TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur Softwaretechnik
Prof. Dr.-Ing. Steffen Becker

Design pattern
Iterator

Design Pattern Iterator

Problem

- A datastructure (e.g. a list) should be accessible/enumerable without knowing the structure‘s implementation
- Multiple accesses should be possible at a single point in time

Solution strategy

- „Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.“

Pattern type

- Behavioural pattern

Design Pattern
Iterator II

- Separate data structure class from the iteration behaviour
- The structure should offer multiple iterator instances
- The structure is able to create its iterator instances itself

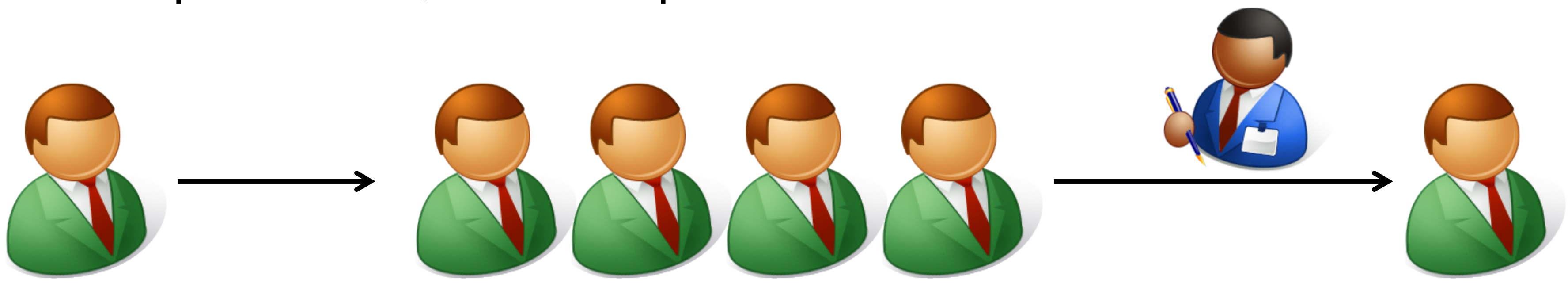
- Same principle as before: Separate implementation of data structure from iterator implementation and its operations

More complex example...

Datastructure Queue

Stores data according to FIFO principle

- Example
  - Queue at supermarkt cash lines
  - Enque at the end, serve the topmost

