult();
        int nbrRolls = 2;
        while (result != prevResult) {
            die.roll();
            nbrRolls = nbrRolls + 1;
            prevResult = result;
            result = die.getResult();
        }
        return nbrRolls;
    }
}
```


Kontrollstrukturer

Sekvens, där satserna utförs i tur och ordning. Att satser ska utföras i sekvens anger man genom att man skriver dem efter varandra.

Alternativ, där man testar ett villkor och beroende på resultatet går olika vägar genom programmet. Alternativ implementeras med `if`-satser eller `switch`-satser.

Repetition, där en sats utförs flera gånger. Repetition implementeras med `for`-, `while`- eller `do while`-satser.

Förkortade tilldelningssatser

Exempel:

```
x += dx;      // <=> x = x + dx
sum += term; // <=> sum = sum + term
nbr /= 10;    // <=> nbr = nbr / 10
```

Öka/minska med 1:

```
x++;          // <=> x = x + 1
x--;          // <=> x = x - 1
```


if-satser

Beräkna det största av värdena a och b (enklare med `Math.max(a,b)`):

```
int a = ...;
int b = ...;
int max;
if (a > b) {
    max = a;
} else {
    max = b;
}
System.out.println("Det största värdet är " + max);
```

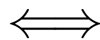
Tag absolutbeloppet av x (enklare med `Math.abs(x)`):

```
if (x < 0) {
    x = -x;
}
```


switch-satser

Testa mot ett antal *konstanta* värden:

```
if (x == 1) {
    S1
} else if (x == 3) {
    S2
} else if (x == 7) {
    S3
} else {
    S4
}
```



```
switch (x) {
    case 1:  S1; break;
    case 3:  S2; break;
    case 7:  S3; break;
    default: S4; break;
}
```


while-satser, for-satser

Rita kvadrat, minska sidlängden. Om sidlängden redan från början är ≤ 10 ritas inte någon gång.

```
while (sq.getSide() > 10) {
    sq.draw(w);
    sq.setSide(sq.getSide() - 5);
}
```

Rita kvadrat, flytta, n gånger. Om n är ≤ 0 ritas inte någon gång.

```
int n = ...;
for (int i = 0; i < n; i++) {
    sq.draw(w);
    sq.move(10, 10);
}
```

Negering av logiskt uttryck, De Morgans lagar

p och q är logiska uttryck, $\neg$ står för "icke", $\wedge$ för "och", $\vee$ för "eller":

$$\neg(p \wedge q) \iff (\neg p) \vee (\neg q)$$
$$\neg(p \vee q) \iff (\neg p) \wedge (\neg q)$$

I vanligt språk:

- Om uttrycket består av deluttryck sammanbundna med && eller ||, ändra alla && till || och omvänt.
- Negera alla ingående deluttryck. En relation negeras genom att man byter == mot !=, < mot >=, etc.

Exempel:

```
! (a < b || (a == 1 && b == 1))          <=>
! (a < b) && ! (a == 1 && b == 1)        <=>
! (a < b) && (! (a == 1) || ! (b == 1)) <=>
a >= b && (a != 1 || b != 1)
```


do while-satser

Satserna i repetitionen genomlöps minst en gång.

```
public static int mystery(int nbr) {
    int sum = 0;
    do {
        int digit = nbr % 10; // ger sista siffran i nbr
        if (digit == 0) {
            sum++;
        }
        nbr /= 10;
    } while (nbr > 0);
    return sum;
}
```


Inläsning från tangentbordet

```
import java.util.Scanner;

public class ScannerExample1 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int sum = 0;
        /* hasNextInt ger true om det finns ett tal */
        while (scan.hasNextInt()) {
            /* nextInt läser talet */
            sum += scan.nextInt();
        }
        System.out.println("Summa: " + sum);
    }
}
```


Inläsning från fil

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class ScannerExample2 {
    public static void main(String[] args) {
        Scanner scan = null;
        try {
            scan = new Scanner(new File("indata.txt"));
        } catch (FileNotFoundException e) {
            System.err.println("Filen indata.txt " +
                               "kunde inte öppnas");
            System.exit(1);
        }
        ... läs som tidigare
    }
}
```

Utskrift

Metoder i PrintStream (System.out) och PrintWriter:

```
void print(String s);   // skriv strängen s
void println(String s); // "print line", som print
                        // men avsluta med övergång
                        // till ny rad
void println();         // enbart ny rad
void flush();           // skicka upplagrade
                        // utskrifter till skärmen
```

Formatering av utskrift med printf:

```
for (int i = 2; i <= 5; i++) {
    double r = Math.sqrt(i);
    System.out.printf("%5d...%6.3f%n", i, r);
}
```


Utskrift på fil

```
import java.util.Random;
import java.io.PrintWriter;
import java.io.File;
import java.io.FileNotFoundException;

public class RandomExample2 {
    public static void main(String[] args) {
        PrintWriter out = null;
        try {
            out = new PrintWriter
                      (new File("random.txt"));
        } catch (FileNotFoundException e) {
            System.err.println("Filen random.txt " +
                                   "kunde inte öppnas");
            System.exit(1);
        }
        ... utskrifter med out.print hamnar på filen
    }
}
```


Algoritmexempel: Beräkning av minimum

Beräkna det minsta värdet i en följd av värden:

```
min = "stort värde";
för alla värden {
    value = "nästa värde";
    if (value < min) {
        min = value;
    }
}
```


Beräkning av minimum, exempel

Beräkna det minsta av 50 tal som läses från tangentbordet. Beräkna också ordningsnumret för det minsta talet.

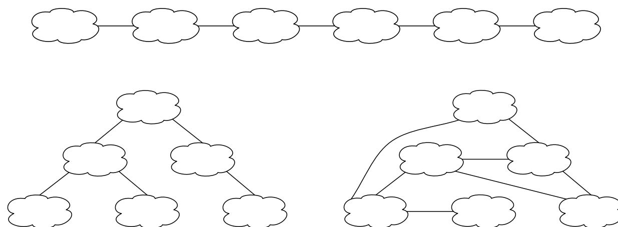
```
Scanner scan = new Scanner(System.in);
double min = Double.MAX_VALUE;
int minIndex = 0; // ordningsnumret för det minsta talet
for (int i = 0; i < 50; i++) {
    double nbr = scan.nextDouble();
    if (nbr < min) {
        min = nbr;
        minIndex = i;
    }
}
System.out.println("Det minsta talet är " + min +
                   " och det finns på plats " +
                   (minIndex + 1));
```

Datastrukturer

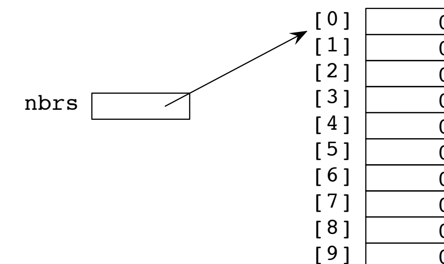
En datastruktur:

- kan innehålla många element,
- har *ett* namn,
- och man kan komma åt de enskilda elementen.

