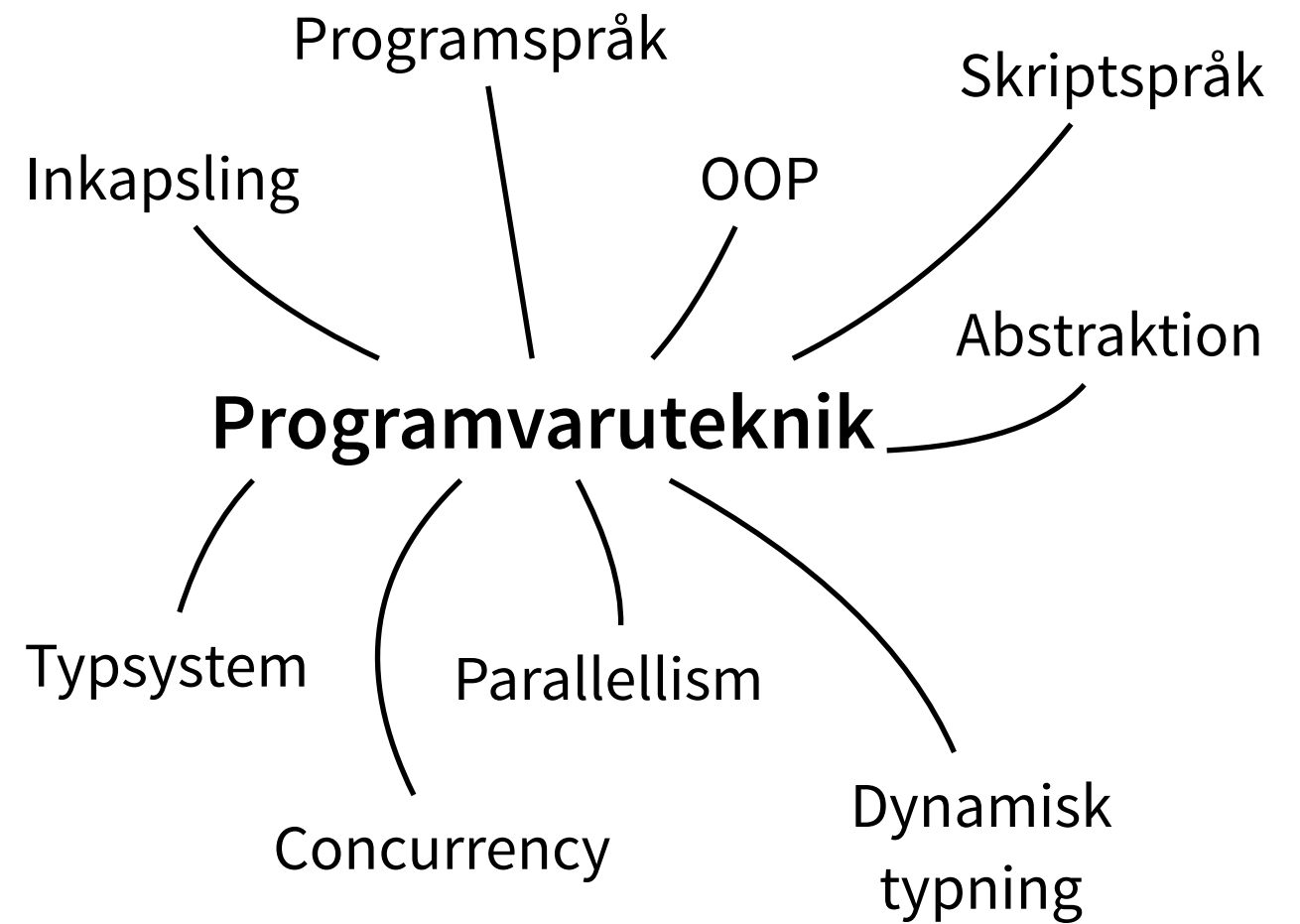


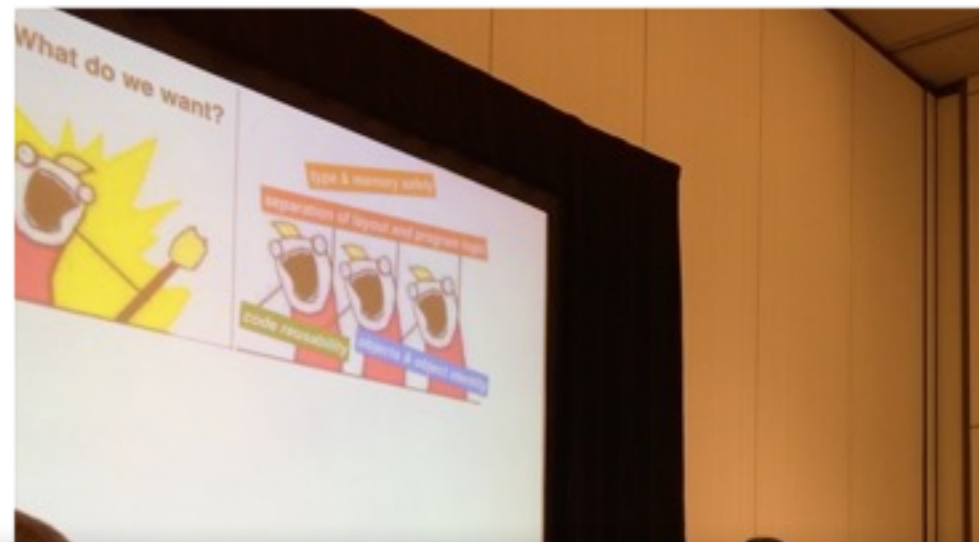




The ACM **SIGPLAN** conference on **Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)** embraces all aspects of software construction and delivery to make it the premier conference at the intersection of programming, languages, and software engineering.

OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) is an annual ACM research conference. **OOPSLA** mainly takes place in the United States, while the sister conference of OOPSLA, **ECOOP**, is typically held in Europe.





SPLASH 2017 Activities

- Author of (CER) Mastery Learning-Like Teaching with Achievements within the SPLASH-E-track in SPLASH 2017
- Author of Reference Capabilities for Concurrency & Scalability: an Experience Report within the OCAP 2017-track in OCAP 2017
- Author of You Can Have it All: Abstraction and Good Cache Performance within the Onward! Papers-track in Onward! 2017
- Author of Orca: GC and Type System Co-Design for Actor Languages within the OOPSLA-track in SPLASH 2017
- Co-chair in Organizing Committee for NOOL 2017





This paper introduces Orca, Pony's concurrent garbage collection algorithm. Orca stands for Counting-based Garbage Collection in the Actor World. It was co-

Welcome to SPLASH 2018!



The ACM SIGPLAN conference on **Systems, Programming, Languages and Applications: Software for Humanity** embraces all aspects of software construction and delivery to make it the premier conference at the intersection of programming, languages, and software engineering. SPLASH 2018 will take place in Boston from Sunday 4th to Friday 9th of November 2018.

SPLASH includes the following co-located conferences: [OOPSLA](#), [Onward!](#), [GPCE](#), [SLE](#), and [DLS](#); as well as sixteen workshops.

The [SPLASH-I talk series](#) features thirty-five talks targeting practitioners. In the evenings, attendees can join one of the eight [meetup groups](#) held at MIT and Northeastern University.

Students curious about research can attend the [Programming Language Mentoring Workshop](#) and our [Mentoring Breakfasts](#). Students who have some research under their belt can either take part in the [Student Research Competition](#) or the [Doctoral Symposium](#). For educators, the [SPLASH-E](#) sessions will have invited talks and discussions.

OOSPLA is committed to open and reproducible science. All papers are published in gold open access with [PACMPL](#). Scientific claims are evaluated by the [Artifact Evaluation Committee](#).

