Introduktion till OO

Tobias Wrigstad
tobias.wrigstad@it.uu.se

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

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].
    ].
```

ACM Turing Awards kopplade till OOP

Ole-Johan Dahl Kristen Nygaard

Barbara Liskov

Alan Kay

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 $\mathscr{P}$(Person) $\Longrightarrow$ $\mathscr{P}$(Gäst) $\wedge$ $\mathscr{P}$(Kypare) $\wedge$ $\mathscr{P}$(Kock)

- Polymorfism

Objektorienteringens grundkoncept

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

    Olika objekt kan ha samma gränssnitt

    Olika maträtter smakar olika, personer reagerar olika på samma input, 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 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, vad tror du om att göra 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

En liten Java-parlör (stämmer för de flesta OOPLs)

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 av principerna för Javas design:

  Enkelt, objektorienterat och familjärt

  Robust och säkert

  Arkitekturoberoende och portabelt

  Snabbt

  Tydligt

Java

Enkelt, objektorienterat och familjärt

Familjär syntax: låna C:s syntax; familjär semantik: låna Smalltalks semantik och några element från C++

Robust och säkert

Inga dangling pointers, inga invalid reads, inga invalid writes, ingen minneskorruption, ingen svag typning, …

Arkitekturoberoende och portabelt

Aldrig mer: ”mitt program funkar hemma men kraschar på institutionens datorer”

Snabbt

Kräver lite mer av programmeraren för att kompilatorn skall kunna göra ett bättre jobb med programmet

Tydligt

Ex.: Klassen Foo måste ligga i filen 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
- Åtkomstmodifierarere
- 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!
$
```

Referenser ≠ pekare

- Förutom de primitiva datatyperna (boolean, integer, etc.) är alla värden i ett Java-program *objekt*

  Alla objekt har referenssemantik, dvs. man kan inte (som i C) välja att lägga ett objekt på stacken eller skicka in en kopia (*)

  Annorlunda uttryckt: alla icke-primitiva data i Java är **referenser**

- 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.…

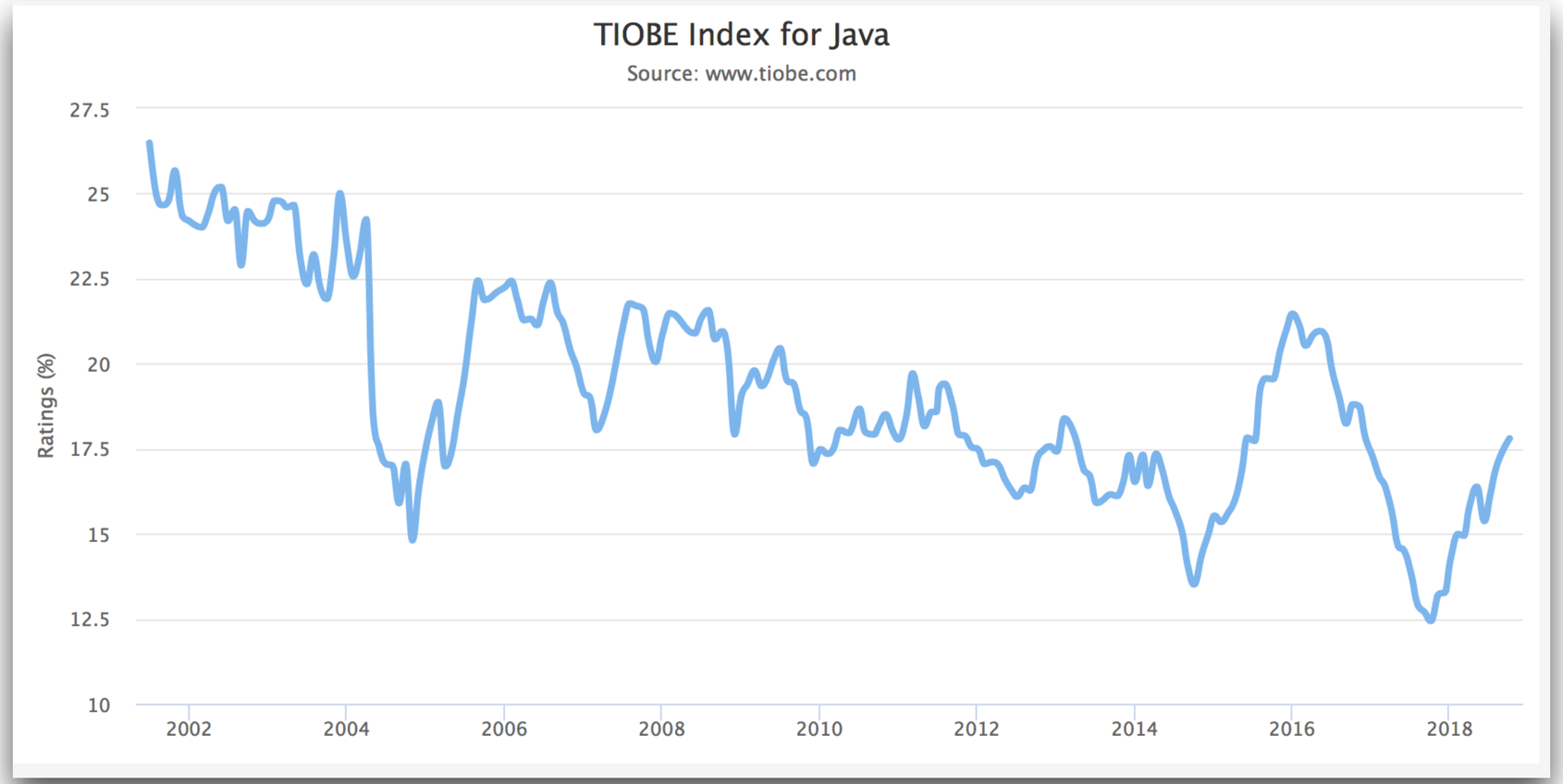
  I stort sett vet ni redan hur man programmerar Java, bara *inte hur man programmerar* ***objektorienterat*** *i Java!*

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

Föreläsning 16

Tobias Wrigstad

Vem använder Java?



Highest Position (since 2001): #1 in Oct 2018

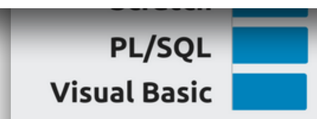
Lowest Position (since 2001): #2 in Mar 2015

Källa: Tiobe

Top Programming Languages

Tiobe Index - December 2017

Oct 2020	Oct 2019	Change	Programming Language
1	2	↑	C
2	1	↓	Java
3	3		Python
4	4		C++
5	5		C#
6	6		Visual Basic
7	7		JavaScript
8	9	↑	PHP
9	15	⇈	R
10	8	↓	SQL
11	19	⇈	Perl



1.37%

1.35%

Källa: Tiobe

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
C	1	2	2	1	1	2	1	1
Java	2	1	1	2	3	29	-	-
Python	3	6	6	6	21	15	-	-
C++	4	3	3	3	2	1	2	9
C#	5	4	5	7	9	-	-	-
JavaScript	6	8	8	10	7	-	-	-
PHP	7	7	4	5	19	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	16	-	-	-	-	-	-
R	10	13	49	-	-	-	-	-
Lisp	29	25	15	13	8	5	6	2

Källa: Tiobe

Java är plattformsoberoende

Datatyper har en standardstorlek

Samtliga klasser i standardbiblioteket finns på alla maskiner

Javas minnesmodell är densamma på alla maskiner

Ditt program kommer att bete sig likadant på din kompis dator

Du behöver inte ens kompilera om programmet — du kan flytta det rakt av

Caveat: Gränssnittsprogram och OS-specifika tjänster

`C:\foo\bar.txt` vs. `/foo/bar.txt`

Type	Default	Size	Example Literals
boolean	false	1 bit	`true`, `false`
byte	0	8 bits	(none)
char	`\u0000`	16 bits	`'a'`, `'\u0041'`, `'\101'`, `'\`
short	0	16 bits	(none)
int	0	32 bits	`-2`, `-1`, `0`, `1`, `2`
long	0	64 bits	`-2L`, `-1L`, `0L`, `1L`, `2L`
float	0.0	32 bits	`1.23e100f`, `-1.23e-100f`,
double	0.0	64 bits	`1.23456e300d`,

Datatyperna är samma oavsett plattform

Automatisk minneshantering

Objects know their size
(but we may not!)

```
new LinkedList();
```

Unreachable objects
are reclaimed

```
/// This program does not leak
LinkedList list = new LinkedList();
for (int i = 0; i < 1000000; ++i) {
    list.add(new Object());
}
list = null;
```

Unused objects
are not…

```
/// This program does might "leak"
LinkedList list = new LinkedList();
for (int i = 0; i < 1000000; ++i) {
    list.add(new Object());
}
```

Metadata

En objekt känner sitt ursprung

```
Object o = new Person();

o instanceof Person // true

Class c = o.getClass();

c.newInstance(); // skapa en ny person
```

Reflection och introspection

```
Method m = o.getClass().getMethod("setName", String.class);

ms.invoke(o, "Barbara")

for (Method m : c.getMethods()) { if (m.startsWith("test")) m.invoke(); }
```

...and more, e.g., `array.length`

Inkapsling

- Namnbaserad inkapsling styr vem som får nämna ett visst namn
- `Node` är bara en valid typ inuti `LinkedList`
- Kräver aktivt ställningstagande från dig!
- Standard är ”package” — dvs. åtkomligt överallt från modulen

```
public class Pair {
    private Object fst;
    private Object snd;
    Object getFst() { return this.fst; }
    void setFst(Object o) { this.fst = o; }
}
```

```
public class LinkedList {
    private Node first = new Node();
    private class Node {
        Node next;
        Object element;
        public Node(Object o, Node n) {
            this.element = o;
            this.next = n;
        }
    public void prepend(Object o) {
        this.first =
            new Node(o, this.first);
    }
…
```

Världens rikaste standardbibliotek(?)