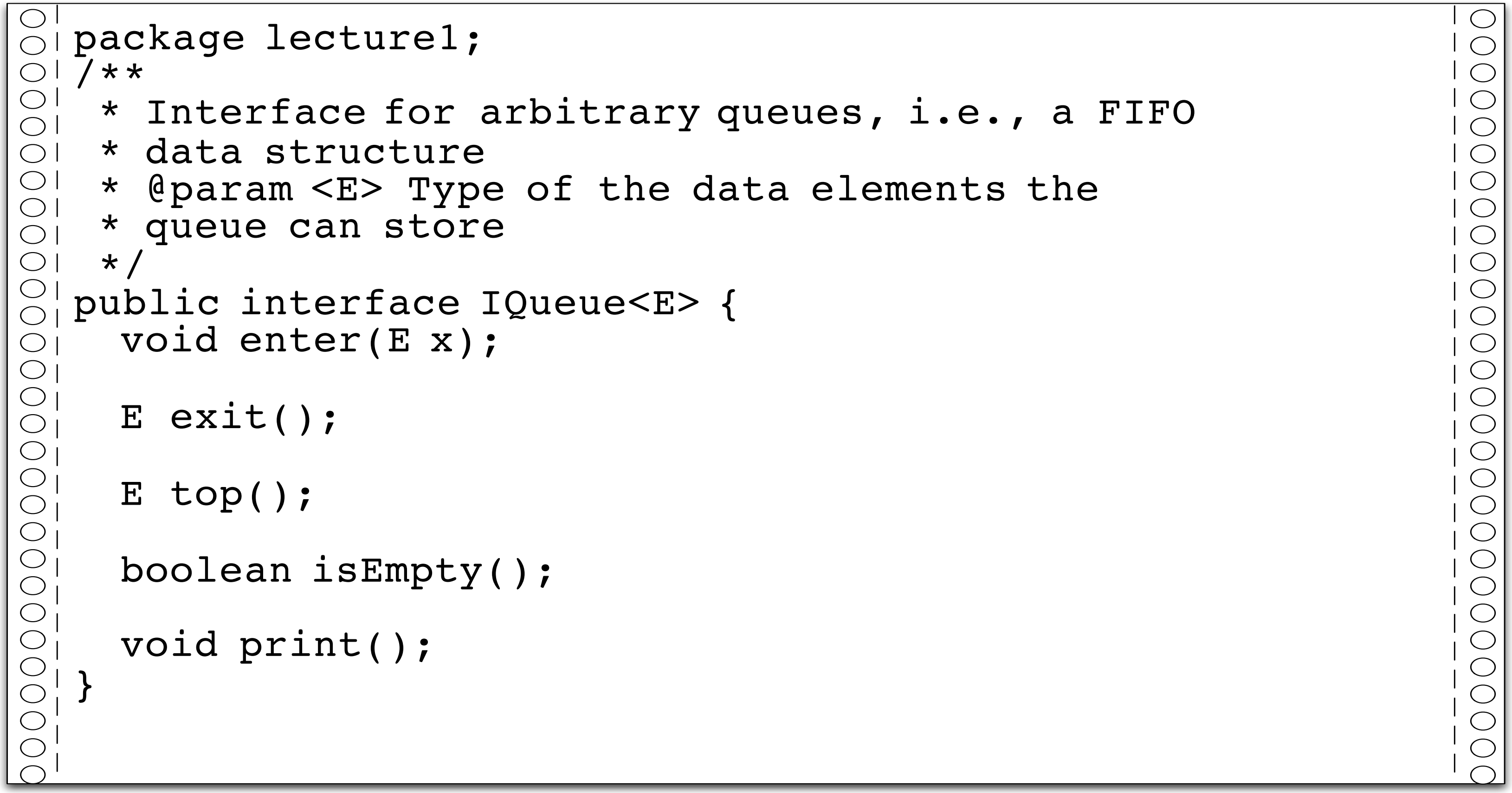


Operations

- Operations on queues
  - enter – Enque an element at the end
  - exit – Remove front element
  - top – Which one is the top element?
  - isEmpty – Is the queue empty?
  - print – Display queue contents

- Goal: Define a generic datastructure

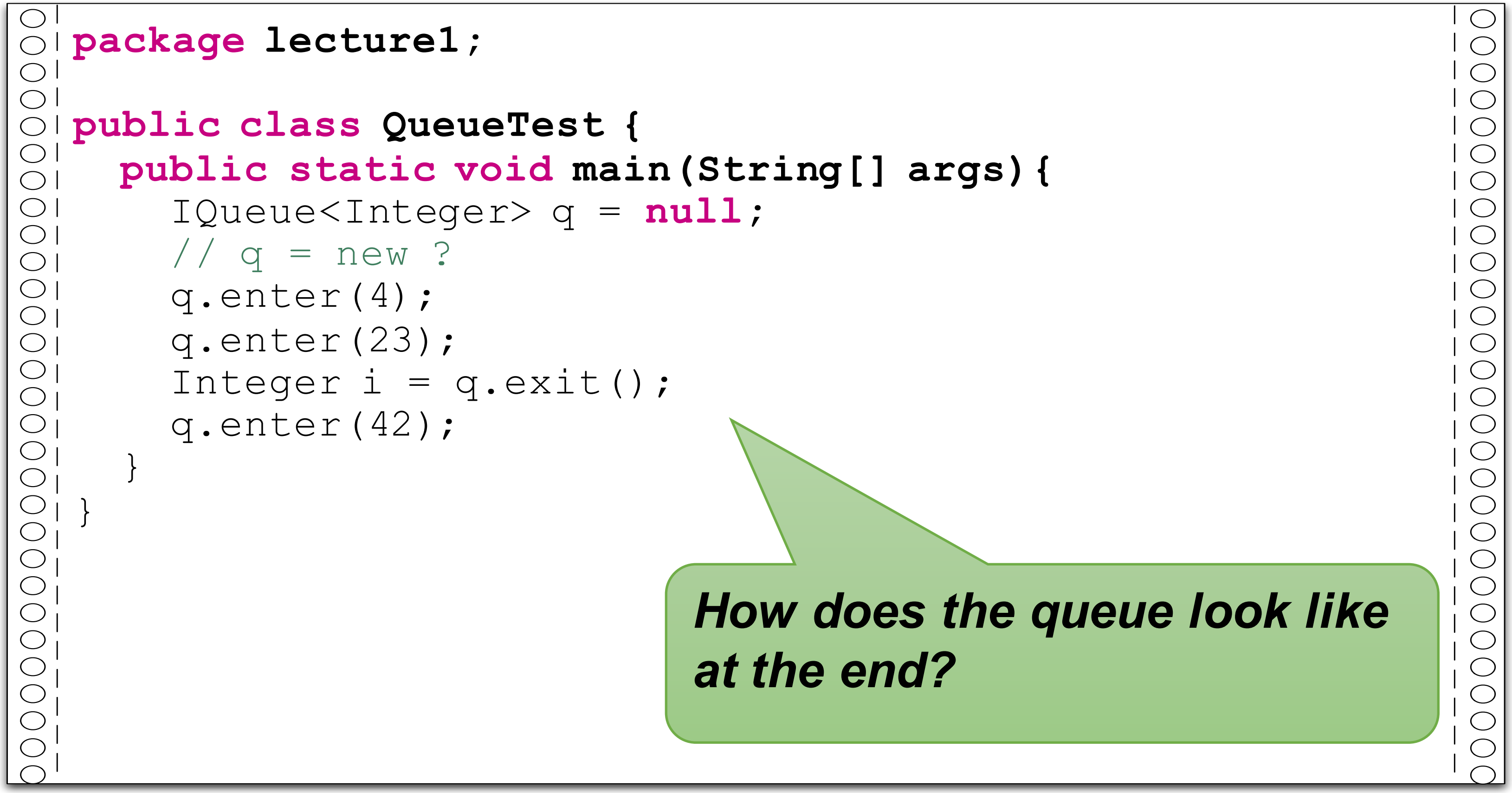
Interface Queue for generic elements



```
package lecture1;
/**
 * Interface for arbitrary queues, i.e., a FIFO
 * data structure
 * @param <E> Type of the data elements the
 * queue can store
 */
public interface IQueue<E> {
  void enter(E x);

  E exit();

  E top();

  boolean isEmpty();

  void print();
}
```