Dave Clarke ersätter mig på 2–3 föreläsningar — info i Piazza



Introduktion till OO

Tobias Wrigstad
tobias.wrigstad@it.uu.se

Föreläsning 15



What is object-oriented software

Objects are like people. They're living, breathing things that have knowledge inside them about how to do things and have memory inside them so they can remember things. And rather than interacting with them at a very low level, you interact with them at a very high level of abstraction [...]



Ole Johan Dahl

Kristen Nygaard





©FLT-PICA, Stockholm, Sverige/Sweden. 1998-05-20 Foto Lennart Nygren/FLT-PICA code 30132 ***Arkivbild/File Picture
1978-01-13***
Arbete vid dataterminal i FOA:s datasystem

Jacob Palme



Why Smalltalk?

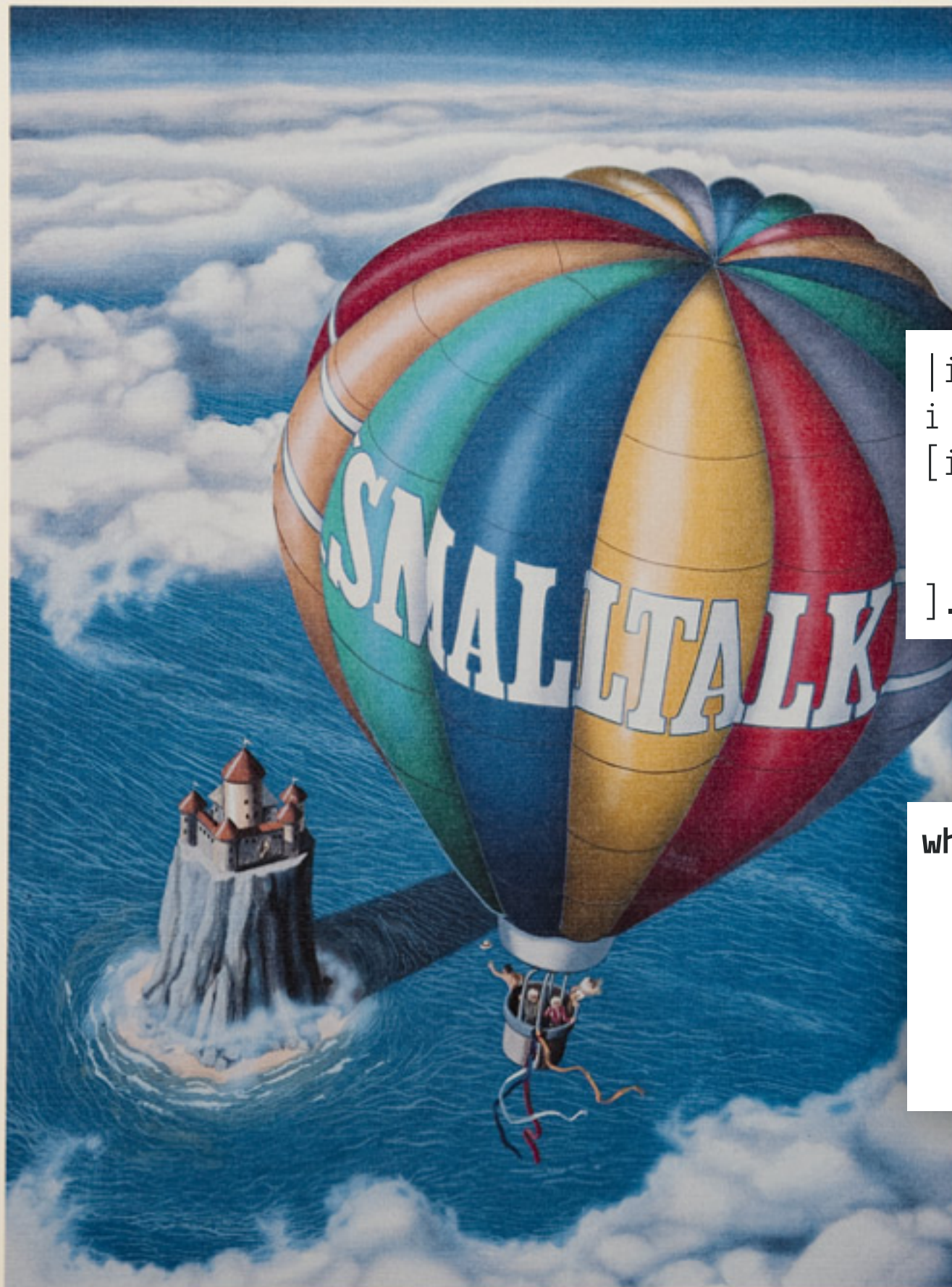
I made up the term “object-oriented,” and I can tell you I did not have C++ in mind.