Sök på ”java 15 api KlassensNamn”

Genererat med JavaDoc utifrån kommentarer i källkoden — inspiration för D9

Indelat i paket; viktigaste paket för er:

`java.lang`

Grundläggande objekt, och systemobjekt

`java.util`

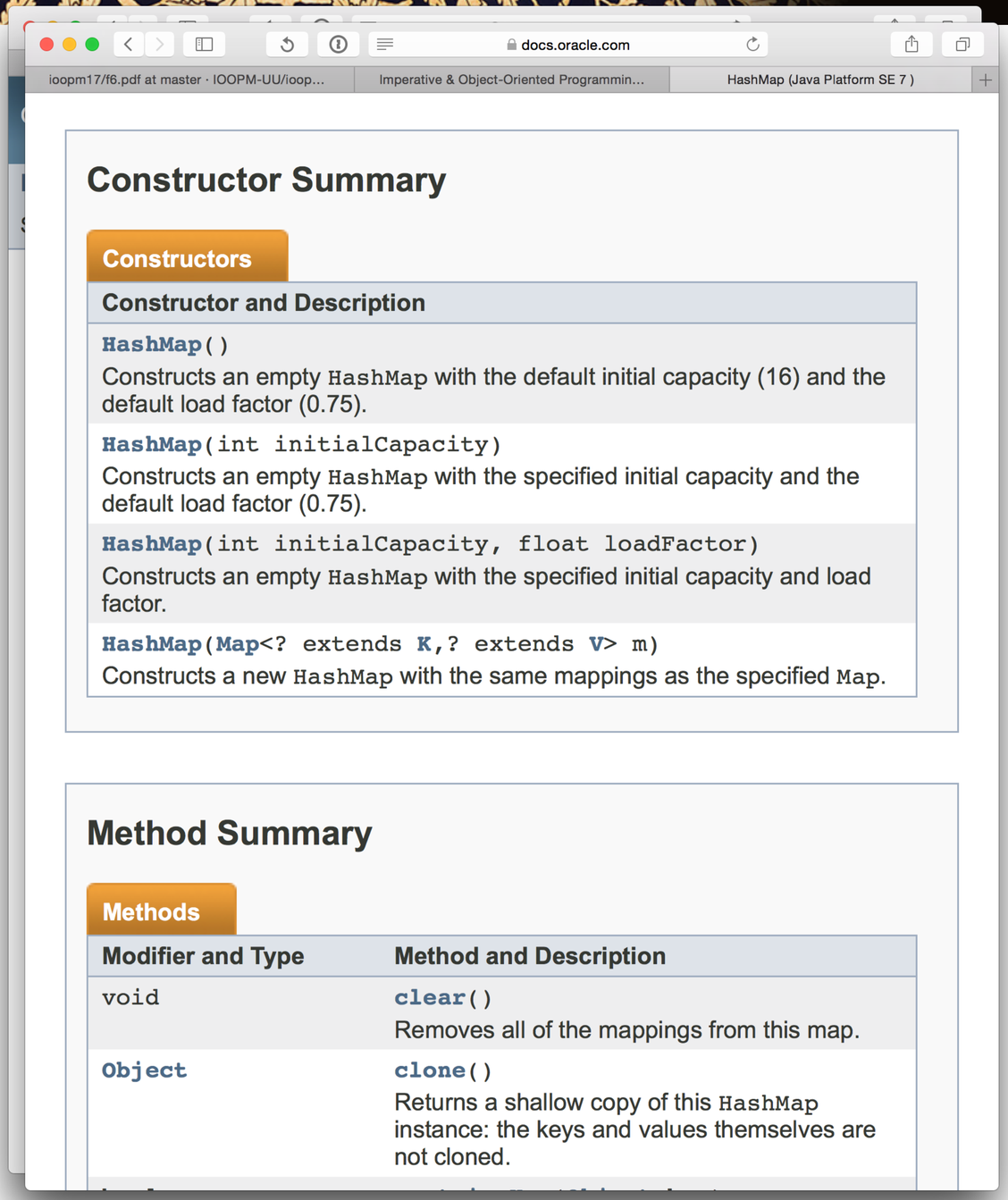
Vanliga datastrukturer, StringTokenizer

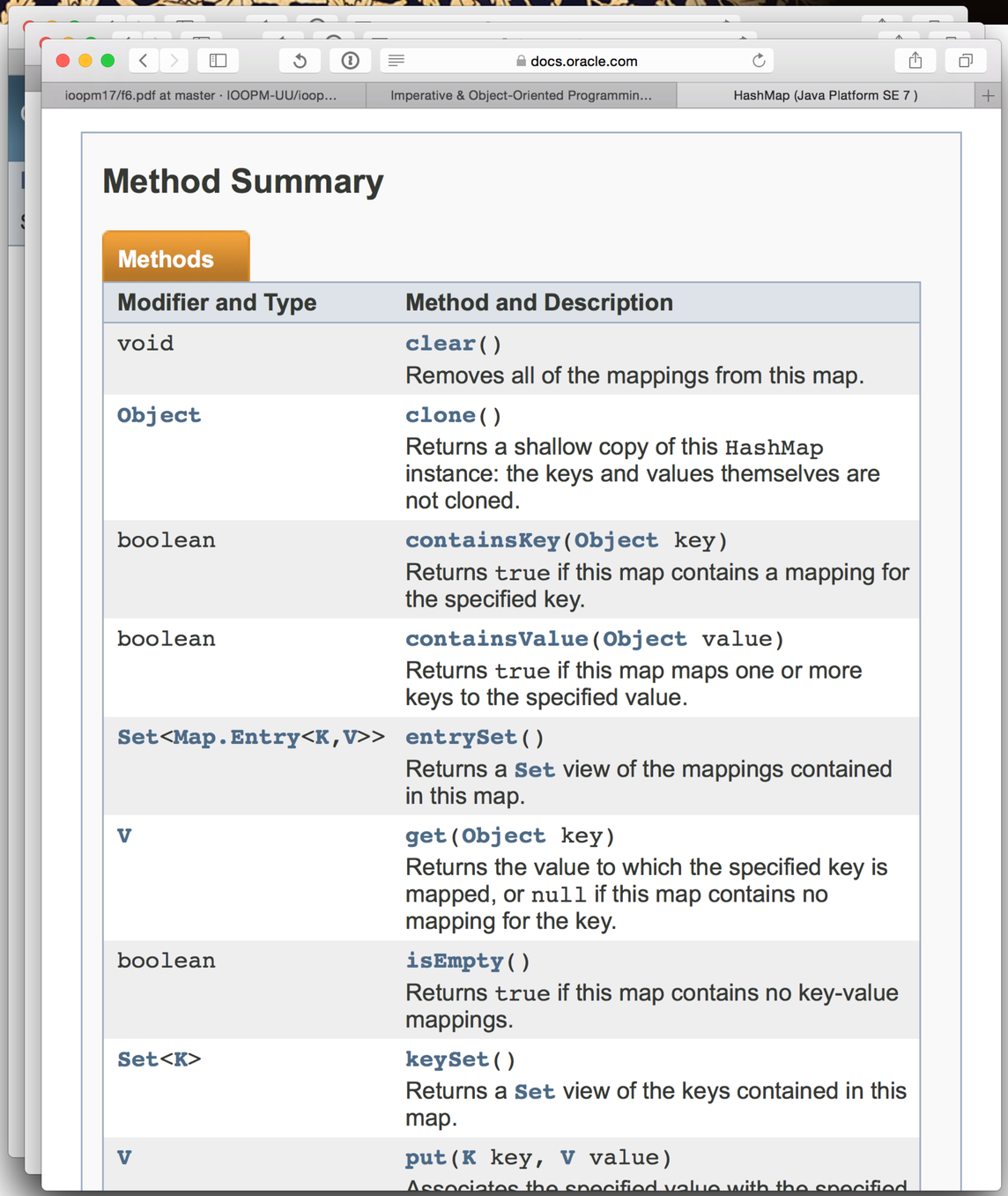
`java.io`

I/O

http://docs.oracle.com/javase/X/docs/api/


docs.oracle.com
ioopm17/f6.pdf at master · IOOPM-UU/ioop...
Imperative & Object-Oriented Programmin...
HashMap (Java Platform SE 7)
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
java.util
Class HashMap<K,V>
java.lang.Object
java.util.AbstractMap<K,V>
java.util.HashMap<K,V>
Type Parameters:
K - the type of keys maintained by this map
V - the type of mapped values
All Implemented Interfaces:
Serializable, Cloneable, Map<K,V>
Direct Known Subclasses:
LinkedHashMap, PrinterStateReasons
public class HashMap<K,V>
extends AbstractMap<K,V>
implements Map<K,V>, Cloneable, Serializable
Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