IQueue.java

Verwendung der Queue



```java
package lecture1;

public class QueueTest {
  public static void main(String[] args){
    IQueue<Integer> q = null;
    // q = new ?
    q.enter(4);
    q.enter(23);
    Integer i = q.exit();
    q.enter(42);
  }
}
```

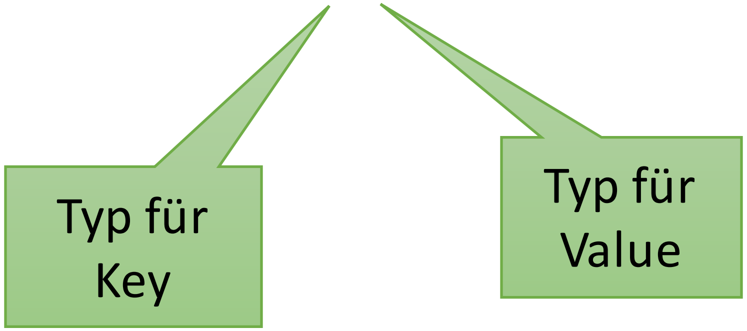
How does the queue look like at the end?

QueueTest.java

Implementierung der Queue

- Based on a HashMap (e.g. could also use ArrayList) http://download.oracle.com/javase/6/docs/api/

- Is generic
- HashMap<K,V>



Typ für Key

Typ für Value

- When using it for Queue<E> what should be K and what V?

Java Realisation contd.

```java
package lecture1;

import java.util.HashMap;

class HashQueue<E> implements IQueue<E> {
   HashMap<Integer,E> h = new
      HashMap<Integer,E>();
   /** Position of first Element
   */
   int firstElement = 0;
   /** Element count
   */
   int noOfElements = 0;
   public void enter (E x) {
     h.put(new
      Integer(firstElement+noOfElements),
      x);
     noOfElements++;
   }
```

