

Introduktion till OO

Tobias Wrigstad
tobias.wrigstad@it.uu.se





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, ...



Objekt-orienteringens grundkoncept

- Aktiva och passiva objekt
- Meddelandesändning
- Aggregering
- Inkapsling
- Arv
- Polymorfism



Objekt-orienteringens grundkoncept

- Aktiva och passiva objekt

Aktiva: serveringspersonal, gäster, kockar

Passiva: maten, stolarna, borden, etc.

- Meddelandesändning

- Aggregering

- Inkapsling

- Arv

- Polymorfism



Objekt-orienteringens 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



Objekt-orienteringens 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



Objekt-orienteringens 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



Objekt-orienteringens 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 $\mathcal{P}(\text{Person}) \Rightarrow \mathcal{P}(\text{Gäst}) \wedge \mathcal{P}(\text{Kypare}) \wedge \mathcal{P}(\text{Kock})$

- Polymorfism



Objekt-orienteringens 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** (och använder)

Ett objects "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** (och använder)

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** (och använder)

Hur en klass är uppbygd 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



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.



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

- **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



Den virtuella maskinen

kompilerat
C-program

kompilerat
Java-program

JVM

OS

OS

HW

HW



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
- En vettig sträng-typ
- Åtkomstmodifierarer

```
/**  
 * @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

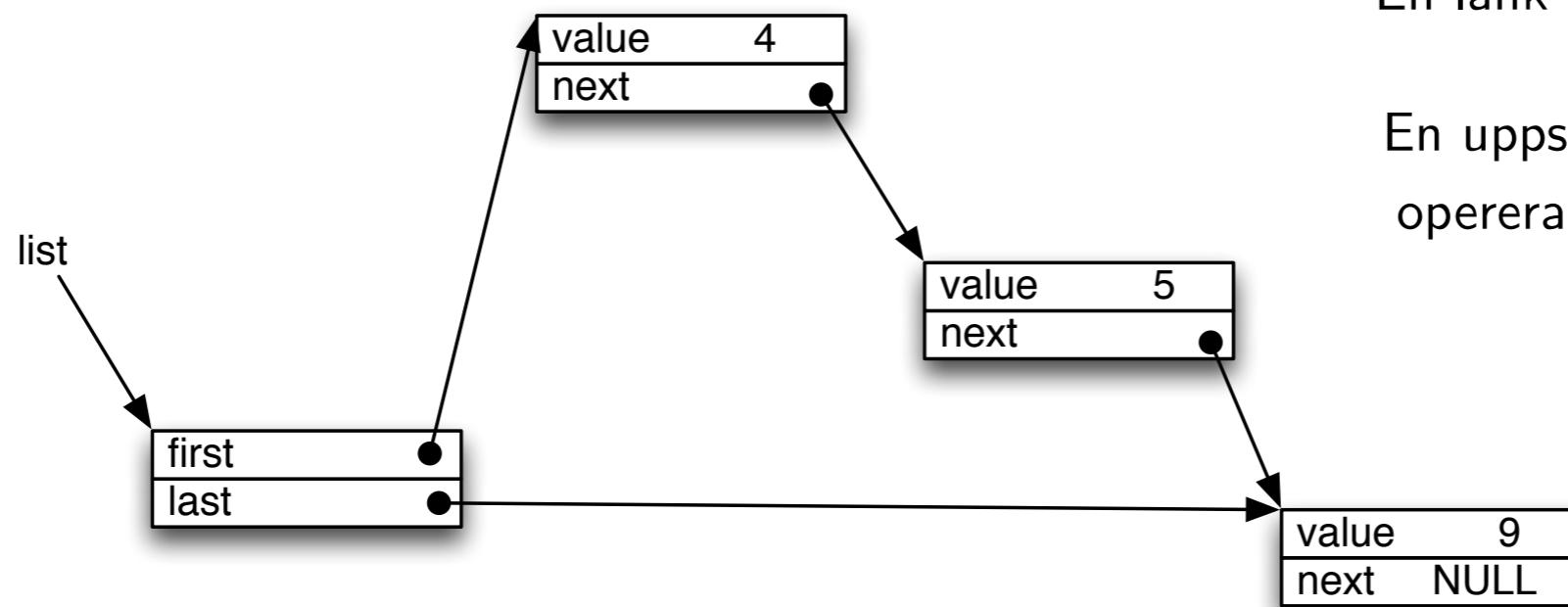
Programmet "**java**"

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

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



Länkad lista i C

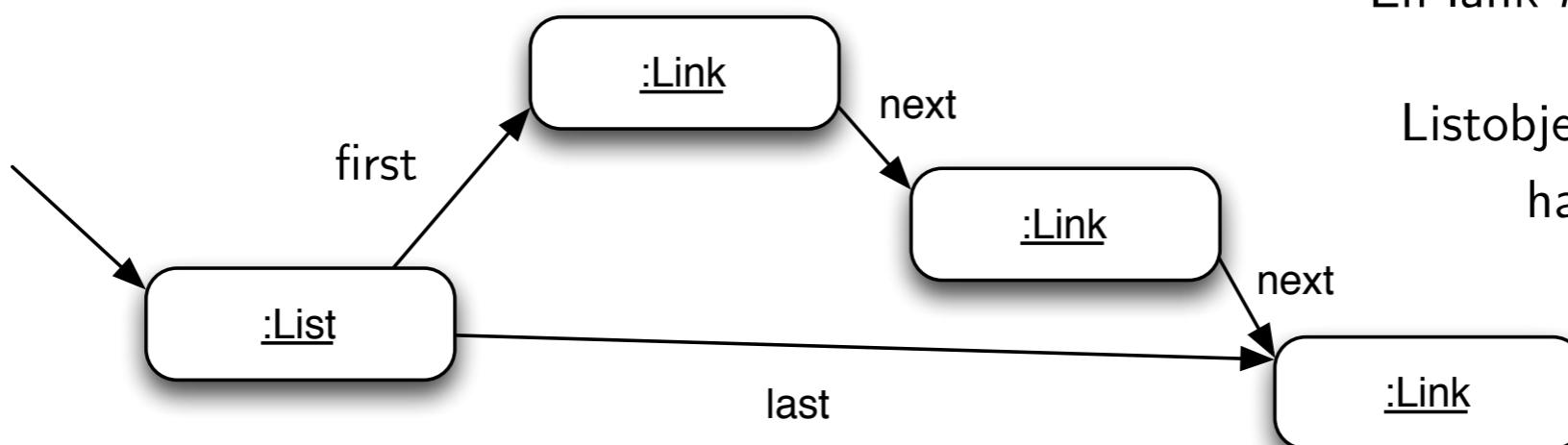


Varje länk är en allokerad struct

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



En länkad lista med OOP