Man kan organisera elementen på olika sätt, till exempel som listor, träd eller grafer:



Vektorer




```
int[] nbrs = new int[10];
```

Fyll vektorn med Fibonacci-tal:

```
nbrs[0] = 1;
nbrs[1] = 1;
for (int i = 2; i < nbrs.length; i++) {
    nbrs[i] = nbrs[i - 1] + nbrs[i - 2];
}
```


Vektorer med objekt (referensvariabler)

```
Point[] points = new Point[10]; // 10 referensvariabler,
                                // alla null från början
Scanner scan = new Scanner(System.in);
for (int i = 0; i < points.length; i++) {
    int x = scan.nextInt();
    int y = scan.nextInt();
    points[i] = new Point(x, y);
}
```


Programexempel: Polygon

Klassen Polygon beskriver en polygon med ett givet (maximalt) antal hörnpunkter. Skapa och rita en triangel:

```
Polygon triangle = new Polygon(3);
triangle.addVertex(10, 10);
triangle.addVertex(50, 10);
triangle.addVertex(30, 40);
triangle.draw(w);
```

Implementering av Polygon, 1

```
public class Polygon {
    private Point[] vertices; // vektor med hörnpunkter
    private int n;            // antalet hörnpunkter

    /** Skapar en polygon som har plats för högst
        size hörnpunkter */
    public Polygon(int size) {
        vertices = new Point[size];
        n = 0;
    }

    /** Definierar en ny punkt med koordinaterna x,y */
    public void addVertex(int x, int y) {
        vertices[n] = new Point(x, y);
        n++;
    }
```


Implementering av Polygon, 2

```
    /** Flyttar polygonen avståndet dx i x-led, dy i y-led */
    public void move(int dx, int dy) {
        for (int i = 0; i < n; i++) {
            vertices[i].move(dx, dy);
        }
    }

    /** Ritar polygonen i fönstret w */
    public void draw(SimpleWindow w) {
        if (n == 0) { return; }
        Point start = vertices[0];
        w.moveTo(start.getX(), start.getY());
        for (int i = 1; i < n; i++) {
            w.lineTo(vertices[i].getX(),
                     vertices[i].getY());
        }
        w.lineTo(start.getX(), start.getY());
    }
}
```


Sätt in element "mitt i" vektorn

```
/** Lägger in en ny punkt med koordinaterna x,y
    på plats pos. Efterföljande element flyttas */
public void insertVertex(int pos, int x, int y) {
    for (int i = n; i > pos; i--) {
        vertices[i] = vertices[i - 1];
    }
    vertices[pos] = new Point(x, y);
    n++;
}
```


Ta bort element "mitt i" vektorn

```
/** Tar bort punkten på plats pos. Efterföljande
    element flyttas */
public void removeVertex(int pos) {
    for (int i = pos; i < n - 1; i++) {
        vertices[i] = vertices[i + 1];
    }
    vertices[n - 1] = null;
    n--;
}
```

”Utöka” en vektors storlek

När vektorn `vertices` blir full: 1) spara en referens till den gamla vektorn, 2) skapa en ny, dubbelt så stor, vektor, 3) kopiera över elementen från den gamla vektorn till den nya.

```
public void addVertex(int x, int y) {
    if (n == vertices.length) {
        Point[] oldVertices = vertices;
        vertices = new Point[2 * vertices.length];
        for (int i = 0; i < oldVertices.length; i++) {
            vertices[i] = oldVertices[i];
        }
    }
    vertices[n] = new Point(x, y);
    n++;
}
```


Matriser

$$mat = \begin{pmatrix} 7 & 9 & 123 & 41 & 1 \\ 22 & -18 & 12 & 3 & -2 \\ 11 & 16 & -4 & 0 & 6 \end{pmatrix}$$

Läs in matrisen (radvis):

```
Scanner scan = new Scanner(System.in);
int[][] mat = new int[3][5];
for (int i = 0; i < mat.length; i++) {
    for (int k = 0; k < mat[i].length; k++) {
        mat[i][k] = scan.nextInt();
    }
}
```


Algoritmexempel: Sökning

Sök upp platsen för ett givet element i en följd av element. Om det finns mer än ett element som är lika med det sökta så ska resultatet vara platsen för det första av dessa element.

Algoritm (linjärsökning):

```
pos = "platsen för det första elementet";
while ("fler element kvar" &&
       "elementet på plats pos inte är det vi söker") {
    pos = "platsen för nästa element";
}
```


Sökning, variant 1

```
public class Data {
    private int[] v;
    private int n;

    /* här finns konstruktorer och andra metoder */

    public int find(int nbr) {
        int i = 0;
        while (i < n && v[i] != nbr) {
            i++;
        }
        return (i < n) ? i : -1;
    }
}
```

Sökning, variant 2

```
public int find2(int nbr) {
    v[n] = nbr;
    int i = 0;
    while (v[i] != nbr) {
        i++;
    }
    return (i < n) ? i : -1;
}
```


Sökning, variant 3

```
public int find3(int nbr) {
    for (int i = 0; i < n; i++) {
        if (v[i] == nbr) {
            return i;
        }
    }
    return -1;
}
```


Binärsökning (bara i sorterad vektor)

```
public int binarySearch(int nbr) {
    int low = 0;      // undre gräns
    int high = n - 1; // övre gräns
    int mid = -1;     // mittpunkt
    boolean found = false;
    while (low <= high && ! found) {
        mid = (low + high) / 2;
        if (v[mid] == nbr) {
            found = true;
        } else if (v[mid] < nbr) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return found ? mid : -(low + 1);
}
```


Tidskomplexitet, sökning

Linjärsökning: $O(n)$
Binärsökning: $O(\log n)$

Vi har en vektor med 1000 element. Vi har mätt tiden för att söka upp ett element många gånger och funnit att det tar ungefär 1 μs både med linjärsökning och binärsökning. Hur lång tid tar det om vi har fler element i vektorn?

	1,000	10,000	100,000	1,000,000	10,000,000
linjär	1	10	100	1000	10000
binär	1	1.33	1.67	2.00	2.33

Algoritmexempel: Sortering

Sortera en följd av tal i växande ordning.

Algoritm (urvalssortering):

Sök upp det minsta talet och låt det byta plats med det första talet, sök upp det minsta av de återstående talen och låt det byta plats med det andra talet, osv.

Sortering

```
/** Sorterar talen i vektorn med urvalssortering */
public void sort() {
    for (int i = 0; i < n - 1; i++) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;
        for (int k = i; k < n; k++) {
            if (v[k] < min) {
                min = v[k];
                minIndex = k;
            }
        }
        v[minIndex] = v[i]; // låt v[i] och
        v[i] = min;         // v[minIndex] byta plats
    }
}
```


Tidskomplexitet, sortering

Urvalssortering: $O(n^2)$
"Bra" metoder: $O(n \log n)$

Vi har en vektor med 1000 element. Vi har mätt tiden för att sortera elementen många gånger och funnit att det tar ungefär 1 ms både med urvalssortering (eller någon annan "dålig" metod) och en "bra" metod. Hur lång tid tar det om vi har fler element i vektorn?

	1,000	10,000	100,000	1,000,000	10,000,000
dålig	1	100	10^4	10^6	10^8
bra	1	13.3	167	2000	23000

Algoritmexempel: Registrering

```
public class Test {
    private Student[] students; // studenterna
    private int n;              // antalet studenter

    /** Skapar ett prov med plats för max studenter */
    public Test(int max) {
        students = new Student[max];
        n = 0;
    }

    /** Lägger till studenten s */
    public add(Student s) {
        students[n] = s;
        n++;
    }

    /** Skriver ut antalet studenter som har 0,1,...,
        50 poäng på provet */
    public void printStatistics() { ... }
}
```

Olika poängintervall

0, 1, 2, . . . , 50 poäng:

```
public void printStatistics() {
    int[] count = new int[51];
    for (int i = 0; i < n; i++) {
        int index = students[i].getPoints();
        count[index]++;
    }
    // ... skriv ut antalen
}
```

0–9, 10–19, 20–29, 30–39, 40–50 poäng:

```
public void printStatistics() {
    int[] count = new int[5];
    for (int i = 0; i < n; i++) {
        int index = students[i].getPoints() / 10;
        if (index == 5) { // om 50 poäng
            index = 4;
        }
        count[index]++;
    }
    // ... skriv ut antalen
}
```


Oregelbundna intervall, dålig lösning

0–24 poäng ger betyg U, 25–34 poäng betyg 3, 35–42 poäng betyg 4, 43–50 poäng betyg 5. Antalet U-betyg registreras i `count[0]`, antalet 3-betyg i `count[1]`, osv.

```
int points = students[i].getPoints();
int index;
if (points < 25) {
    index = 0;
} else if (points < 35) {
    index = 1;
} else if (points < 43) {
    index = 2;
} else {
    index = 3;
}
count[index]++;
```


Oregelbundna intervall, bättre lösning

```
int[] limits = { 25, 35, 43, 51 };
...
int points = students[i].getPoints();
int index = 0;
while (points >= limits[index]) {
    index++;
}
count[index]++;
```


ArrayList

En `ArrayList`:

- är en standardklass (i paketet `java.util`),
- innehåller alltid objekt (inte `int`, `double`, . . .),
- är en *generisk* klass som kan innehålla objekt av godtycklig typ,
- lagrar sina element i en vektor,
- utökar vektorns storlek vid behov,
- har metoder för att sätta in och ta bort element (bland annat).