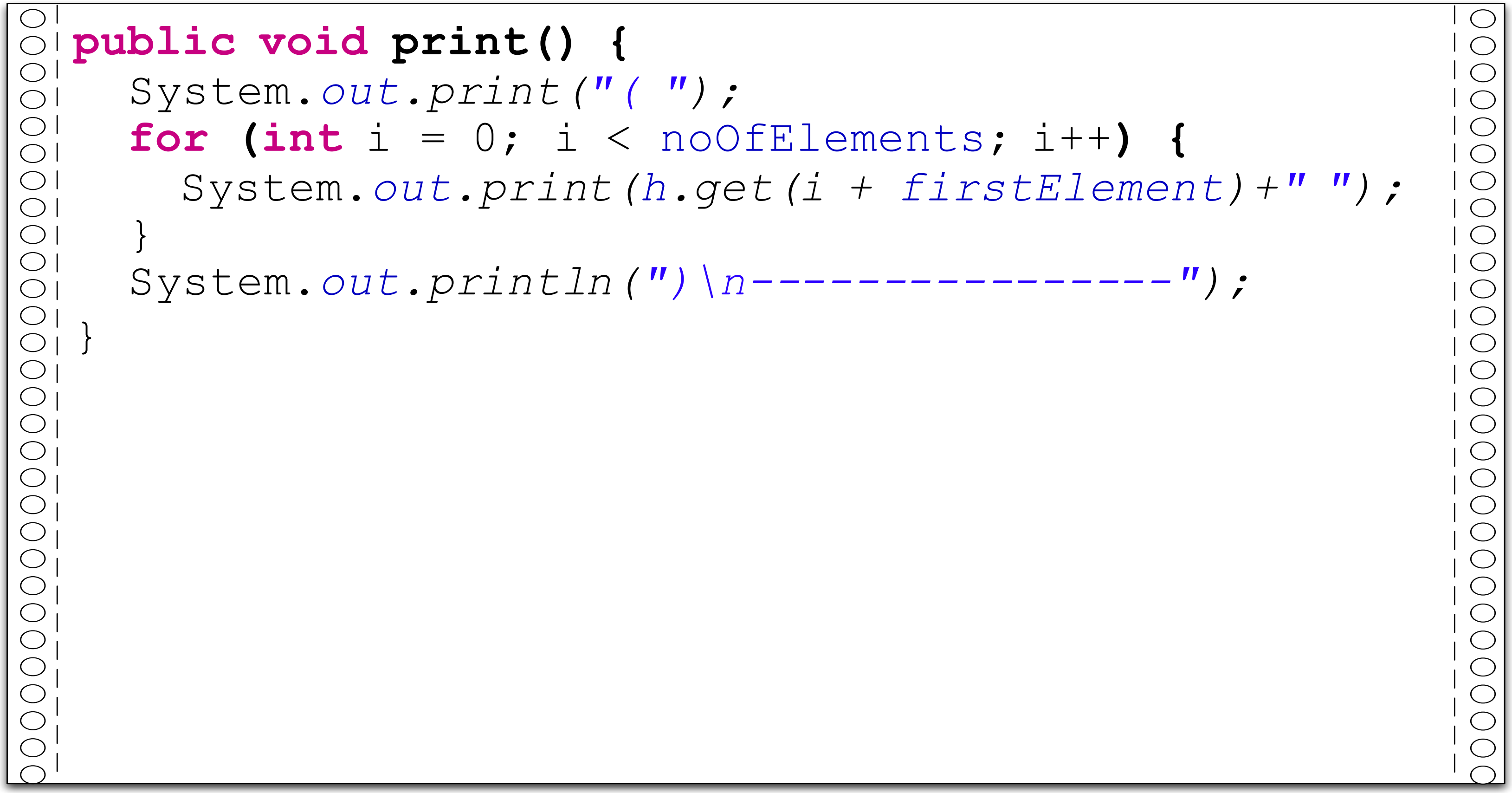
HashQueue.
java

Exit, top, isEmpty

```
  public E exit() {
    E elem = h.remove(firstElement);
    noOfElements--;
    firstElement++;
    return elem;
  }
  public E top () {
    return h.get(firstElement);
  }
  public boolean isEmpty() {
    return noOfElements == 0;
  }
```

HashQueue.java

Print operation



```
public void print() {
  System.out.print("( ");
  for (int i = 0; i < noOfElements; i++) {
    System.out.print(h.get(i + firstElement)+" ");
  }
  System.out.println(")\n----------------");
}
```

QueueTest.java

Adding the iterator

- Interface Iterator<E>
- Three methods
  - `boolean hasNext()`
  - `E next()`
  - `void remove()`
- Enumerate elements of a datastructure