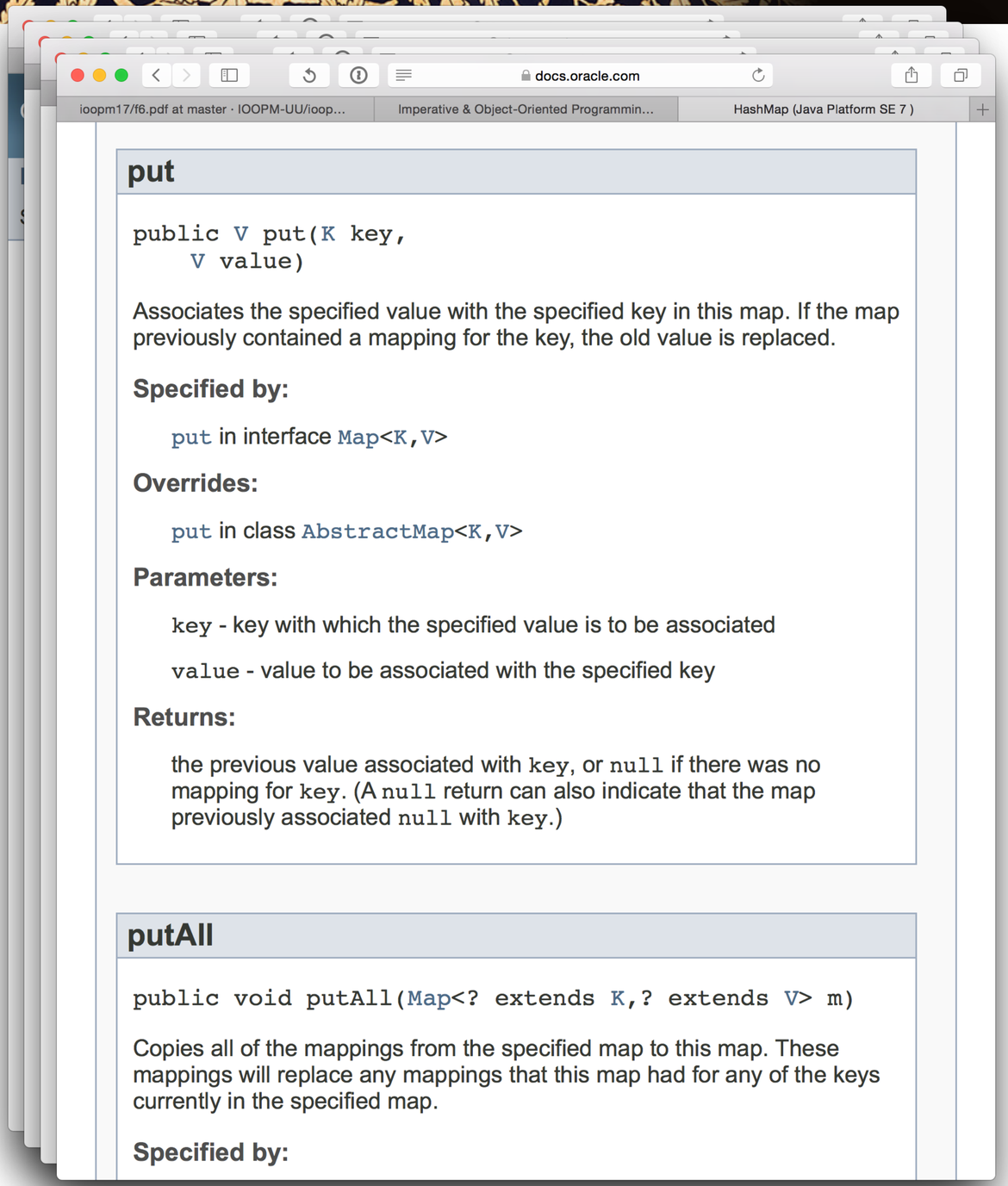

docs.oracle.com
ioopm17/f6.pdf at master · IOOPM-UU/ioop...
Imperative & Object-Oriented Programmin...
HashMap (Java Platform SE 7)
Constructor Summary
Constructors
Constructor and Description
HashMap()
Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
HashMap(int initialCapacity)
Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
HashMap(int initialCapacity, float loadFactor)
Constructs an empty HashMap with the specified initial capacity and load factor.
HashMap(Map<? extends K,? extends V> m)
Constructs a new HashMap with the same mappings as the specified Map.
Method Summary
Methods
Modifier and Type
Method and Description
void
clear()
Removes all of the mappings from this map.
Object
clone()
Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.



Method Summary

Methods

Modifier and Type	Method and Description
`void`	**`clear()`** Removes all of the mappings from this map.
`Object`	**`clone()`** Returns a shallow copy of this `HashMap` instance: the keys and values themselves are not cloned.
`boolean`	**`containsKey(Object key)`** Returns `true` if this map contains a mapping for the specified key.
`boolean`	**`containsValue(Object value)`** Returns `true` if this map maps one or more keys to the specified value.
`Set<Map.Entry<K,V>>`	**`entrySet()`** Returns a **`Set`** view of the mappings contained in this map.
`V`	**`get(Object key)`** Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`boolean`	**`isEmpty()`** Returns `true` if this map contains no key-value mappings.
`Set<K>`	**`keySet()`** Returns a **`Set`** view of the keys contained in this map.
`V`	**`put(K key, V value)`** Associates the specified value with the specified



put

```
public V put(K key,
    V value)
```

Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.

Specified by:

`put` in interface `Map<K,V>`

Overrides:

`put` in class `AbstractMap<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`value` - value to be associated with the specified key

Returns:

the previous value associated with `key`, or `null` if there was no mapping for `key`. (A `null` return can also indicate that the map previously associated `null` with `key`.)

putAll

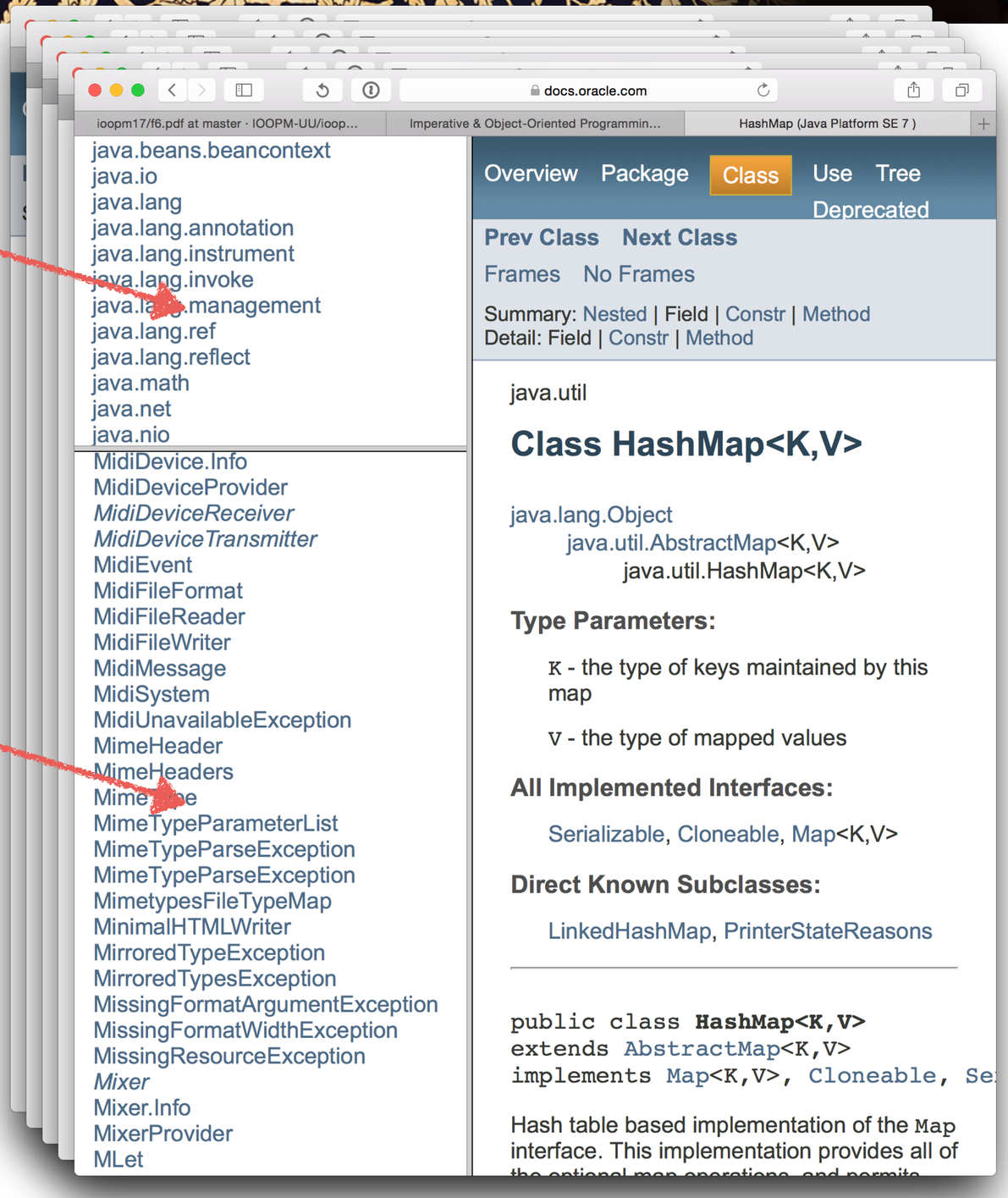
```
public void putAll(Map<? extends K,? extends V> m)
```

Copies all of the mappings from the specified map to this map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map.