Viktiga operationer på ArrayList

```
ArrayList<E>();            // skapar en tom ArrayList
                           // för element av typen E

int size();                // antalet element
E get(int pos);            // elementet på plats pos

void add(E obj);           // lägger in obj sist
                           // (efter de element som
                           // finns i listan)
void add(int pos, E obj);  // lägger in obj på plats
                           // pos. Efterföljande
                           // element flyttas
E remove(int pos);         // tar bort elementet på
                           // plats pos, returnerar
                           // det borttagna elementet
```


Ny implementering av Polygon

Samma (dynamiska) polygonklass som i kapitel 8, men punkterna lagras i en ArrayList:

```
import java.util.ArrayList;

public class Polygon {
    private ArrayList<Point> vertices;

    /** Skapar en polygon */
    public Polygon() {
        vertices = new ArrayList<Point>();
    }

    /** Definierar en ny punkt med koordinaterna x,y */
    public void addVertex(int x, int y) {
        vertices.add(new Point(x, y));
    }

    ...
```


Polygon, 2

```
    /** Flyttar polygonen avståndet dx i x-led,
        dy i y-led */
    public void move(int dx, int dy) {
        for (int i = 0; i < vertices.size(); i++) {
            vertices.get(i).move(dx, dy);
        }
    }

    /** Ritar polygonen i fönstret w */
    public void draw(SimpleWindow w) {
        Point start = vertices.get(0);
        w.moveTo(start.getX(), start.getY());
        for (int i = 1; i < vertices.size(); i++) {
            w.lineTo(vertices.get(i).getX(),
                     vertices.get(i).getY());
        }
        w.lineTo(start.getX(), start.getY());
    }
```


Polygon, 3

```
    /** Lägger in en ny punkt med koordinaterna x,y
        på plats pos. Efterföljande element flyttas */
    public void insertVertex(int pos, int x, int y) {
        vertices.add(pos, new Point(x, y));
    }

    /** Tar bort punkten på plats pos. Efterföljande
        element flyttas */
    public void removeVertex(int pos) {
        vertices.remove(pos);
    }
}
```

Enkla värden i listor

Om man vill lagra "enkla värden" i en ArrayList måste de kapslas in i objekt, till exempel int-värden i objekt av standardklassen Integer:

```
public class Integer {
    private int value;

    /** Skapar ett nytt Integer-objekt med värdet
        value */
    public Integer(int value) {
        this.value = value;
    }

    /** Tar reda på värdet */
    public int intValue() {
        return value;
    }

    ...
}
```


Lista med heltal

```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<Integer> list =
            new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            Integer obj = new Integer(nbr);
            list.add(obj);
        }
        for (int i = list.size() - 1; i >= 0; i--) {
            Integer obj = list.get(i);
            int nbr = obj.intValue();
            System.out.println(nbr);
        }
    }
}
```


Algoritmer med ArrayList, 1

```
import java.util.ArrayList;

public class IntSequence {
    private ArrayList<Integer> list;

    /** Skapar en tom lista */
    public IntSequence() {
        list = new ArrayList<Integer>();
    }

    /** Lägger in talet nbr sist i listan */
    public void insert(int nbr) {
        list.add(new Integer(nbr));
    }

    ...
```


Algoritmer med ArrayList, 2

```
    /** Beräknar summan av elementens värden */
    public int getSum() {
        int sum = 0;
        for (int i = 0; i < list.size(); i++) {
            Integer elem = list.get(i);
            sum += elem.intValue();
        }
        return sum;
    }
```

Algoritmer med ArrayList, 3

```
/** Beräknar det största av elementens värden,
    Integer.MIN_VALUE om talföljden är tom */
public int getMax() {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < list.size(); i++) {
        Integer elem = list.get(i);
        int nbr = elem.intValue();
        if (nbr > max) {
            max = nbr;
        }
    }
    return max;
}
```


Algoritmer med ArrayList, 4

```
    /** Undersöker om talföljden innehåller någon nolla */
    public boolean containsZero() {
        int i = 0;
        while (i < list.size() &&
                list.get(i).intValue() != 0) {
            i++;
        }
        return i < list.size();
    }
}
```


Mängdlära

En mängd är en samling av element, där varje element förekommer högst en gång.

Beteckning	$\{1, 7, 5, 3\}$
Alternativ	$\{x : x \text{ udda heltal}, 0 \leq x \leq 8\}$
Lägg till	$\{1, 7, 5, 3\} + 8 = \{1, 7, 5, 3, 8\}$,
	$\{1, 7, 5, 3\} + 7 = \{1, 7, 5, 3\}$
Kardinalitet	$\lvert\{1, 7, 5, 3\}\rvert = 4$
Union	$\{1, 7\} \cup \{7, 3\} = \{1, 7, 3\}$
Snitt	$\{1, 7\} \cap \{7, 3\} = \{7\}$

Programexempel, mängder

IntSet

```
/** Skapar en tom mängd */
IntSet();

/** Lägger in talet nbr i mängden */
void insert(int nbr);

/** Tar bort talet nbr ur mängden */
void remove(int nbr);

/** Undersöker om talet nbr finns i mängden */
boolean contains(int nbr);

/** Tar reda på antalet element i mängden */
int size();
```

Användning av IntSet

Räkna antalet *olika* heltal som man läst in:

```
import java.util.Scanner;

public class SetExample {
    public static void main(String[] args) {
        IntSet numbers = new IntSet();
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            numbers.insert(nbr);
        }
        System.out.println("Antal olika tal: " +
                           numbers.size());
    }
}
```


Implementering av IntSet

Talen i mängden lagras i en *sorterad* ArrayList:

```
import java.util.ArrayList;

public class IntSet {
    private ArrayList<Integer> elems;

    /** Skapar en tom mängd */
    public IntSet() {
        elems = new ArrayList<Integer>();
    }
```


Leta upp element

```
/** Söker upp nbr i listan med binärsökning. Returnerar index
    om talet finns, -(index+1) annars, där index är platsen
    där talet ska stoppas in enligt sorteringsordningen */
private int binarySearch(int nbr) {
    int low = 0;
    int high = elems.size() - 1;
    int mid = -1;
    boolean found = false;
    while (low <= high && ! found) {
        mid = (low + high) / 2;
        if (elems.get(mid).intValue() == nbr) {
            found = true;
        } else if (elems.get(mid).intValue() < nbr) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return found ? mid : -(low + 1);
}
```


insert och remove

```
/** Lägger in talet nbr i mängden */
public void insert(int nbr) {
    int pos = binarySearch(nbr);
    if (pos < 0) {
        elems.add(-(pos + 1), new Integer(nbr));
    }
}

/** Tar bort talet nbr ur mängden */
public void remove(int nbr) {
    int pos = binarySearch(nbr);
    if (pos >= 0) {
        elems.remove(pos);
    }
}
```

contains och size

```
/** Undersöker om talet nbr finns i mängden */
public boolean contains(int nbr) {
    int pos = binarySearch(nbr);
    return pos >= 0;
}

/** Tar reda på antalet element i mängden */
public int size() {
    return elems.size();
}
```