```
public class List {
```

```
    private Link first = null;  
    private Link last = null;
```

```
    public class Link {
```

```
        private Object element = null;  
        private Link next = null;
```

En länkad lista med OOP

```
public class List {  
  
    // Instansvariabler  
    private Link first = null;  
    private Link last = null;  
  
    // Metoder  
    public void prepend(final Object element) {  
        first = new Link(element, first); ←  
        if (last == null) {  
            this.last = first;  
        }  
    }  
  
    public class Link {  
  
        // Instansvariabler  
        private Object element;  
        private Link next;  
  
        // Konstruktor  
        public Link(Object element, Link next) {  
            this.element = element;  
            this.next     = next;  
        }  
    }  
}
```

```
public class List {  
    // Instansvariabler  
    private Link first = null;  
    private Link last = null;  
  
    // Metoder  
    public void prepend(final Object element) {  
        this.first = new Link(element, first);  
        if (last == null) {  
            this.last = first;  
        }  
    }  
  
    public void append(final Object element) {  
        Link newLink = new Link(element, null);  
        if (first == null) {  
            this.last = this.first = newLink;  
        } else {  
            last.setNext(newLink); // * setter krävs pga private next i Link  
            this.last = newLink;  
        }  
    }  
}  
// i Link-klassen  
public void setNext(final Link next) {  
    this.next = next;  
}
```



En länkad lista med OOP

```
// ... forts metoder i class List
```

```
public Object get(final int index) {  
    if (index == 0) {  
        return first.getElement();  
    } else {  
        first.get(index-1);  
    }  
}
```

```
// i Link-klassen  
public Object getElement() {  
    return this.element;  
}
```

```
public Object get(final int index) {  
    return (index <= 0) ? this.element : this.next.get(index);  
}
```

En länkad lista med OOP

```
// ... forts metoder i class List

public Object get(final int index) {
    if (index == 0) {
        return first.getElement();
    } else {
        return first.get(index-1);
    }
}

public int length() {
    int length = 0;      // ** nedan - getter krävs pga private next i Link
    for (Link cursor = first; cursor != null; cursor = cursor.getNext()) {
        ++length;
    }
    return length;
}
}

// i Link-klassen
public Link getNext() {
    return this.next;
}
```



```
public class Link {  
  
    // Instansvariabler  
    private Object element;  
    private Link next;  
  
    // Konstruktor  
    public Link(Object element, Link next) {  
        this.element = element;  
        this.next    = next;  
    }  
  