Specified by:

Hundratals paket

Tusentals klasser



Stark typning

Java är starkt typat (C *svagt*!)

Vi kan inte behandla en typ som en annan

Försök att göra så genererar ett tydligt fel under körning

```
/// Type punning in C
elem_t e; /// unknown content
e.int_value = 42;
float f = e.float_value; ///???
```

```
/// Type casting in C is abusive
void *ptr = (void *)42;
int i = (int) ptr;
```

```
/// Bad type cast generates runtime error
Object o = new Object();
Person p = (Person) o; /// Compiles
```

ClassCastException

Parametrisk Polymorfism

```
/// Revisiting previous examples – and improving them!
Person p1 = new Person();
Class<Person> cp = p1.getClass();
Person p2 = cp.newInstance();
```

```
/// Revisiting previous examples – and improving them!
LinkedList<Person> list = new LinkedList<>(); /// !!
list.add(new Object()); /// Will not compile
Person p = list.first();
```

```
/// Revisiting previous examples – and improving them!
public class Person implements Comparable<Person> {
    ...
}
```

Inga Segfaults

Avrefererad `null`-pekare i Java:

```
Exception in thread "main" java.lang.NullPointerException
        at com.example.myproject.Book.getTitle(Book.java:16)
        at com.example.myproject.Author.getBookTitles(Author.java:25)
        at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

Uppslagning i array med `index < 0` eller `index >= array.length`

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at com.example.myproject.BookStore.getBook(BookStore.java:52)
        at com.example.myproject.BookStore.getLatest(BookStore.java:33)
        at com.example.myproject.Bootstrap.main(Bootstrap.java:17)
```

Notera att programmet skriver ut vilken rad det kraschade på!

Exception Handling

```
/// Bad type cast generates runtime error
BufferedReader in = null;
try {
    in = new BufferedReader(new FileReader("foo.in"));
    while (true) {
        node = node.next;
    }
    ...
} catch (NullPointerException e) {
    /// Went to far in the list!
    e.printStackTrace(System.err);
} finally {
    if (in != null) {
        in.close();
    }
}
```

Referenser

- Ingen pekararitmetik
- En referens kan inte skapas ur tomma intet
- Inga dangling pointers
- En **pekare är en adress** — ett heltal — ett offset från 0
- En **referens är ett handtag**, en token genom vars försorg jag kan accessa ett objekt
- Ofta säger vi pekare ändå, att de är referenser är uppenbart av kontexten
- En `null`-pekare är inte någon referens, utan är **avsaknaden av en referens**

Jshell

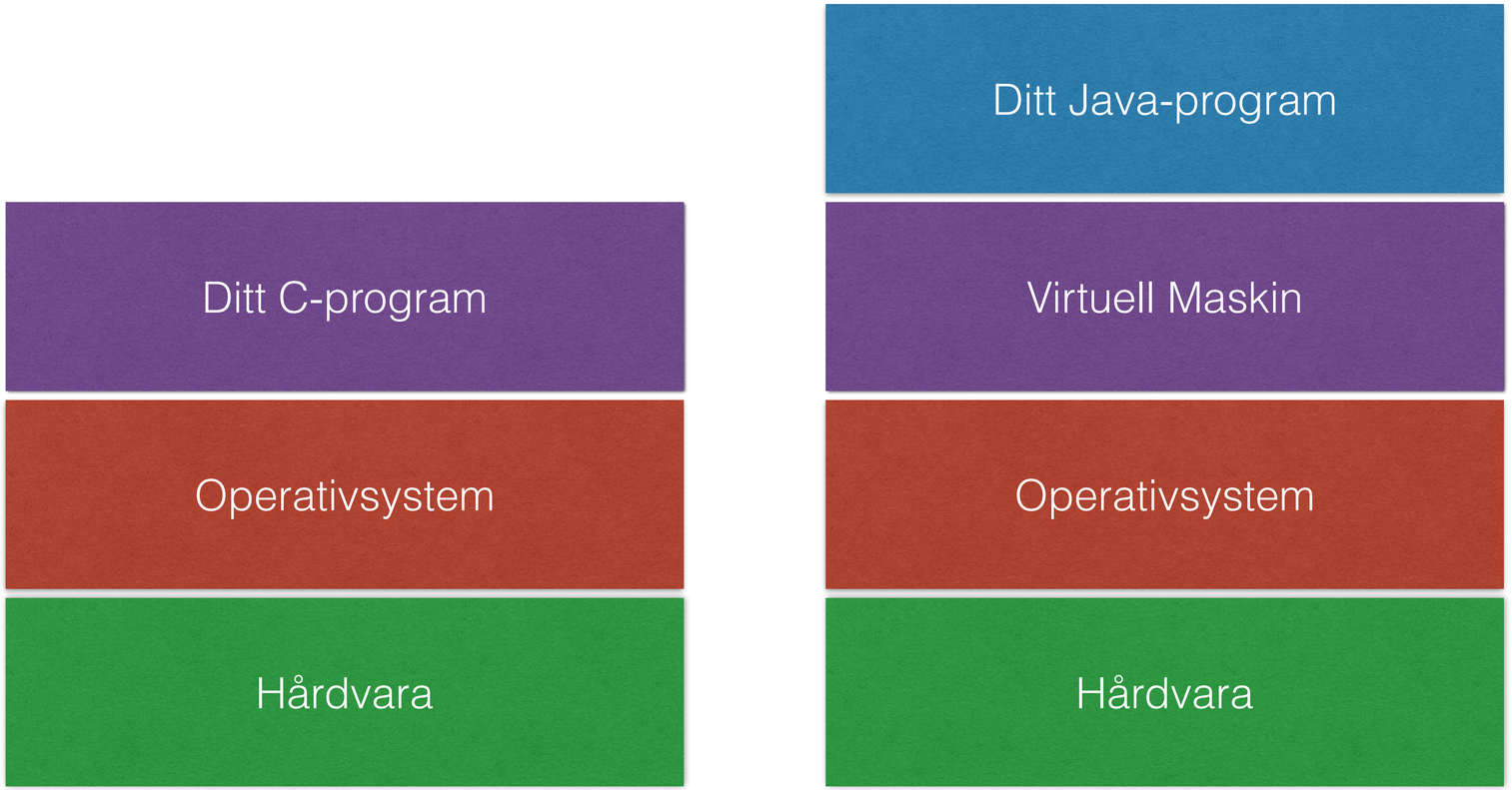
- Från och med JDK 9 har Java äntligen fått en REPL (Read-Eval-Print Loop)
- Lek med Java i en interaktiv, levande miljö

```
[writo649@trygger:6 ~]$ jshell
|  Welcome to JShell -- Version 10.0.2
|  For an introduction type: /help intro

jshell> 42 + 42
$1 ==> 84

jshell> public class Test { publict Test(int i) { this.i = i; } int i; }
|  Error:
|  cannot find symbol
|    symbol:   class publict
|  public class Test { publict Test(int i) { this.i = i; } int i; }
|                      ^-----^
|  Error:
|  missing return statement
|  public class Test { publict Test(int i) { this.i = i; } int i; }
|                                            ^-------------^

jshell> public class Test { public Test(int i) { this.i = i; } int i; }
|  created class Test

jshell> Test t = new Test()
|  Error:
|  constructor Test in class Test cannot be applied to given types;
|    required: int
|    found: no arguments
|    reason: actual and formal argument lists differ in length
|  Test t = new Test();
|           ^----^

jshell> Test t = new Test(42)
t ==> Test@5e3a8624
```

Objektorientering?