Fler ArrayList-operationer

```
boolean isEmpty();     // true om listan är tom

int indexOf(E obj);    // söker upp obj. Returnerar
                       // index för elementet, -1
                       // om elementet inte fanns

E set(int pos, E obj); // ersätter elementet på plats
                       // pos med obj, returnerar
                       // elementet som fanns på
                       // platsen
void clear();          // tar bort alla element i
                       // listan
```


equals

I `indexOf` utnyttjas operationen equals på objekten för att undersöka om man hittat elementet. I klassen `Integer` är operationen implementerad så här:

```
public class Integer {
    private int value;

    ...

    /** Undersöker om värdet i detta objekt är detsamma
        som värdet i objektet obj */
    public boolean equals(Object obj) {
        if (! (obj instanceof Integer)) {
            return false;
        }
        return value == ((Integer) obj).value;
    }
}
```


equals i Point

```
public class Point {
    private int x;
    private int y;

    // ... konstruktorer och metoder

    public boolean equals(Object obj) {
        if (! (obj instanceof Point)) {
            return false; // obj är inte ett Point-objekt
        }
        Point p = (Point) obj;
        return x == p.x && y == p.y;
    }
}
```

Arv

Fyra klasser som beskriver olika slags fordon:

```
public class Car {
    private String licenseNbr;
    private Person owner;
    private int maxPassengers;
}

public class Bus {
    private String licenseNbr;
    private Person owner;
    private int maxPassengers;
}

public class Truck {
    private String licenseNbr;
    private Person owner;
    private int maxLoad;
}

public class MotorCycle {
    private String licenseNbr;
    private Person owner;
}
```

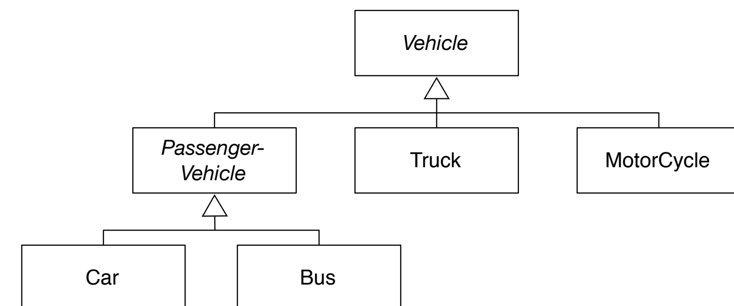

Samma modell, med arv ("extends")

```
public class Vehicle {
    private String licenseNbr;
    private Person owner;
}

public class Car extends Vehicle {
    private int maxPassengers;
}

public class Truck extends Vehicle {
    private int maxLoad;
}

public class Bus extends Vehicle {
    private int maxPassengers;
}

public class MotorCycle extends Vehicle {
}
```


Ännu mera arv

```
public class PassengerVehicle extends Vehicle {
    private int maxPassengers;
}

public class Car extends PassengerVehicle {
}

public class Bus extends PassengerVehicle {
}
```


Arvshierarki i UML ("klassträd")



`Vehicle` och `PassengerVehicle` är *abstrakta* klasser.

Arv och referenser, typer

```
Vehicle[] queue = new Vehicle[10];
```

En variabel med typen `Vehicle` får referera till objekt av alla subklasser till `Vehicle`. Så i kön `queue` kan man ställa fordon av olika slag (bilar, bussar, ...).

Skydd i samband med arv

```
public class A {
    private int x;
    protected int y;
    public int z;
}

public class B extends A {
    // här är de ärvda attributen y och z tillgängliga,
    // x är inte tillgängligt
}
```


Konstruktorer och arv

Konstruktorn i superklassen `Vehicle` ser ut som vanligt:

```
public abstract class Vehicle {
    private String licenseNbr;
    private Person owner;

    /** Initierar ett Vehicle-objekt med registrerings-
        numret licenseNbr och ägaren owner */
    protected Vehicle(String licenseNbr, Person owner) {
        this.licenseNbr = licenseNbr;
        this.owner = owner;
    }
}
```


Konstruktorer i subklasser

Konstruktorn i subklassen `Truck` måste först anropa superklassens konstruktor med `super(...)`:

```
public class Truck extends Vehicle {
    private int maxLoad;

    /** Skapar ett Truck-objekt med registreringsnumret
        licenseNbr, ägaren owner och maxlasten maxLoad */
    public Truck(String licenseNbr, Person owner,
                 int maxLoad) {
        super(licenseNbr, owner);
        this.maxLoad = maxLoad;
    }
}
```

När ska man använda arv?

Vi ska implementera ett grafiksystem med klasserna Point, Square och Circle. Punkter har ett läge och kan flyttas.

Man ska *inte* låta Square och Circle ärva läget och flyttningsoperationen från Point! Subklasserna ska vara specialiseringar av superklassen, och kvadrater och cirklar är inte ett speciellt slags punkter.

I stället använder vi *sammansättning*, som tidigare:

```
public class Square {
    private Point location;
    private int side;
    ...
}

public class Circle {
    private Point location;
    private int radius;
    ...
}
```


Figurer med arv — superklass

```
public abstract class Shape {
    private Point location;

    protected Shape(int x, int y) {
        location = new Point(x, y);
    }

    public void move(int dx, int dy) {
        location.move(dx, dy);
    }
}
```

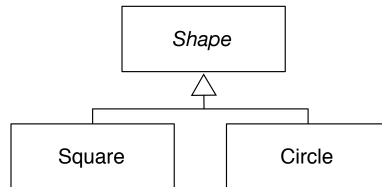

Figurer, subklasser

```
public class Square extends Shape {
    private int side;

    public Square(int x, int y, int side) {
        super(x, y);
        this.side = side;
    }
}

public class Circle extends Shape {
    private int radius;

    public Circle(int x, int y, int radius) {
        super(x, y);
        this.radius = radius;
    }
}
```


”Är-en” och ”har-en”

- En kvadrat ”har-ett” läge som beskrivs av en punkt, men är ingen punkt. Alltså: sammansättning.
- En kvadrat ”är-en” geometrisk figur, men har ingen figur. Alltså: arv.
- En lastbil ”är-ett” fordon.
  En lastbil ”har-en” motor.
- En hund ”är-ett” däggdjur.
  En hund ”har-en” ägare (en människa).
  En människa ”är-ett” däggdjur.
  En människa ”har-en” hund.

Typregler för arv



En referensvariabel som deklarerats med typen `C` får referera till objekt av klassen `C` och dessutom till objekt av alla subklasser till `C`.

Typregel, exempel

```
Shape aShape;
Square aSquare;
```

Garanterat korrekt:

```
aSquare = new Square(...); // samma typ
aShape = new Square(...);  // Square är subklass
                           // till Shape
aShape = aSquare;          // också korrekt, av
                           // samma orsak
```

Säkert fel:

```
aSquare = new Circle(...); // inte samma typ och
                           // Circle är inte subklass
                           // till Square
aSquare = aShape;          // Shape är inte subklass
                           // till Square
```


Explicit konvertering

Inte tillåtet:

```
aShape = new Square(...);
aSquare = aShape;
```

Men vi är ju här säkra på att `aShape` refererar till ett `Square`-objekt. Då kan vi explicit konvertera referensen:

```
aSquare = (Square) aShape;
```

Javasystemet kontrollerar *under exekvering* att typerna stämmer (om de inte gör det får man ett `ClassCastException`).

Olika implementeringar av samma metod

Rita figurer: en `draw`-metod i `Square` (ritar fyra linjer), en annan i `Circle` (ritar många linjer).

```
Shape[] shapes = new Shape[100];
... // 100 figurer av olika slag läggs in i vektorn
for (int i = 0; i < shapes.length; i++) {
    shapes[i].draw(w);
}
```

När kompilatorn letar efter `draw`-metoden tittar den bara i variabelns klass, alltså i `Shape`. Man måste alltså skriva en `draw`-metod där.

Dynamisk bindning, polymorfism

Man kan skriva en tom draw-metod i Shape:

```
public abstract class Shape {
    ...
    public void draw(SimpleWindow w) {}
}
```

Det fungerar eftersom Java använder *dynamisk bindning* av metodanrop. Det innebär att det är *objektets* typ som under exekvering avgör vilken metod som ska anropas. När satsen shapes[i].draw(w) ska utföras så tittar Javasystemet alltså på objektet shapes[i]: om det är ett Square-objekt så anropas draw i Square, om det är ett Circle-objekt så anropas draw i Circle.