Generic Interface, too

HashQueue mit Iterator

- Implement Iterator-Interface on HashQueue example

```
class HashQueue<E> implements IQueue<E>, Iterable<E> {
…}
```

- A hash queue object should be able to return an iterator which can iterate the current queue elements
- Java Interface

```
Iterable<T> {
  /**
  Returns an iterator over a set of elements of type T.
  @return an Iterator.
  */
  Iterator<T> iterator();
}
```

HashQueue with Iterator II

- Idea
  - New iterator uses a position marker on first element and runs until firstElement+noOfElements-1
  - ⇒ Iterator is closely coupled to internals of HashQueue!
- Consequence:
  - Use an inner class for the iterator implementation (or a friend class in C++, etc.)
  - Retains the encapsulation

Excursion
Inner classes

Example

- Modelling a bank account together with ist operations (withdrawl, transfer, etc.): A class for account, another for actions
- Actions should be tightly bound to account objects (no action without associated account)

Idea

- Encapsulate class action in class account
- It becomes an inner class

In Java:

- Put inner classes in other classes similar to member variables

Beispiel
Konto und Aktion

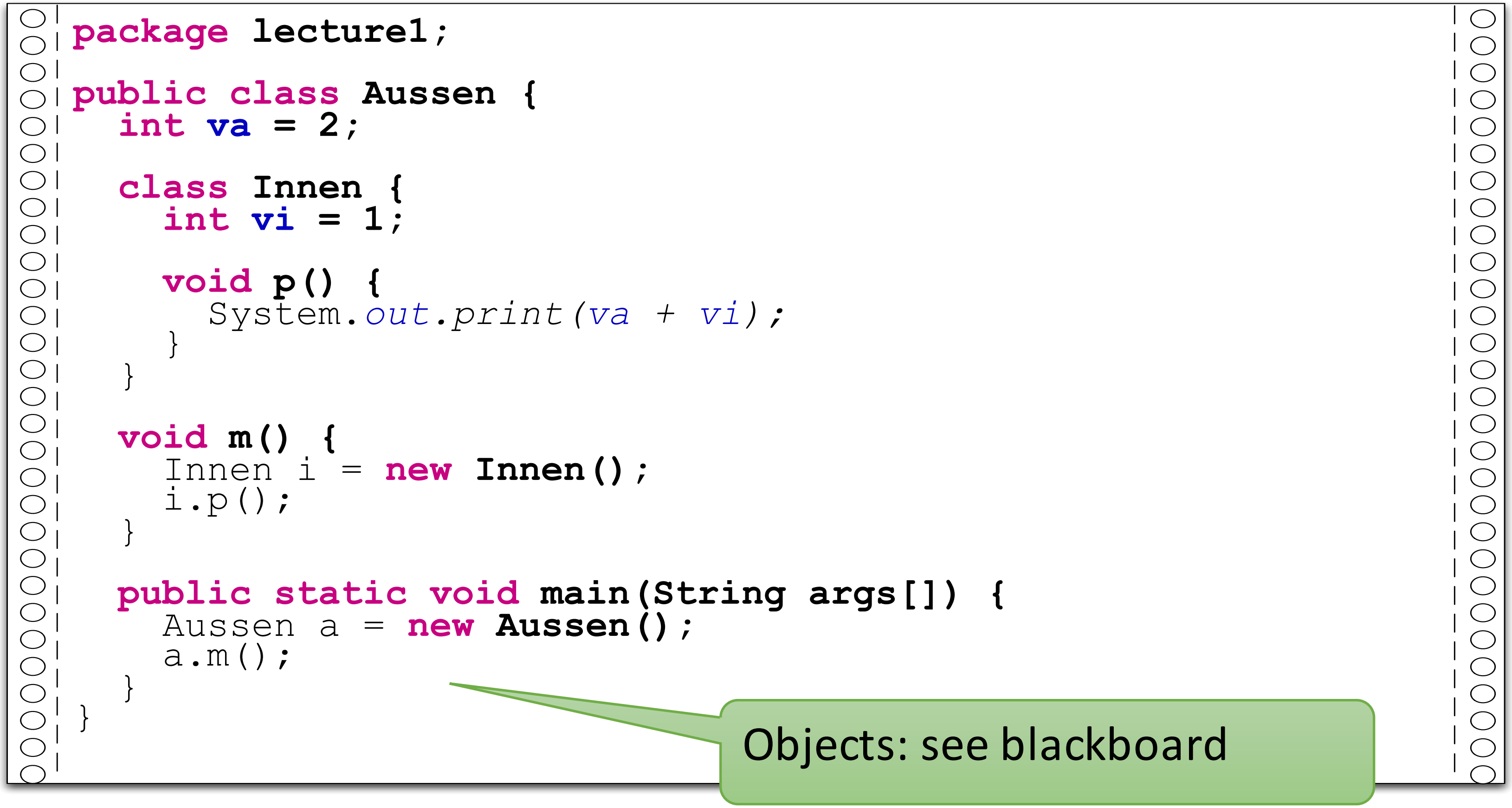
```
package lecture1;

public class Konto {
  private int kontonummer;
  private int kontostand;
  private Aktion letzteAktion;
  public class Aktion {
    private String aktion;
    private int summe;
    Aktion (String a, int s) {
      this.aktion = a;
      this.summe = s;
    }
    public String toString() {
      return kontonummer + ":" + aktion + summe;
    }
  }
  public void abheben (int summe) {
    kontostand = kontostand - summe;
    letzteAktion = new Aktion("Abheben", summe);
  }
  // weitere, z.B. Einzahlen
}
```