Förhoppningsvis kommer du också att uppskatta objektorientering…

Stacken i Java vs. Stacken i C



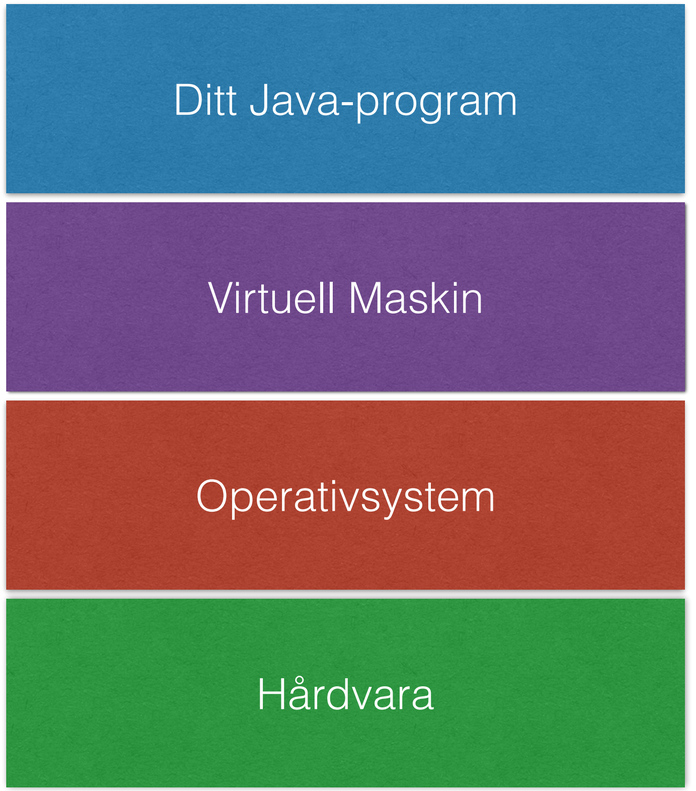
Kompilera och köra ett Java-program

```
$ javac MyProg.java
```

Skapar en eller flera .class-filer varav en heter MyProg.class

```
$ java MyProg
```

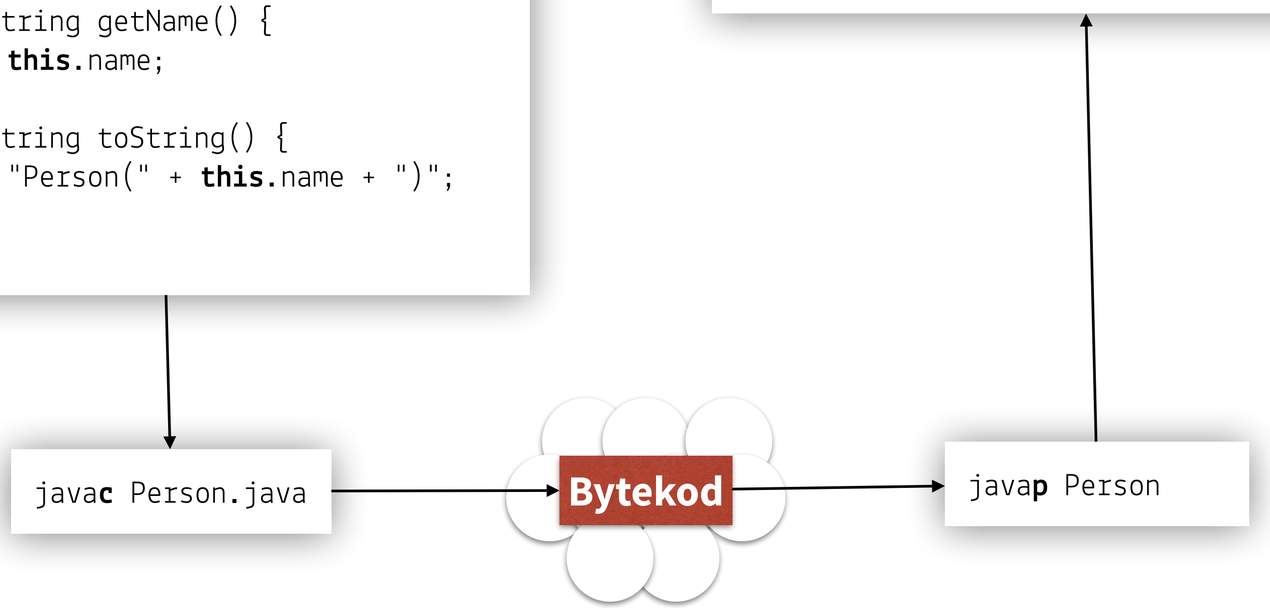
Startar den virtuella maskinen och laddar in MyProg och kör



Kompilera ditt Java-program

```
public class Person {
    public String name;
    public Person(String name) {
      assert name != null : "Name == null!";
      this.name = name;
    }
    public void setName(String name) {
      this.name = name;
    }
    public String getName() {
      return this.name;
    }
    public String toString() {
      return "Person(" + this.name + ")";
    }
}
```

```
public class Person {
  public java.lang.String name;
  static final boolean $assertionsDisabled;
  public Person(java.lang.String);
  public void setName(java.lang.String);
  public java.lang.String getName();
  public java.lang.String toString();
  static {};
}
```



Use the source, Luke!

```
javap -c Person
```

```
Compiled from "Person.java"
public class Person {
  public java.lang.String name;

  static final boolean $assertionsDisabled;

  public Person(java.lang.String);
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: getstatic     #2                  // Field $assertionsDisabled:Z
       7: ifne          24
      10: aload_1
      11: ifnonnull     24
      14: new           #3                  // class java/lang/AssertionError
      17: dup
      18: ldc           #4                  // String Name must not be null!
      20: invokespecial #5                  // Method java/lang/AssertionError."<init>":(Ljava/lang/Object;)V
      23: athrow
      24: aload_0
      25: aload_1
      26: putfield      #6                  // Field name:Ljava/lang/String;
      29: return

  public void setName(java.lang.String);
    Code:
       0: aload_0
       1: aload_1
```

Use the s

```
    24: aload_0
    25: aload_1
    26: putfield      #6                  // Field name:Ljava/lang/String;
    29: return

public void setName(java.lang.String);
  Code:
     0: aload_0
     1: aload_1
     2: putfield      #6                  // Field name:Ljava/lang/String;
     5: return

public java.lang.String getName();
  Code:
     0: aload_0
     1: getfield      #6                  // Field name:Ljava/lang/String;
     4: areturn

public java.lang.String toString();
  Code:
     0: new           #7                  // class java/lang/StringBuilder
     3: dup
     4: invokespecial #8                  // Method java/lang/StringBuilder."<init>":()V
     7: ldc           #9                  // String Person(
     9: invokevirtual #10                 // Method java/lang/StringBuilder.append:(Ljava/lang/
                                          //                         String;)Ljava/lang/StringBuilder;
    12: aload_0
    13: getfield      #6                  // Field name:Ljava/lang/String;
    16: invokevirtual #10                 // Method java/lang/StringBuilder.append:(Ljava/lang/
                                          //                         String;)Ljava/lang/StringBuilder;
    19: ldc           #11                 // String )
    21: invokevirtual #10                 // Method java/lang/StringBuilder.append:(Ljava/lang/
                                          //                         String;)Ljava/lang/StringBuilder;
    24: invokevirtual #12                 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
    27: areturn

static {};
  Code:
```

```
public class Person {
    public String name;
    public Person(String name) {
      assert name != null : "Name == null!";
      this.name = name;
    }
    public void setName(String name) {
      this.name = name;
    }
    public String getName() {
      return this.name;
    }
    public String toString() {
      return "Person(" + this.name + ")";
    }
}
```

Automatisk skräpsamling

- Mål: att ge programmeraren en illusion att minnet är oändligt
- Metod: identifiera *skräp-data* och frigör det **automatiskt**
- Definitionen av skräp: data som inte kan nås av programmet
- Mer formellt: objektet O är skräp om det inte finns någon väg i minnesgrafen från något rot (variabeln på stacken, globala variabler, o.dyl.) till O
- Två grundläggande sätt att göra automatisk skräpsamling:

  Referensräkning

  Tracing

Referensräkning

Grundläggande idé: varje objekt sparar information om hur många som pekar till det