Abstrakta metoder

Det är bättre att specificera draw-metoden som *abstrakt* i Shape. Man talar då om att implementeringarna av operationen finns i subklasserna:

```
public abstract class Shape {
    /** Ritar figuren i fönstret w */
    public abstract void draw(SimpleWindow w);
}
```


Sammanfattning

- "Att kunna flyttas" är en egenskap hos alla figurer. Alla figurer flyttas på samma sätt och därför placerar man move-operationen i klassen Shape. Operationen ärvs av subklasserna Square och Circle.
- "Att kunna ritas upp" är också en egenskap hos alla figurer. Eftersom olika figurer ritas på olika sätt specificerar man draw-operationen som abstrakt i klassen Shape. En implementering av operationen finns i varje subklass till Shape.
- "Att ha en sidlängd" och "att ha en radie" är egenskaper som är specifika för klassen Square respektive Circle. Implementeringarna av de metoder som hanterar motsvarande attribut är också specifika för respektive subklass.

Programexempel: Grupper av figurer

Skriv ett ritprogram där man kan rita (och flytta, ta bort, ...) figurer. Man ska också kunna gruppera figurerna (members borde varit en ArrayList):

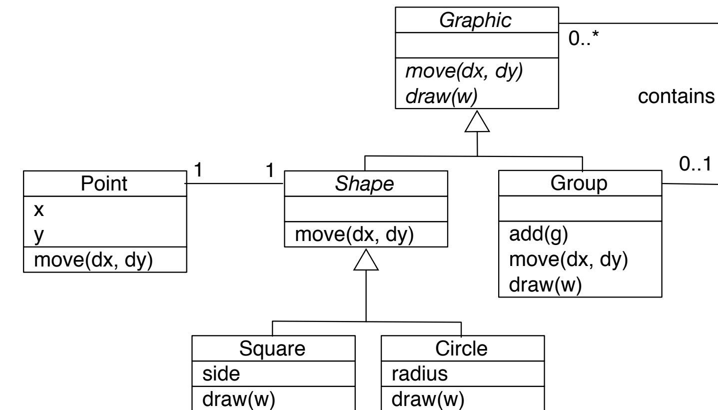
```
public class Group {
    private Shape[] members; // figurerna
    private int n;           // antalet figurer

    /** Skapar en tom grupp */
    public Group() {
        members = new Shape[100];
        n = 0;
    }

    /** Lägger in figuren s i gruppen */
    public void add(Shape s) {
        members[n] = s;
        n++;
    }
```

Grupper, forts

```
    /** Ritar alla figurer i gruppen i fönstret w */
    public void draw(SimpleWindow w) {
        for (int i = 0; i < n; i++) {
            members[i].draw(w);
        }
    }

    /** Flyttar alla figurer i gruppen avståndet dx
        i x-led, dy i y-led */
    public void move(int dx, int dy) {
        for (int i = 0; i < n; i++) {
            members[i].move(dx, dy);
        }
    }
}
```


Bättre gruppering

Inför en ny superklass `Graphic`. Samma `Group` som tidigare, men `members` och parametern till `add` har nu typen `Graphic`.



this och super

`this`: referens till det aktuella objektet.
`super`: referens till det aktuella objektet men har superklassens typ.

```
public abstract class Shape {
    protected Point location;
    protected void print() {
        System.out.print("Läge: " + location.getX() +
                         "," + location.getY());
    }
}

public class Square extends Shape {
    private int side;
    public void print() {
        System.out.print("Detta är en kvadrat. ");
        super.print(); // skriv ut läget
        System.out.print(", sidlängd: " + side);
    }
}
```


Rekursiva metoder

Definition av fakultet:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

```
public static int factorial(int n) {
    if (n <= 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Ni kommer att läsa mera om rekursiva metoder i fortsättningskurserna ...

instanceof

Räkna figurer av olika slag:

```
Shape[] shapes = new Shape[100];
...
int nbrSquares = 0;
int nbrCircles = 0;
for (int i = 0; i < shapes.length; i++) {
    Shape aShape = shapes[i];
    if (aShape instanceof Square) {
        nbrSquares++;
    } else if (aShape instanceof Circle) {
        nbrCircles++;
    }
}
System.out.println("Antalet kvadrater: " + nbrSquares);
System.out.println("Antalet cirklar:   " + nbrCircles);
```


String och StringBuilder

Standardklasser (i paketet `java.lang`, behöver inte importeras):

`String`	Beskriver en följd av tecken. Tecknen kan läsas av men inte ändras. Med operatorn + konkatenerar man (slår ihop) två strängar, eller en sträng med ett talvärde. Då bildas ett nytt strängobjekt.
`StringBuilder`	En följd av tecken som kan läsas av *och* ändras.

```
String s1 = "En text";
String s2 = "en text till";
String result = s1 + " och " + s2;

int x = 10;
int y = 30;
String s1 = "Summan är " + x + y;
String s2 = "Summan är " + (x + y);
```


Viktiga operationer på String

```
String();                      // skapar en tom sträng
                               // (kan också skrivas "")
int length();                  // antalet tecken
char charAt(int pos);          // tecknet på plats pos
boolean equals(Object s);      // true om innehållet i
                               // aktuell sträng är lika
                               // med innehållet i s
int compareTo(String s);       // jämför aktuell sträng med s
int indexOf(char ch);          // index för den första
                               // förekomsten av ch, -1
                               // om ch inte finns
String substring(int start,    // ny sträng med tecknen
                 int end);     // med index [start, end)
```

```
"Java".compareTo("Java") == 0
"java".compareTo("javac") < 0
"java".compareTo("Java") > 0
"java".compareTo("jazz") < 0
```


Användning av String, 1

```
public class Text {
    private String s;

    public Text(String s) {
        this.s = s;
    }

    /** Tar reda på antalet blanktecken i strängen */
    public int getNbrSpaces() {
        int spaces = 0;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ' ') {
                spaces++;
            }
        }
        return spaces;
    }
}
```

Användning av String, 2

```
/** Tar reda på index för den första förekomsten av tecknet
    ch i texten, -1 om inget sådant tecken finns */
public int indexOf(char ch) {
    int i = 0;
    while (i < s.length() && s.charAt(i) != ch) {
        i++;
    }
    return (i < s.length()) ? i : -1;
}
```


Användning av String, 3

```
/** Tar reda på det första ordet i texten. Ett ord är en
    följd av tecken som inte är blanka */
public String firstWord() {
    int start = 0;
    while (start < s.length() &&
           Character.isWhitespace(s.charAt(start))) {
        start++;
    }
    int end = start;
    while (end < s.length() &&
           ! Character.isWhitespace(s.charAt(end))) {
        end++;
    }
    return s.substring(start, end);
}
```


StringBuilder

Skapa, ta reda på längd och tecken (som String):

```
StringBuilder();          // tom strängbuffert
StringBuilder(String s); // kopia av s
int length();             // antalet tecken
char charAt(int pos);     // tecknet på plats pos
```

Skapa String-objekt med samma innehåll:

```
String toString(); // skapar ett String-objekt med
                   // samma innehåll som denna
                   // strängbuffert
```


Ändra innehållet i StringBuilder-objekt

```
void setCharAt(int k, char ch); // ändrar tecknet på
                                // plats k till ch
StringBuilder append(String s); // lägger till s
                                // efter tecknen
StringBuilder insert(int k, String s); // lägger in s
                                // på plats k, flyttar
                                // tecknen efter
StringBuilder deleteCharAt(int k); // tar bort tecknet
                                // på plats k
StringBuilder delete(int start, int end); // tar bort
                                // tecknen [start,end)
StringBuilder replace(int start, int end, String s);
                                // ersätter tecknen
                                // [start,end) med s
```

Användning av StringBuilder, 1