    // Metoder  
    public Object getElement() { return this.element; }  
    public Link getNext()      { return this.next; }  
    public void setNext(final Link next) { this.next = next; }  
    public Object get(final int index) {  
        return index <= 0 ? this.element : this.next.get(index-1);  
    }  
}
```

En remove()-metod

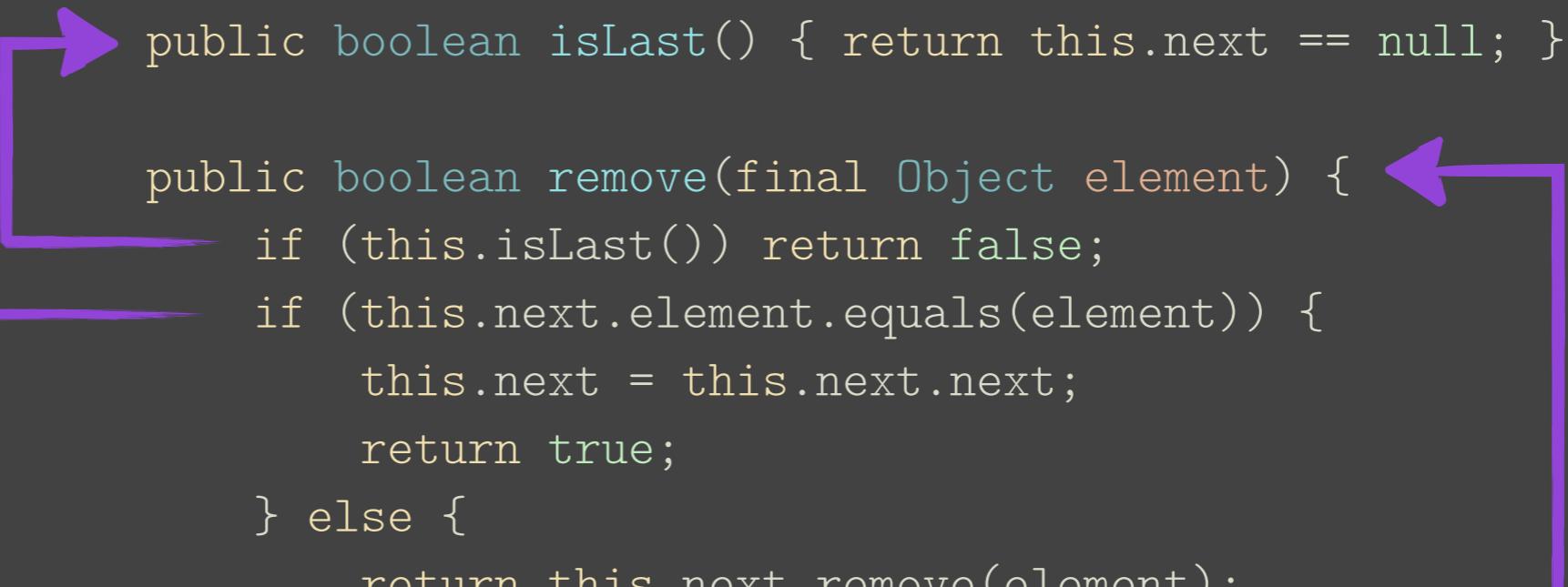
```
public class List {  
    private Link first = null;  
  
    public boolean remove(final Object element) {  
        if (first == null) return false;  
        if (first.getElement().equals(element)) {  
            first = first.getNext();  
            return true;  
        } else {  
            return first.remove(element); // i Object-klassen  
        }  
    }  
}
```

```
// i Object-klassen  
public boolean equals(Object other) {  
    ...  
}
```

```
// i Link-klassen  
public boolean remove(Object element) {  
    ...  
}
```

```
public class List {  
    private Link first = null;  
    public boolean remove(final Object element) {  
        if (this.first == null) return false;  
        if (this.first.getElement().equals(element)) {  
            this.first = this.first.getNext();  
            return true;  
        } else {  
            return first.remove(element);  
        }  
    }  
}
```

En remove()-metod

```
// Object  


```
public class Link {
 private Object element;
 private Link next;

 public boolean isLast() { return this.next == null; }

 public boolean remove(final Object element) {
 if (this.isLast()) return false;
 if (this.next.element.equals(element)) {
 this.next = this.next.next;
 return true;
 } else {
 return this.next.remove(element);
 }
 }
}
```


```

Observationer

- Två klasser: List och Link

List-objekt aggergerar länk-objekt

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

- Privat och publik åtkomst

Kräver set- och get-metoder i Link för rad * och ** i listan



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)

Varför?



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!