– Alan Kay



Alan Kay





```
|i|  
i := 5.  
[i > 0] whileTrue: [  
    Transcript show: ((i*2) asString) ; cr.  
    i := i - 1.  
].
```

```
whileTrue: aBlock  
^self value  
  ifTrue: [  
    aBlock value.  
    [self value] whileTrue: [aBlock value].  
  ].
```




Objekt

- Gäst A, gäst B, gäst C...
- Kypare A, kypare B, ...
- Kock A, ...
- Tallrikar med mat
- En massa bord
- En massa stolar
- Dukar, glas, bestick, ...

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

Arv

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Aktiva: serveringspersonal, gäster, kockar

Passiva: maten, stolarna, borden, etc.

Meddelandesändning

Aggregering

Inkapsling

Arv

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Mellan aktiva objekt: beställa mat

Aktiva—passiva objekt: dra ut stol, äta, lyfta en gaffel

Aggregering

Inkapsling

Arv

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Ett bord består av en bordsskiva och fyra ben

Ett middagssällskap består av flera gäster

Inkapsling

Arv

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

En gäst kan inte interagera direkt med kökspersonalen

Hur maten lagas (Det är inte uppenbart att det är hästkött i lasagnen, jmf. abstraktion)

Arv

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

Arv

En Gäst är en Person, en Kypare är en Person, en Kock är en Person

Om $\mathcal{P}(\text{Person}) \implies \mathcal{P}(\text{Gäst}) \wedge \mathcal{P}(\text{Kypare}) \wedge \mathcal{P}(\text{Kock})$

Polymorfism

Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

Arv

Polymorfism

Olika objekt kan ha samma gränssnitt

Olika maträtter smakar olika, personer agerar olika, etc.

Kockarna går också på restaurang som gäster, man kan dricka vin ur ölglas

Koncept

Objekt

Klasser – ritningar för objekt

Koncept

Objekt

Världen består av objekt (som består av objekt...) som skickar **meddelanden** till varandra

Objekt "av samma sort" grupperas i **klasser** som beskriver hur objekten fungerar

Relationer mellan objekt: **aggregering** (objekt har referenser till andra objekt)

Ett objekts "byggstenar" är inte direkt åtkomliga (**inkapsling**)

Klasser – ritningar för objekt

Koncept

Objekt

Världen består av objekt (som består av objekt...) som skickar **meddelanden** till varandra

Objekt "av samma sort" grupperas i **klasser** som beskriver hur objekten fungerar

Relationer mellan objekt: **aggregering** (objekt har referenser till andra objekt)

Ett objects "byggstenar" är inte direkt åtkomliga (**inkapsling**)

Klasser – ritningar för objekt

Beskriver inte bara vad ett objekt innehåller (**tillstånd**) utan också dess **beteende**

Relationer mellan klasser: **arv** (En pudel är en hund är ett djur är ett...)

Hur en klass är uppbyggd internt är inte synligt utifrån (**inkapsling**)

Procedurell programmering

$f(x)$ — du bestämmer ”nu skall jag göra f på datat x ”

Objektorienterad programmering

$x.f()$ — du ber ”snälla objekt x , utför f ”



Statisk bindning i C

$f(x)$ — gcc väljer f beroende på x :s typ vid kompilering

Dynamisk bindning i Java

$x.f()$ — VM:en väljer f beroende på vad som finns i x under körning!