```
public class MutableText {
    private StringBuilder sb;

    public MutableText(String s) {
        sb = new StringBuilder(s);
    }

    /** Ändrar alla små bokstäver a-z i texten till motsvarande
        stora bokstäver */
    public void changeToUpperCase() {
        for (int i = 0; i < sb.length(); i++) {
            char ch = sb.charAt(i);
            if (ch >= 'a' && ch <= 'z') {
                sb.setCharAt(i, (char) (ch - 'a' + 'A'));
            }
        }
    }
```


Användning av StringBuilder, 2

```
/** Lägger in ett blanktecken efter varje punkt och kommatecken
    i texten, dock ej efter textens sista tecken */
public void insertSpaces() {
    int i = 0;
    while (i < sb.length() - 1) {
        if (sb.charAt(i) == '.' || sb.charAt(i) == ',') {
            sb.insert(i + 1, ' ');
            i++;
        }
        i++;
    }
}
```


Ta bort blanktecken från sträng

```
public static String removeSpacesFrom(String s) { // "DÅLIGT"
    String result = "";
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) != ' ') {
            result += s.charAt(i);
        }
    }
    return result;
}

public static String removeSpacesFrom(String s) { // "BRA"
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) != ' ') {
            sb.append(s.charAt(i));
        }
    }
    return sb.toString();
}
```


Formatering

Automatisk formatering vid utskrift:

```
int sum = 209;
System.out.println("Summan är " + sum);
```

Formatering utan utskrift:

```
int sum = 209;
String s1 = "" + sum;              // "209"
String s2 = String.valueOf(sum); // "209"
```

toString

```
public class Complex {
    private double re; // realdel
    private double im; // imaginärdel

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public String toString() {
        return "(" + re + ", " + im + ")";
    }
}
```

Exempel på användning:

```
Complex c = new Complex(1.5, 2.3);
System.out.println("c = " + c.toString());
System.out.println("c = " + c);
```


Programexempel: Datakomprimering

Följdlängdskodning:

- Man räknar hur många gånger som ett tecken förekommer i följd.
- Om antalet är större än 3 lagras först ett dollartecken, sedan antalet tecken, sedan tecknet. Dollartecknet fungerar som ett "escapetecken" som talar om att de följande tecknen ska tolkas på ett speciellt sätt. Antalet ska lagras i ett tecken och måste rymmas i en `char`-variabel (16 bitar).
- Om antalet är mindre än eller lika med 3 lagras alla tecken.
- Exempel: *aabbbbbcdddeeeeeeffff* kodas som *aa$5bcddd$6e$4f*. Siffrorna är inte tecknen `'5'`, `'6'` och `'4'` utan tecknen med Unicode-numren 5, 6 respektive 4.
- Vi förutsätter att det inte finns några dollartecken i texten som ska konverteras.

Komprimering (dålig formatering, platsbrist)

```
public static String compress(String s) {
    StringBuilder sb = new StringBuilder();
    int i = 0;
    while (i < s.length()) {
        char ch = s.charAt(i);
        int nbrEqual = 1;
        i++;
        while (i < s.length() && s.charAt(i) == ch) {
            i++; nbrEqual++;
        }
        if (nbrEqual > 3) {
            sb.append('$'); sb.append((char) nbrEqual);
            sb.append(ch);
        } else {
            for (int k = 0; k < nbrEqual; k++) { sb.append(ch); }
        }
    }
    return sb.toString();
}
```


Dekomprimering

```
public static String decompress(String s) {
    StringBuilder sb = new StringBuilder();
    int i = 0;
    while (i < s.length()) {
        char ch = s.charAt(i);
        if (ch != '$') {
            sb.append(ch);
        } else {
            i++;
            int nbrEqual = s.charAt(i);
            i++;
            ch = s.charAt(i);
            for (int k = 0; k < nbrEqual; k++) {
                sb.append(ch);
            }
        }
        i++;
    }
    return sb.toString();
}
```

Objektorienterad programutveckling

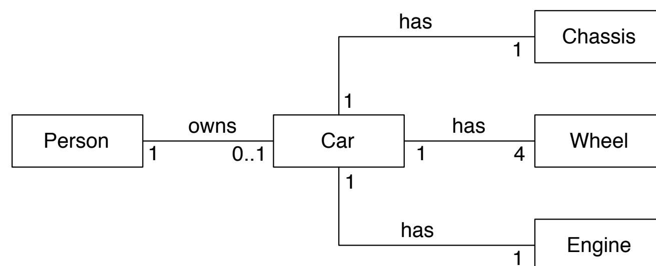
Vattenfallsmetod	Analys, sedan design, sedan implementering, sedan testning, sedan klart.
Iterativ metod	Analys–design–implementering–testning– mera analys–design–implementering–testning– mera analys–design–implementering–testning– mera analys–design–implementering–testning– . . .

Moment i en analys- och designmetodik

1. Finn klasser.
2. Finn samband mellan objekt och mellan klasser.
3. Finn operationer.
4. Finn attribut.

Krav?

Notation, UML



Designexempel: Myntvändning

I en hög ligger N mynt på varandra, alla med klave uppåt. Man vänder det översta myntet och lägger tillbaka det överst i högen. Sedan tar man de två översta mynten i högen, vänder på dem och lägger tillbaka dem. Därefter tar man de tre översta mynten, vänder på dem och lägger tillbaka dem. Detta upprepas med 4, 5, . . . , N mynt. Sedan startar man på nytt med 1, 2, 3, . . . mynt och vänder dessa. Man fortsätter tills alla mynten åter har klave uppåt.

Skriv ett program som beräknar hur många vändningar som erfordras för en hög med 10 mynt.

T	F	F	F	T	T	F	T	F	T
T	T	T	F	F	F	T	T	F	T
T	T	T	T	T	T	F	F	F	T
Efter vändning:	1	2	3	4	5	6	7	8	9

Kandidater till klasser (substantiv)

I en hög ligger N mynt på varandra, alla med klave uppåt. Man vänder det översta myntet och lägger tillbaka det överst i högen. Sedan tar man de två översta mynten i högen, vänder på dem och lägger tillbaka dem. Därefter tar man de tre översta mynten, vänder på dem och lägger tillbaka dem. Detta upprepas med 4, 5, ..., N mynt. Sedan startar man på nytt med 1, 2, 3, ... mynt och vänder dessa. Man fortsätter tills alla mynten åter har klave uppåt.

Skriv ett program som beräknar hur många vändningar som erfordras för en hög med 10 mynt.

Förkasta och behåll kandidater

Kandidater: hög (av mynt), mynt, klave, man, program, vändning.

klave	Egenskap hos ett mynt, attribut i myntklassen.
man	Obestämt, någon som ser till att rätt antal mynt vänds, och det behövs en klass med detta ansvarsområde.
program	Ingår inte i systemet.
vändning	Substantiverat verb, motsvarar en operation.

Scenario, operationer

1. Skapa högen med 10 mynt. Alla mynten ska ha klave uppåt.
2. Vänd 1 mynt i högen.
3. Avsluta om alla mynt i högen har klave uppåt.
4. Vänd 2 mynt i högen.
5. Avsluta om alla mynt i högen har klave uppåt.
6. ... osv tills 10 mynt har vänts.
7. Vänd 1 mynt i högen.
8. ... osv tills alla mynt i högen har klave uppåt.

Ur detta scenario får vi direkt tre operationer på klassen `Pile`:

- Skapa en hög (konstruktor).
- Vänd *m* mynt i högen.
- Undersök om alla mynten i högen har klave uppåt.

Specifikation, `Pile`

`Pile`

```
/** Skapar en hög med n mynt */
Pile(int n);

/** Vänder de m översta mynten i högen */
void flipPart(int m);

/** Tar reda på om alla mynten har klave uppåt */
boolean allTailsUp();
```

Operationer och specifikation, Coin

`flipPart`	Flytta om mynten i högen, vänd varje enskilt mynt.
`allTailsUp`	Undersök om varje enskilt mynt har klave uppåt.

Coin