När denna räknare når 0 — ta bort objektet

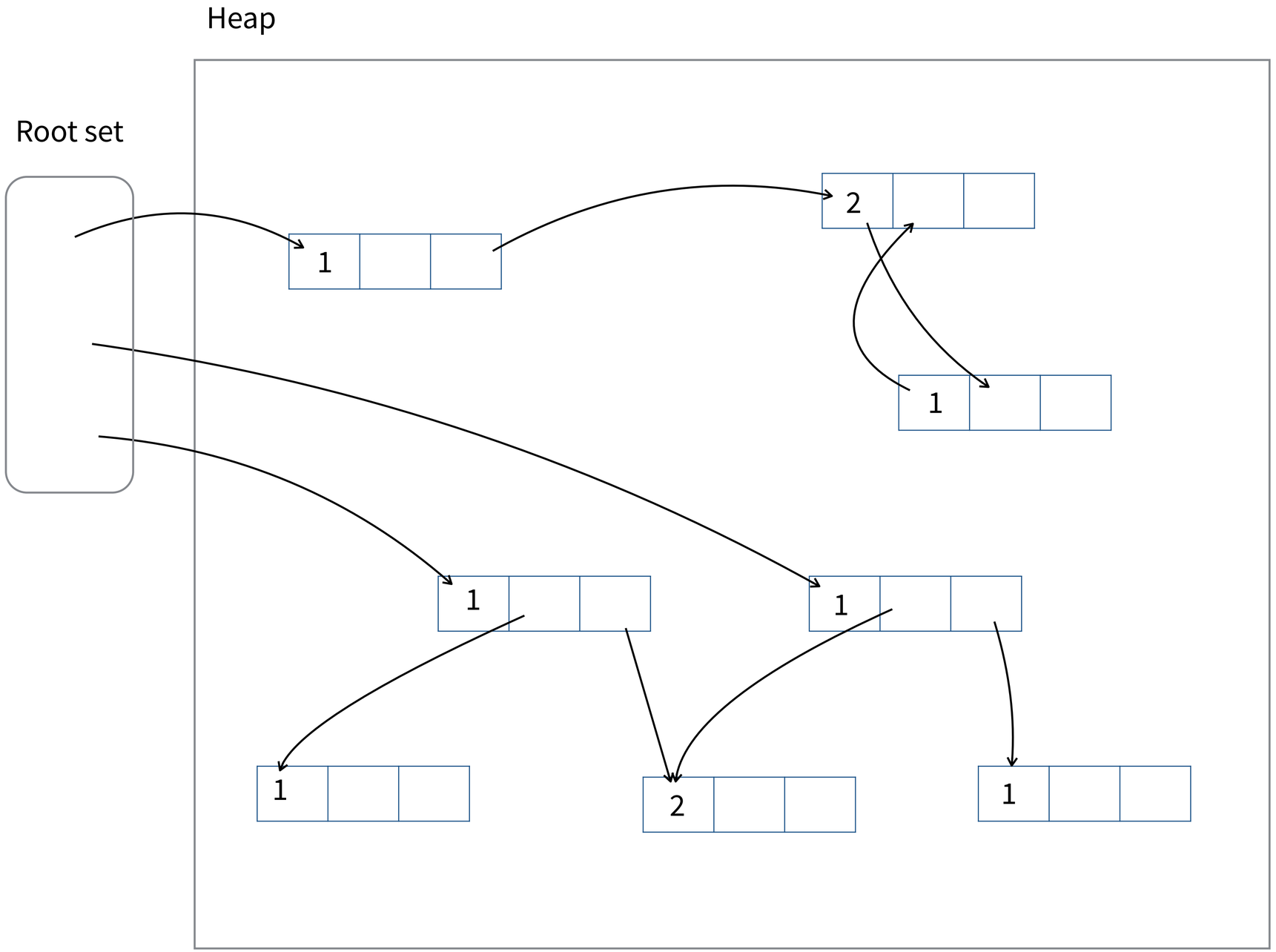
Varje gång en referens skapas/tas bort, manipulera referensräknaren:

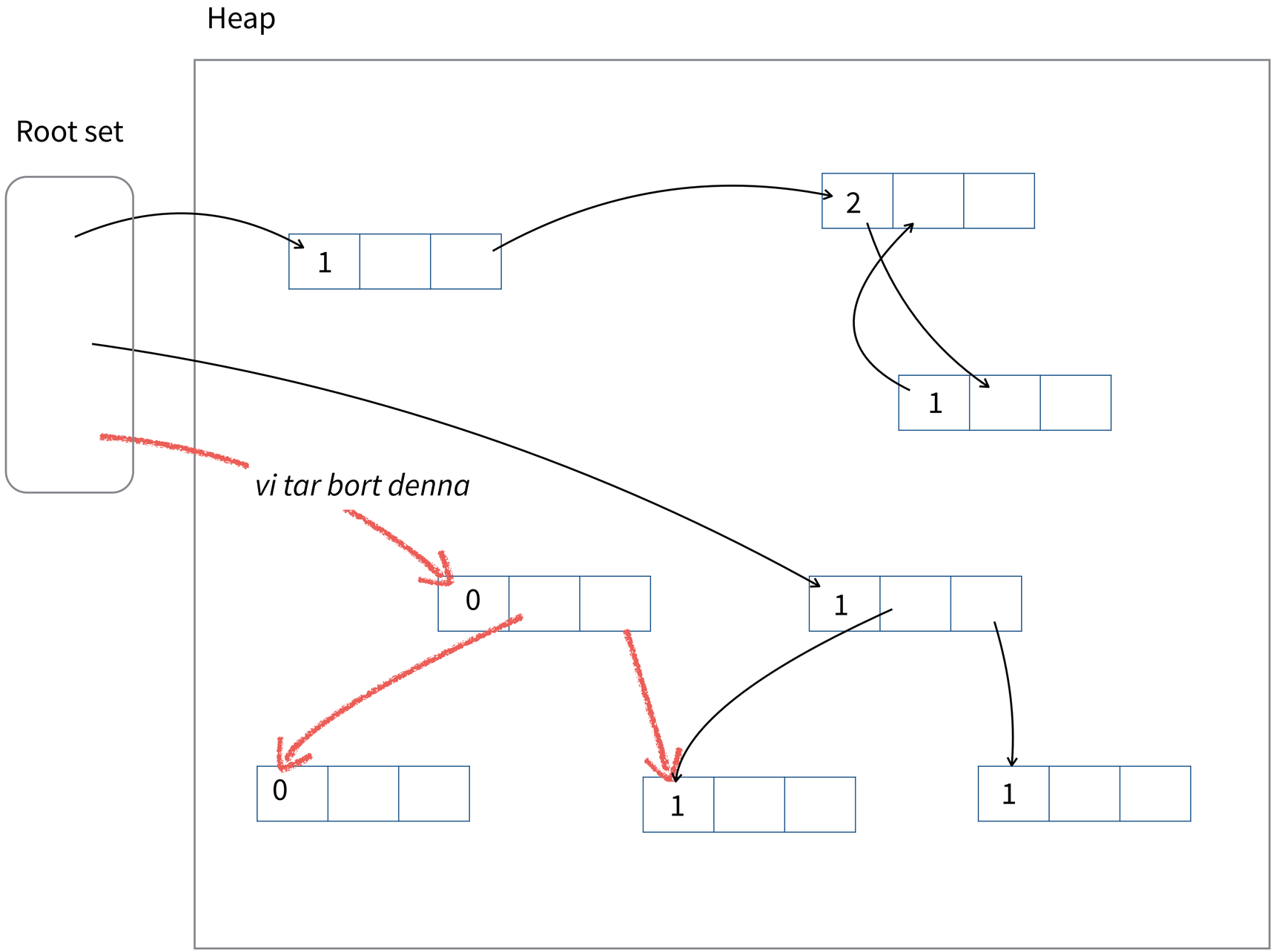
```
void *p = malloc(2048); // refcount 1
void *x = p; // refcount 2
p = NULL; // refcount 1
x = NULL; // refcount 0, free(x)
```

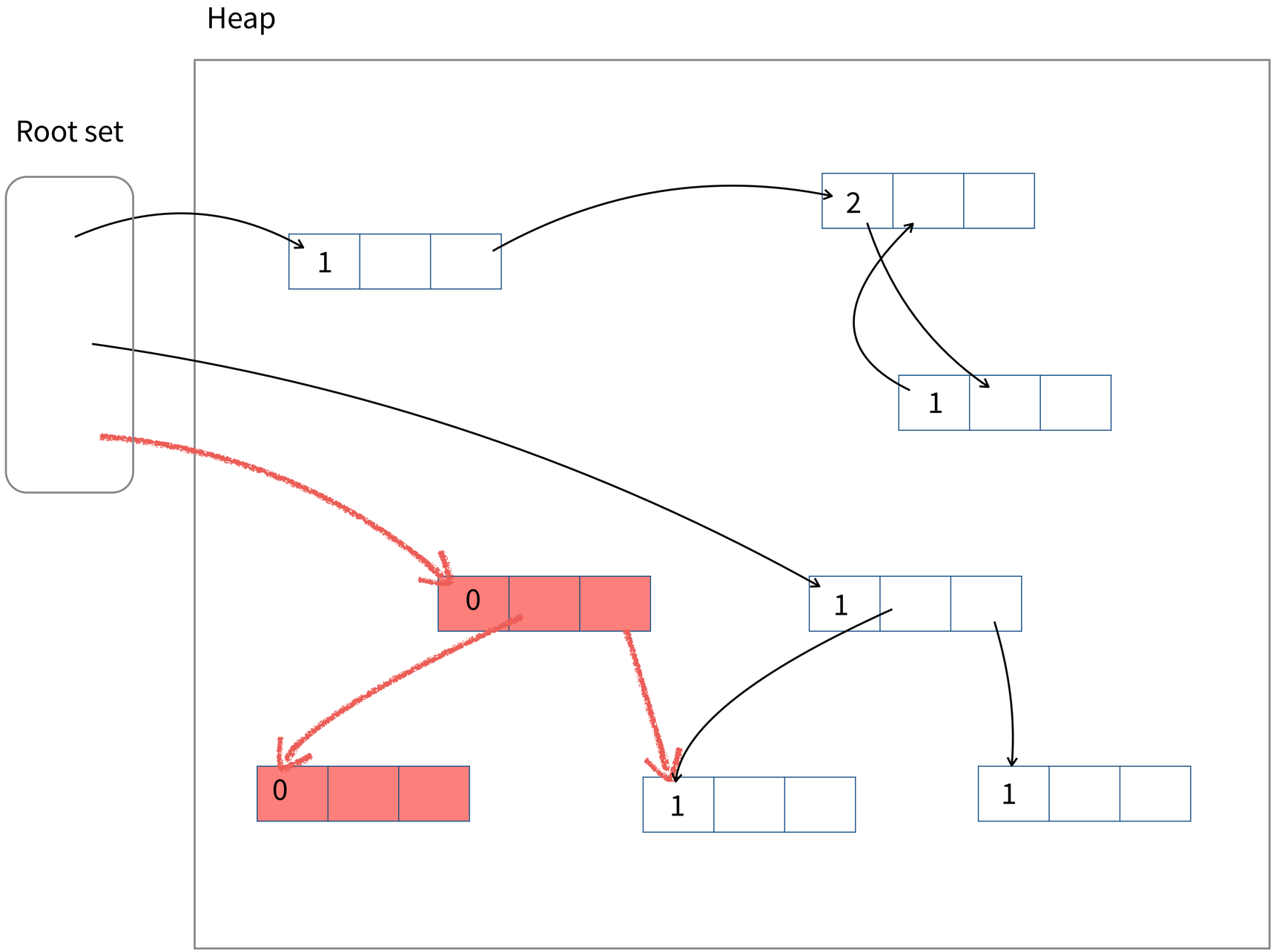
Problem:

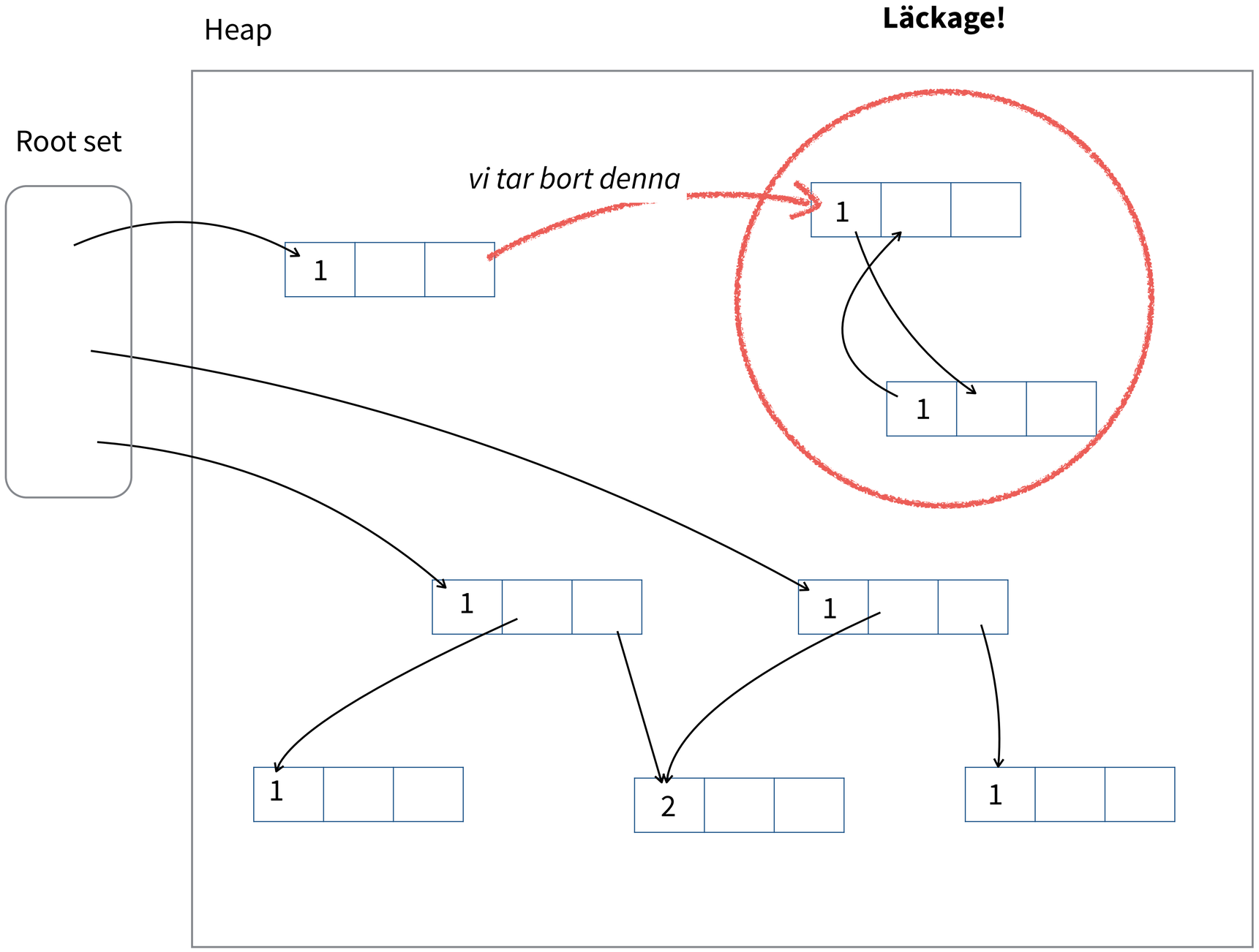
Cykliska strukturer (se nästföljande sidor)