Introduktion till OOP

med Java

Tobias Wrigstad
tobias.wrigstad@it.uu.se



En liten Java-parlör

Svenska

Engelska

Objekt

Object

Klass

Class

Arv

Inheritance

Instansvariabel / fält

Instance variable / field

Metod

Method

Superklass / basklass

Super class / base class

Subklass / härledd klass

Sub class / derived class

Abstrakt klass

Abstract class

Superanrop

Super call

Svenska

Engelska

Metodspecialisering

Overriding

Överlagring

Overloading

Överskuggning

Shadowing

Klasshierarki

Class hierarchy

Aggregering

Aggregation

Typomvandling

Type cast

Polymorfism

Polymorphism

Barnklass

Child / sub / derived class

Instantieras

Instantiate

Tolka Javakompilatorns felmeddelanden!

```
<Filnamn.Java>:<Radnummer>: error: <Beskrivning av felet>
```

```
    <information om var det uppstår>
```

```
    <övrig hjälpinformation om tillämpligt>
```

Mall

```
CommonCompilerErrors.java:66: error: cannot find symbol
```

```
    LinkedList myList;
```

```
    ^
```

```
symbol:   class LinkedList
```

```
location: class ErrorThree
```

Exempel

Förstå kompilatorns språk

Kompilatorn säger

Betyder i regel

`cannot find symbol`

Felstavat namn, eller namnet är inte synligt ännu, t.ex. inte importerat in, alt finns inte

`method X cannot be applied`

Argumentlistans typer fel (för få argument, fel ordning, fel argument?)

`incompatible types`

Typen på högersidan är inte kompatibel med den till vänster. Är de subtyper?

`X cannot be converted to String`

Glömt att anropa `toString()`?

Objekt

- En samling data (tillstånd) samt operationer som opererar på datat
- Man kan skicka meddelanden till ett objekt

Objektet väljer själv vad som skall utföras som svar på ett meddelande

- Objekt-orienterad design är data-driven design

Vilka aktörer finns det?

Hur är de relaterade med **arv**, **aggregering**, **användning**, etc. (mer senare)

Klass (finns i nästan alla OO-språk)

- En klass är en ”ritning” från vilken man kan bygga oändligt många objekt

- Medlemmar

Instansvariabler (även fält)

Metoder

- En klass är ung. som en strukt + alla funktioner som opererar på strukten

- Saker vi skall prata om senare

Relationer mellan klasser

Åtkomstmodifikatorer

Arv

- **Instansiering:** att skapa ett objekt från en klass

Java

- Utvecklades av Sun (James Gosling) under 90-talets början, släpptes 1995
- Några designprinciper för Java

Enkelt, objektorienterat och familjärt

Robust och säkert

Arkitekturoberoende och portabelt

Snabbt

Tydligt

Klassen Foo måste ligga i Foo.java

Ett första Java-program

```
/**
 * @author Tobias Wrigstad (tobias.wrigstad@it.uu.se)
 * @date 2013-10-01
 */
public class Hello {
    String who = null;
    public Hello (String who) {
        this.who = who;
    }
    public void greet() {
        System.out.println("Hello " + who);
    }
    public static void main(String args[]) {
        if (args.length > 0) {
            Hello hello = new Hello(args[0]);
            hello.greet();
        } else {
            System.out.println("Usage: java Hello <who>");
        }
    }
}
```

Koncept

- Klass
- Objekt
- Instansvariabel
- Konstruktor
- Metod
- Main-metod
- Arrayer är objekt
- Instantiering
- Metodanrop
- Åtkomstmodifierare
- En vettig sträng-typ

```
/**
 * @author Tobias Wrigstad (tobias.wrigstad@it.uu.se)
 * @date 2013-10-01
 */
public class Hello {
    String who = null;
    public Hello (String who) {
        this.who = who;
    }
    public void greet() {
        System.out.println("Hello " + who);
    }
    public static void main(String args[]) {
        if (args.length > 0) {
            Hello hello = new Hello(args[0]);
            hello.greet();
        } else {
            System.out.println("Usage: java Hello <who>");
        }
    }
}
```

Kompilera och köra...