```
/** Skapar ett mynt med klave uppåt */
Coin();

/** Vänder myntet */
void flip();

/** Tar reda på om myntet har klave uppåt */
boolean hasTailUp();
```


CoinTurner

CoinTurner, specifikation

```
/** Skapar en myntvändare som vänder mynten
    i högen pile */
CoinTurner(Pile pile);

/** Vänder mynten, returnerar antalet
    vändningar */
int turnCoins();
```


CoinTurner, implementering (obs getNbrOfCoins)

```
public class CoinTurner {
    private Pile pile;
    ...
    /** Vänder mynten, returnerar antalet
        vändningar */
    public int turnCoins() {
        int nbrOfFlips = 0; // antal vändningar
        int m = 1; // antal mynt att vända
        do {
            pile.flipPart(m);
            nbrOfFlips++;
            m++;
            if (m > pile.getNbrOfCoins()) {
                m = 1;
            }
        } while (! pile.allTailsUp());
        return nbrOfFlips;
    }
}
```


main-metoden

```
public static void main(String[] args) {
    Pile pile = new Pile(10);
    CoinTurner turner = new CoinTurner(pile);
    int turns = turner.turnCoins();
    System.out.println("Antal vändningar: " +
                       turns);
}
```

Pile, implementering 1

```
public class Pile {
    private Coin[] coins;

    /** Skapar en hög med n mynt */
    public Pile(int n) {
        coins = new Coin[n];
        for (int i = 0; i < n; i++) { coins[i] = new Coin(); }
    }

    /** Tar reda på om alla mynten har klave uppåt */
    public boolean allTailsUp() {
        int i = 0;
        while (i < coins.length &&
               coins[i].hasTailUp()) {
            i++;
        }
        return i >= coins.length;
    }
}
```


Pile, implementering 2

```
/** Vänder de m översta mynten i högen */
public void flipPart(int m) {
    // low och high är index för det understa
    // och det översta myntet som vänds
    int low = coins.length - m;
    int high = coins.length - 1;
    while (low < high) {
        Coin temp = coins[low];
        coins[low] = coins[high];
        coins[high] = temp;
        coins[low].flip();
        coins[high].flip();
        low++;
        high--;
    }
    if (low == high) {
        coins[low].flip();
    }
}
```


Coin är enkel

```
public class Coin {
    private boolean tailUp; // true om myntet har
                            // klave uppåt

    /** Skapar ett mynt med klave uppåt */
    public Coin() {
        tailUp = true;
    }

    /** Vänder myntet */
    public void flip() {
        tailUp = ! tailUp;
    }

    /** Tar reda på om myntet har klave uppåt */
    public boolean hasTailUp() {
        return tailUp;
    }
}
```


Designexempel: Nim-spel

Nim är ett spel för två personer, som spelas på en spelplan med tre högar. Spelet börjar med att ett antal stickor fördelas slumpmässigt på högarna. De båda spelarna turas därefter om att ta en eller två stickor, efter eget val, från någon av högarna. Den som tar den sista stickan har vunnit spelet.

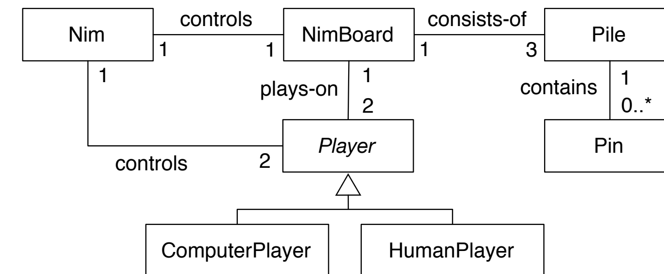
Skriv ett program där en spelomgång mellan en människa och datorn genomförs. "Människospelaren" skriver sina drag på tangentbordet. "Datorspelaren" väljer alltid den hög som innehåller minst antal stickor och tar två stickor ur den om högen innehåller mer än en sticka. Annars tar spelaren den enda stickan.

Klasser

Kandidater till klasser: spel, person, spelplan, hög, antal, sticka, spelare, val, program, spelomgång, människa, dator, människospelare, drag, tangentbord, datorspelare.

person	Samma som spelare, som är ett bättre namn
antal	Attribut (antalet stickor i spelet)
val	Substantiverat verb, något som utförs under spelet
program	Tillhör inte systemet
spelomgång	Något som utförs
människa	Samma som människospelare
dator	Samma som datorspelare
drag	Något som utförs
tangentbord	Tillhör inte systemet

UML-diagram



Scenario

1. Skapa en spelplan och två spelare.
2. Låt spelarna göra drag i tur och ordning, så länge det finns stickor kvar.

Mera detaljerat:

1. (Skapa spelplan) För alla stickor som ska fördelas:
   - Lägg en sticka i en hög som väljs slumpmässigt.
2. (Gör drag) Upprepa så länge det finns stickor kvar:
   - Låt aktuell spelare ta reda på antalet stickor i varje hög och välja hög enligt sin strategi.
   - Låt aktuell spelare ta ett antal stickor från den valda högen.

NimBoard, specifikation

NimBoard

```
/** Skapar en spelplan med tre högar med totalt
    n stickor */
NimBoard(int n);

/** Tar reda på antalet stickor i hög nr k */
int getNbrPins(int k);

/** Undersöker om det finns stickor kvar */
boolean hasMorePins();

/** Tar m stickor från hög nr k */
void takePins(int m, int k);
```

Pile, specifikation

Pile

```
/** Skapar en tom hög */
Pile();

/** Tar reda på antalet stickor i högen */
int getNbrPins();

/** Lägger m stickor i högen */
void putPins(int m);

/** Tar m stickor från högen */
void takePins(int m);
```


Player, specifikation

Player

```
/** Skapar en spelare med namnet name */
Player(String name);

/** Tar reda på spelarens namn */
String getName();

/** Gör ett drag på spelplanen board */
abstract void makeMove(NimBoard board);
```


ComputerPlayer och HumanPlayer, specifikationer

ComputerPlayer

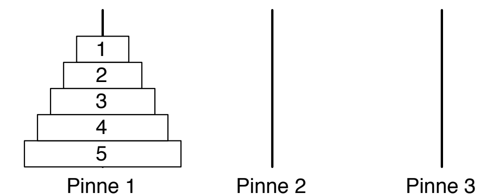
```
/** Skapar en datorspelare med namnet name */
ComputerPlayer(String name);

/** Väljer den minsta högen på spelplanen board,
    tar 2 stickor om det finns så många, annars 1 */
void makeMove(NimBoard board);
```

HumanPlayer

```
/** Skapar en människospelare med namnet name */
HumanPlayer(String name);

/** Gör ett drag på spelplanen board. Draget
    läses från tangentbordet */
void makeMove(NimBoard board);
```


Designexempel: Hanois torn

Det finns tre pinnar numrerade 1, 2, 3. Från början finns n brickor av avtagande storlek på pinne 1 med den största brickan underst. Pinne 2 och pinne 3 är tomma.



De n brickorna ska flyttas så att de hamnar i avtagande storlek på en av de övriga pinnarna. Detta ska ske i en följd av drag där man i varje drag flyttar den översta brickan från en pinne till en annan pinne. Den bricka som flyttas får aldrig placeras ovanpå en mindre bricka.

Algoritm

- I drag nr 1, 3, ... flyttar man den minsta brickan, bricka 1, till pinnen närmast till höger. Pinne 1 anses därvid finnas till höger om pinne 3.
- I drag nr 2, 4, ... flyttar man en bricka mellan de två pinnar som inte innehåller bricka 1.

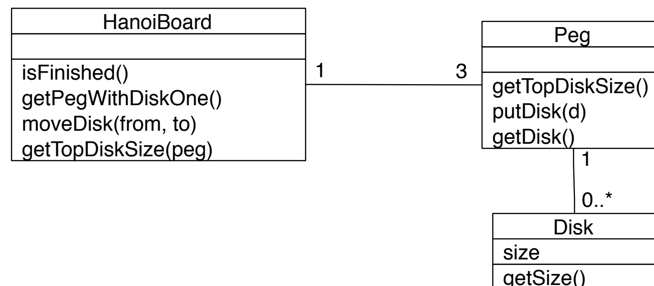
Skriv ett program som börjar med att läsa in antalet brickor som ska flyttas. Därefter ska brickorna flyttas enligt reglerna. Utskrift (exempel med tre brickor):