Konto.java

Inner Classes

- Inner classes may not have static members
- Inner classes similar to variables of classes
    - Objects of inner classes are bound to objects of outer class
- Inner classes can access all members of the outer class
- Inner class has reference to outer class:
    - this refers to the current Action
    - Action.this refers to the surrounding account

Beispiel
Objekte und Referenzen



```java
package lecture1;

public class Aussen {
   int va = 2;

   class Innen {
      int vi = 1;

      void p() {
         System.out.print(va + vi);
      }
   }

   void m() {
      Innen i = new Innen();
      i.p();
   }

   public static void main(String args[]) {
      Aussen a = new Aussen();
      a.m();
   }
}
```

Objects: see blackboard

Aussen.java

Object creation

Objects of inner classes can be created from outside

Example:

```
        Konto k = new Konto();
```

Then

```
        k.letzteAktion = k.new Aktion(„Einzahlen“,100);
```

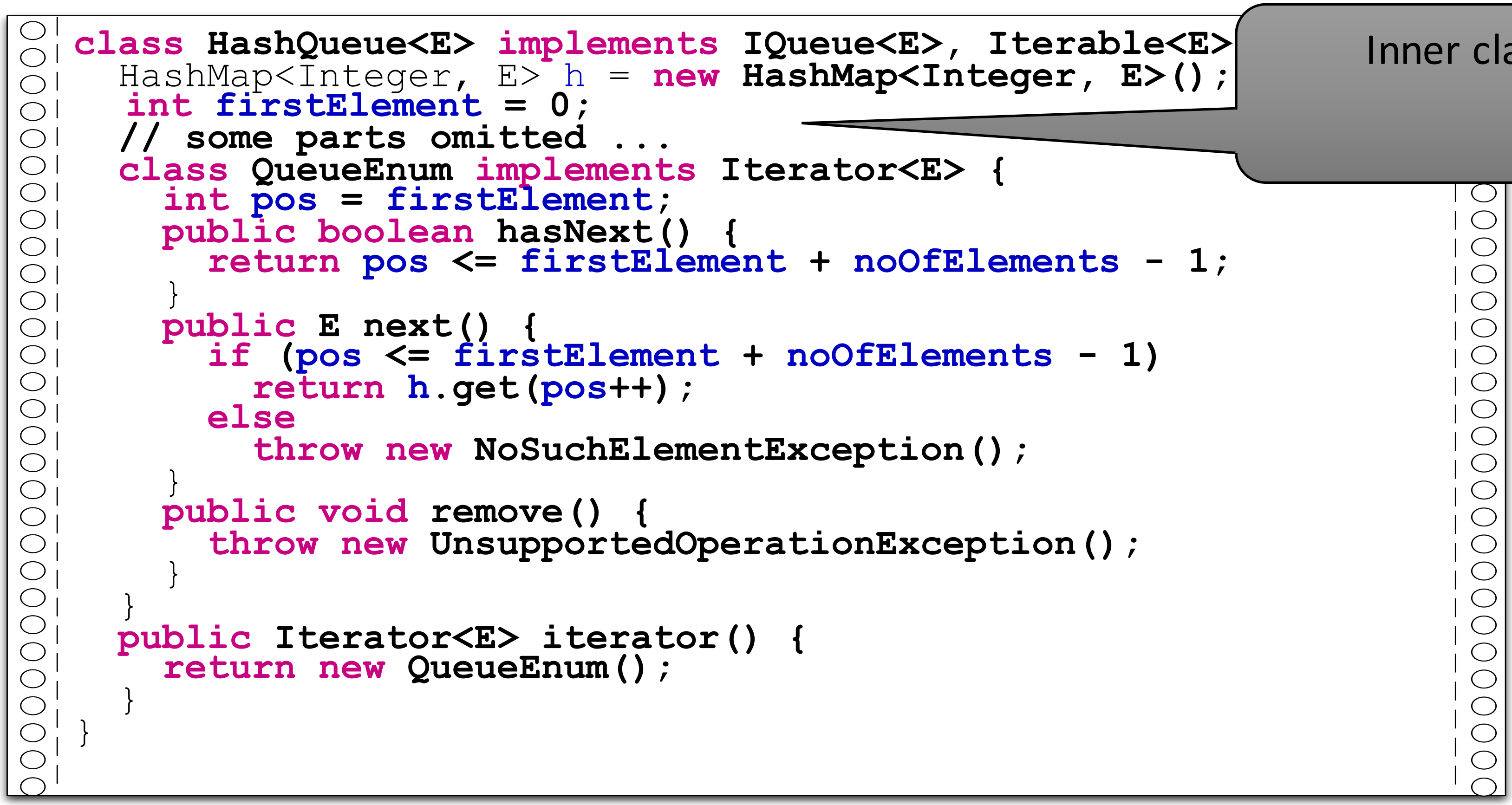
- Sets last action of account (however: without an account object there is no action object)