- Kompilatorn "javac"

Förstår beroenden

Kompilerar till "Java-bytekod"

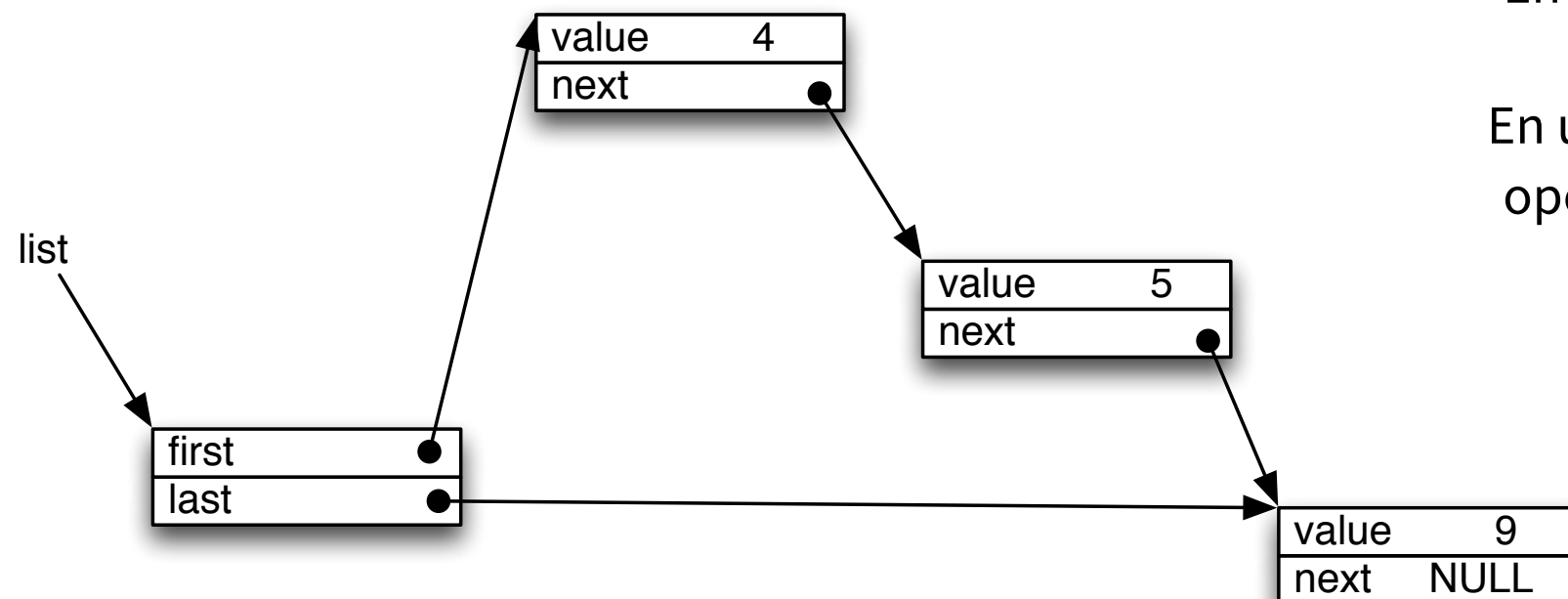
- Programmet måste köras i den virtuella maskinen

Programmet "java"

Tar som argument namnet på en klass med en main-metod

```
$ javac Hello.java
$ ls
Hello.java
Hello.class
$ java Hello
Usage: java Hello <name>
$ java Hello "Tobias"
Hello Tobias!
$
```