```
Flytta bricka 1 från pinne 1 till pinne 2
Flytta bricka 2 från pinne 1 till pinne 3
Flytta bricka 1 från pinne 2 till pinne 3
Flytta bricka 3 från pinne 1 till pinne 2
Flytta bricka 1 från pinne 3 till pinne 1
Flytta bricka 2 från pinne 3 till pinne 2
Flytta bricka 1 från pinne 1 till pinne 2
```


Scenario

- Lägg brickorna på pinne 1.
- Så länge spelet inte är slut:
- om udda drag:
  - tag reda på numret på pinnen där bricka 1 finns
  - flytta den översta brickan från denna pinne till pinnen närmast till höger
- annars (om jämnt drag):
  - tag reda på numret på pinnen där bricka 1 finns
  - räkna ut numren på de båda andra pinnarna
  - flytta den minsta brickan mellan dessa pinnar

Klasser och operationer



Disk, en bricka

```
public class Disk {
    private int size;

    /** Skapar en bricka med storleken size */
    public Disk(int size) {
        this.size = size;
    }

    /** Tar reda på brickans storlek */
    public int getSize() {
        return size;
    }
}
```

Peg, en pinne

```java
public class Peg {
    private ArrayList<Disk> disks;

    public Peg() { disks = new ArrayList<Disk>(); }

    public int getTopDiskSize() {
        return (! disks.isEmpty()) ?
               disks.get(disks.size() - 1).getSize() :
               Integer.MAX_VALUE;
    }

    public void putDisk(Disk d) {
        disks.add(d);
    }

    public Disk getDisk() {
        Disk d = disks.remove(disks.size() - 1);
        return d;
    }
}
```


HanoiBoard, spelplanen, 1

```java
public class HanoiBoard {
    private Peg[] pegs;

    public HanoiBoard(int nbrDisks) {
        pegs = new Peg[3];
        for (int i = 0; i < pegs.length; i++) {
            pegs[i] = new Peg();
        }
        for (int i = nbrDisks; i >= 1; i--) {
            pegs[0].putDisk(new Disk(i));
        }
    }

    public int getPegWithDiskOne() {
        int peg1 = 0;
        while (getTopDiskSize(peg1) != 1) {
            peg1++;
        }
        return peg1;
    }
```


HanoiBoard, spelplanen, 2

```java
    public boolean isFinished() {
        return isEmpty(0) &&
               (isEmpty(1) || isEmpty(2));
    }

    public int getTopDiskSize(int peg) {
        return pegs[peg].getTopDiskSize();
    }

    public void moveDisk(int from, int to) {
        Disk d = pegs[from].getDisk();
        pegs[to].putDisk(d);
    }

    private boolean isEmpty(int peg) {
        return pegs[peg].getTopDiskSize() ==
               Integer.MAX_VALUE;
    }
}
```


HanoiStrategy, algoritmen, 1

```java
public class HanoiStrategy {
    private HanoiBoard board;

    public HanoiStrategy(HanoiBoard board) {
        this.board = board;
    }

    public void moveDisks() {
        int moveNbr = 1;
        while (! board.isFinished()) {
            moveOneDisk(moveNbr);
            moveNbr++;
        }
    }
```

HanoiStrategy, algoritmen, 2

```
    private void moveOneDisk(int moveNbr) {
        int peg1 = board.getPegWithDiskOne();
        int from; // pinne att flytta en bricka från
        int to;   // pinne att flytta en bricka till
        if (moveNbr % 2 != 0) {
            from = peg1;
            to = (from + 1) % 3;
        } else {
            from = (peg1 + 1) % 3;
            to = (from + 1) % 3;
            if (board.getTopDiskSize(from) >
                board.getTopDiskSize(to)) {
                int temp = from;
                from = to;
                to = temp;
            }
        }
        System.out.println(...);
        board.moveDisk(from, to);
    }
```


TowersOfHanoi, main-metoden

```
import java.util.Scanner;

public class TowersOfHanoi {
    public static void main(String[] args) {
        System.out.print("Antal brickor: ");
        Scanner scan = new Scanner(System.in);
        int nbrDisks = scan.nextInt();
        HanoiBoard board = new HanoiBoard(nbrDisks);
        HanoiStrategy strategy = new HanoiStrategy(board);
        strategy.moveDisks();
    }
}
```


Autoboxing och autounboxing

- Om ett `int`-värde `value` förekommer där det behövs ett `Integer`-objekt så utförs automatiskt `new Integer(value)`.
- Om ett `Integer`-objekt `obj` förekommer där ett `int`-värde behövs så utförs automatiskt `obj.intValue()`.

```
public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            list.add(nbr);          // autoboxing
        }
        for (int i = list.size() - 1; i >= 0; i--) {
            int nbr = list.get(i); // autounboxing
            System.out.println(nbr);
        }
    }
}
```


Varning för autoboxing och autounboxing

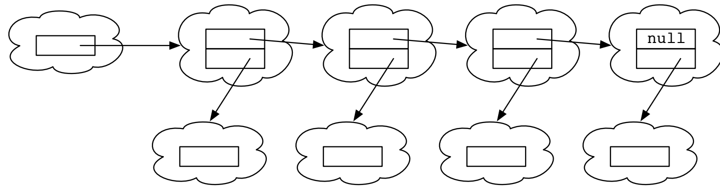
Tester på likhet och olikhet kanske inte fungerar:

```
ArrayList list = new ArrayList<Integer>();
...
Integer i1 = list.get(0);
Integer i2 = list.get(1);

if (i1 <= i2) ... // fungerar
if (i1 == i2) ... // fungerar INTE!
```

Parametrar kan också medföra problem. För att lägga in ett tal `nbr` på platsen `pos` i en lista kan man skriva `add(pos,nbr)`, och det blir autoboxing på `nbr`. Om man i ett anrop råkar skriva parametrarna i fel ordning upptäcks inte detta, utan det blir autoboxing på `pos` och talet `pos` läggs in på plats `nbr`.

Länkade listor

Enkellänkad lista:



För- och nackdelar jämfört med vektor:

+ Dynamisk struktur.
+ Det går snabbt att lägga in och ta bort element "mitt i" listan.
- Det tar lång tid att hitta ett element på en given plats — man måste börja från början och följa referenserna.

Gå igenom lista

Gå igenom en lista från början till slut:

```
ArrayList<Integer> list = new ArrayList<Integer>();
...
int i = 0;
while (i < list.size()) {
    Integer obj = list.get(i);
    // ... gör någonting med objektet obj
    i++;
}
```

Om listan är en `LinkedList` (standardklass, dubbellänkad lista) är det *inte* bra att gå igenom listan på detta sätt — `get(i)` börjar varje gång från början av listan och räknar fram `i` steg.

Iteratorer

Iterator

```
/** Flyttar aktuell position ett steg framåt i listan
    och returnerar det nya aktuella elementet. Om det
    inte finns något element på denna plats fås
    NoSuchElementException */
E next();

/** Undersöker om det finns fler element i listan */
boolean hasNext();
```

```
Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    Integer obj = it.next();
    // ... gör någonting med objektet obj
}
```


Ny `for`-sats

Med följande variant av `for`-satsen går man igenom alla elementen i en lista:

```
LinkedList<Integer> list = new LinkedList <Integer>();
...
for (Integer obj : list) {
    // ... gör någonting med objektet obj
}
```

Exempel (klassen `Polygon`):

```
public void move(int dx, int dy) {
    for (Vertex v : vertices) {
        v.move(dx, dy);
    }
}
```