Långlivat minne som manipuleras ofta kostar, fast vi aldrig tar bort det


Heap
Root set
1
2
1
1
1
1
2
1


Heap
Root set
1
2
1
vi tar bort denna
0
1
0
1
1


Heap
Root set
1
2
1
0
1
0
1
1

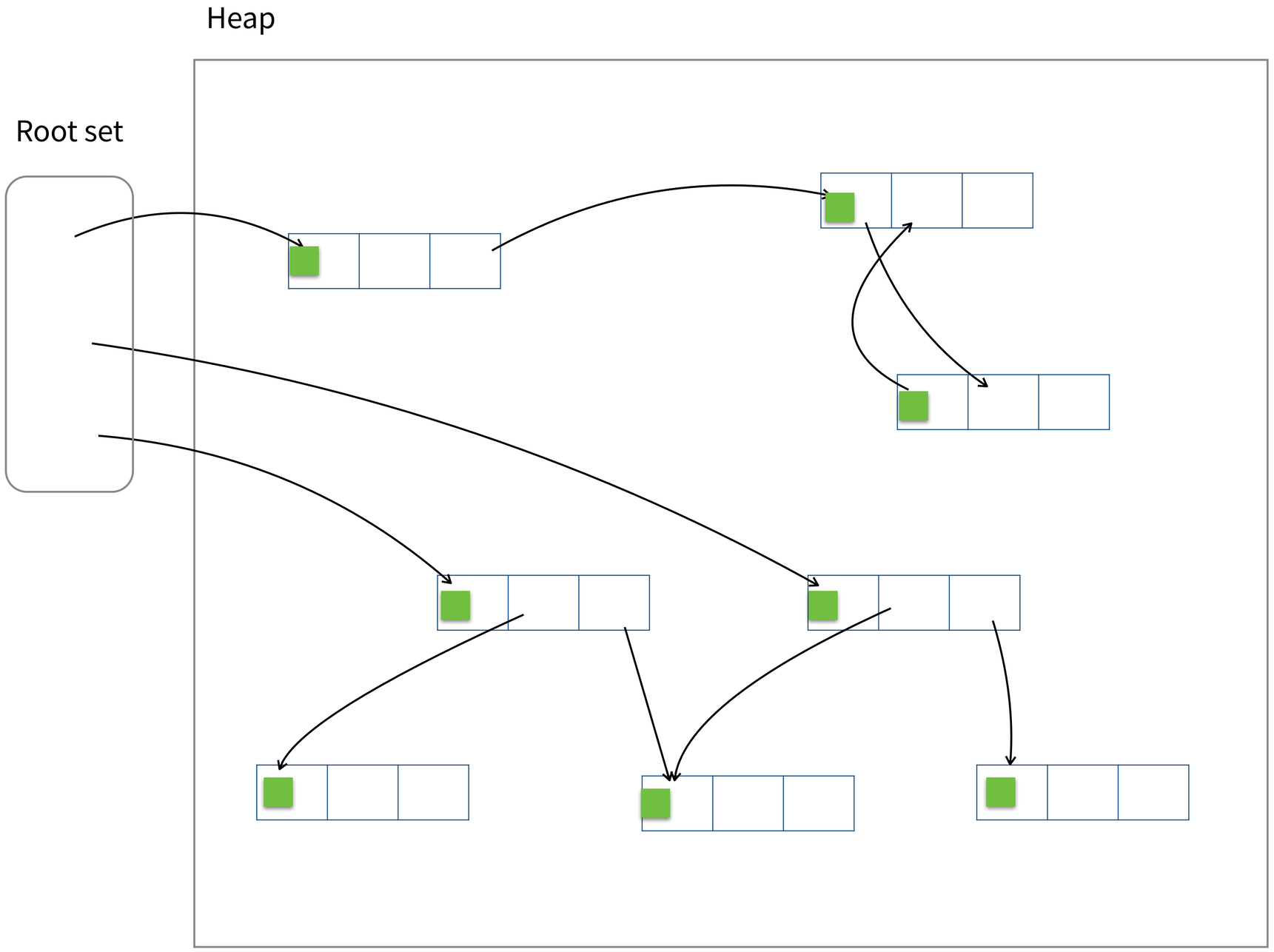

Heap
Läckage!
Root set
vi tar bort denna
1
1
1
1
1
1
2
1

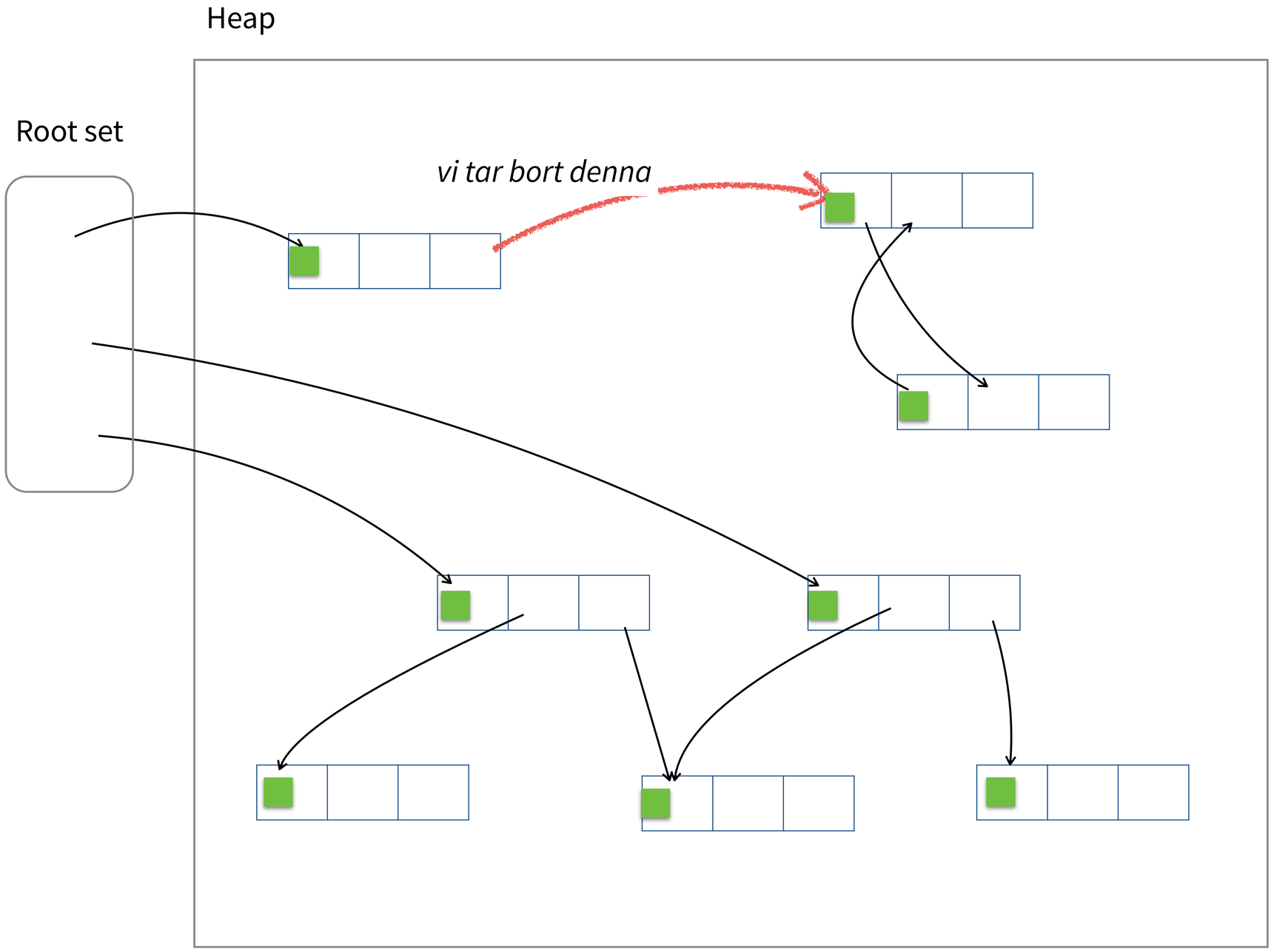
Tracing GC: Mark-Sweep

När minnet tar slut:

1. Följ rötterna och markera alla objekt som kan nås
2. Iterera över alla objekt och frigör alla som inte markerats i 1.

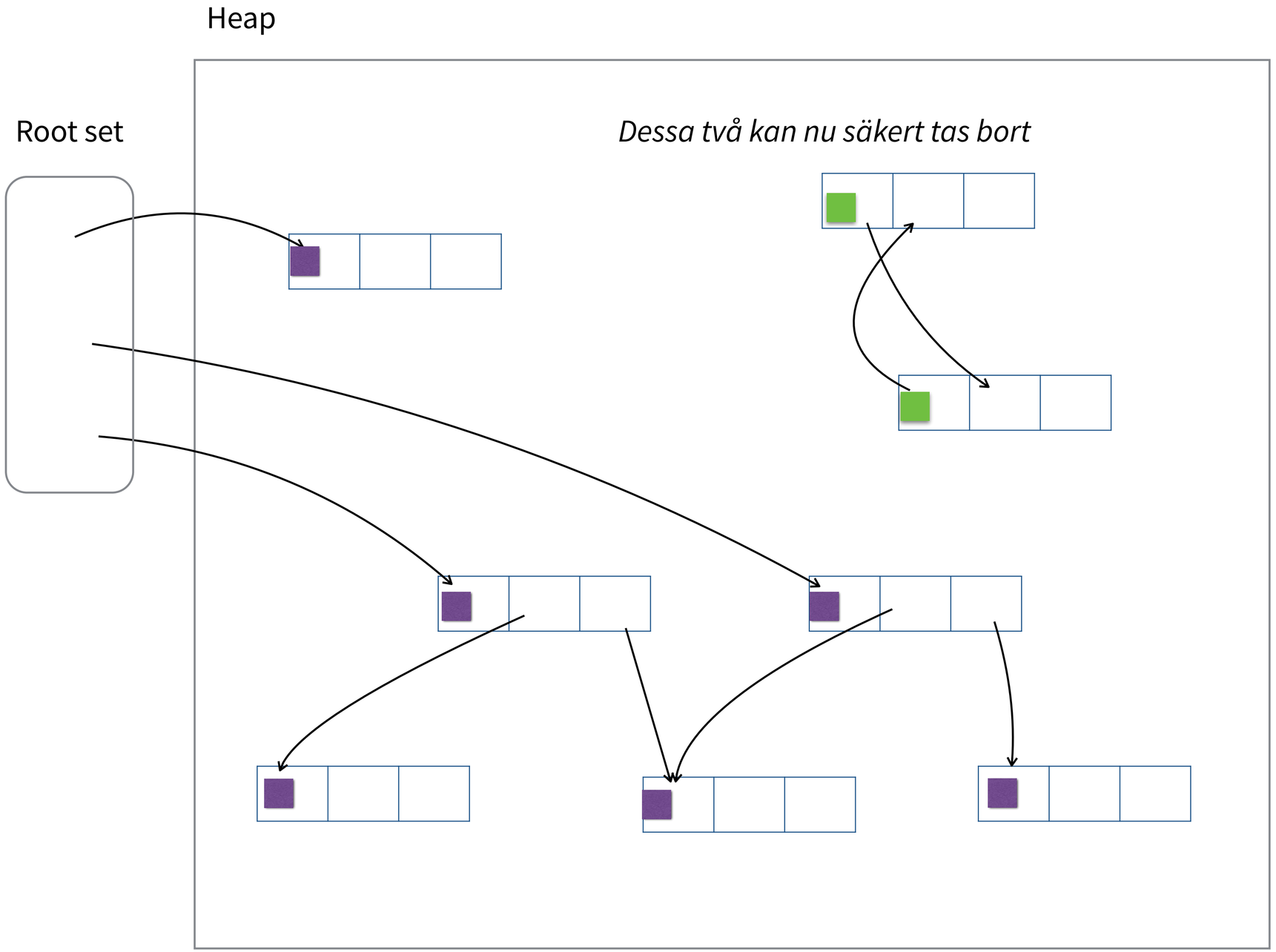
Nästa bild visar markering i *mark-fasen (1.)*


Heap
Root set


Heap
Root set
vi tar bort denna

I nästa mark-fas markerar vi med annan färg

Om något är grönt fortfarande efter denna fas är det skräp och skall tas bort


Heap
Root set
Dessa två kan nu säkert tas bort