Länkad lista i C

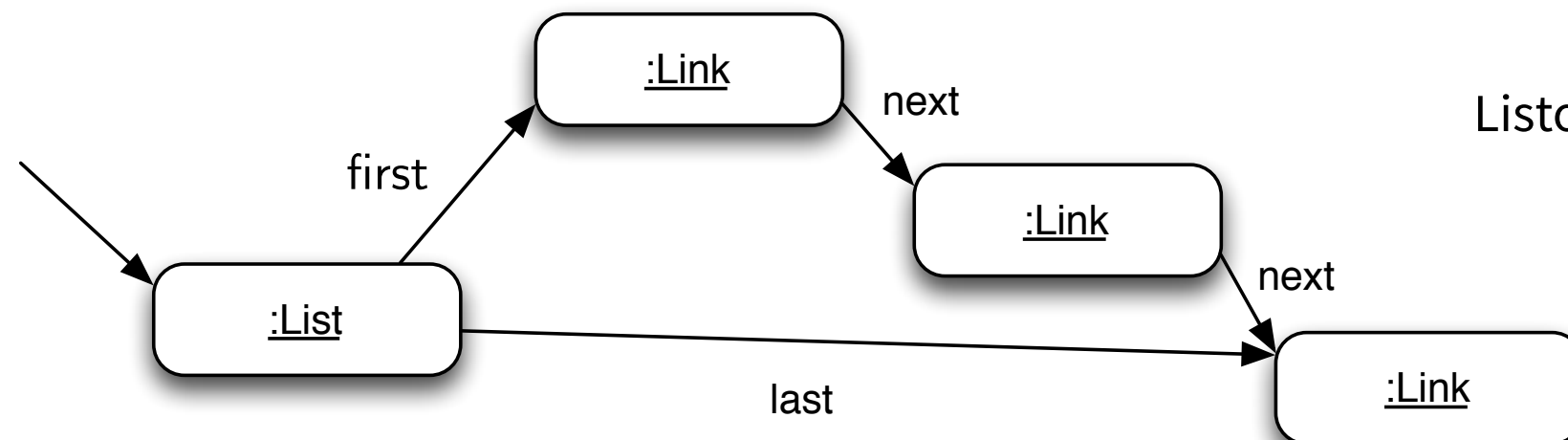


Varje link är en mallokerad struct

En länk *pekar* ut sin nästa länk

En uppsättning funktioner som opererar på alla delar av listan

Länkad lista i Java



Varje länk är ett objekt

En länk *refererar* sin nästa länk

Listobjekten och länkobjekten har separat tillstånd och definierar ett eget beteende



Observationer

- Två klasser: `List` och `Link`

`List`-objekt aggregerar länk-objekt

Rekursion och iteration som i C (märk att det rekursiva anropet växlar mottagare!)

- Privat och publik åtkomst

Kräver `set-` och `get-`metoder i `Link` för privata medlemsvariabler

Referenser $\neq$ pekare

- En referens är ett handtag till ett objekt — det är inte en adress till en plats i minnet

Alla valida pekare i C pekar inte på det de skall (eller något alls)

Alla referenser i Java pekar alltid på det de skall och på någonting!

- Referensen null är inte adressen 0
- Den är inte heller ett booleskt värde
- Referenser möjliggör automatisk minneshantering (GC)

Automatisk minneshantering

- Java hanterar minnet automatiskt

`new` `ClassName(...)` allokerar automatiskt nog med minne på heapen

När sista referensen till ett objekt tas bort är objektet skräp

När minnet blir fullt städas skräp bort automatiskt för att lämna plats för nya objekt

- Alltså:

Ingen `malloc` (`new` allokerar alltid på heapen, åtminstone vad du vet!)

Ingen `free`

Förrädiskt likt C

- Syntaxen vald för att göra det enkelt för C och C++-programmerare att programmera Java
- Många konceptuella skillnader (Java är mer likt Smalltalk än C++)

- **Men:** vi **kan** ta med oss mycket från C!

While, for, if, switch, variabeldeklarationer, funktionsyntax, primitiva typer, etc....

I stort sett vet ni redan hur man programmerar Java, bara *inte hur man programmerar **objektorienterat** i Java!*

- Tag er i akt så ni inte programmerar C i Java!