Usage example

```
public void ueberweisen(Konto anderes, int summe) {
   anderes.abheben(summe);
   this.einzahlen(summe);
   letzteAktion = new Aktion("Ueberweisen", summe);
   anderes.letzteAktion =
        anderes.new Aktion("Ueberweisen", summe);
}
```

Konto.java

Iterator for HashQueue contd.



```
class HashQueue<E> implements IQueue<E>, Iterable<E>
 HashMap<Integer, E> h = new HashMap<Integer, E>();
 int firstElement = 0;
 // some parts omitted ...
 class QueueEnum implements Iterator<E> {
  int pos = firstElement;
  public boolean hasNext() {
   return pos <= firstElement + noOfElements - 1;
  }
  public E next() {
   if (pos <= firstElement + noOfElements - 1)
    return h.get(pos++);
   else
    throw new NoSuchElementException();
  }
  public void remove() {
   throw new UnsupportedOperationException();
  }
 }
 public Iterator<E> iterator() {
  return new QueueEnum();
 }
}
```

Inner class

Using the iterator object...

```
public void print() {
    Iterator<E> e = iterator();
    System.out.println("-------------------");
    while (e.hasNext()) {
      System.out.print(e.next() + " ");
    }
    System.out.println();
}
```

Second example...

```
// External usage:
// Implementing a search function
// boolean find(...)

 public static boolean find(HashQueue<Integer> q, int x)  {
    Iterator<Integer> e = q.iterator();
    boolean xGefunden = false;
    while (!xGefunden && e.hasNext()) {
      xGefunden = (x == ((Integer) e.next()).intValue());
    }
    return xGefunden;
  }
```

HashQueue.java

2. Example
Iterator on a linked list

```java
package lecture1;

class Node<E> {
   final E data;
   Node<E> link;

   Node(E d, Node<E> n) {
      data = d;
      link = n;
   }
}
```

Node.java

LinkedList

```
class LinList<E> {
 Node<E> start = null;

 public void add(E x) {
  if (isEmpty())
   start = new Node<E>(x, null);
  else {
   Node<E> pos = start;
   while (hasSuccessor(pos)) {
    pos = pos.link;
   }
   pos.link = new Node<E>(x, null);
  }
 }
 private boolean hasSuccessor(Node<E> pos) {
  return pos.link != null;
 }
 private boolean isEmpty() {
  return start == null;
 }
}
```

Node.java

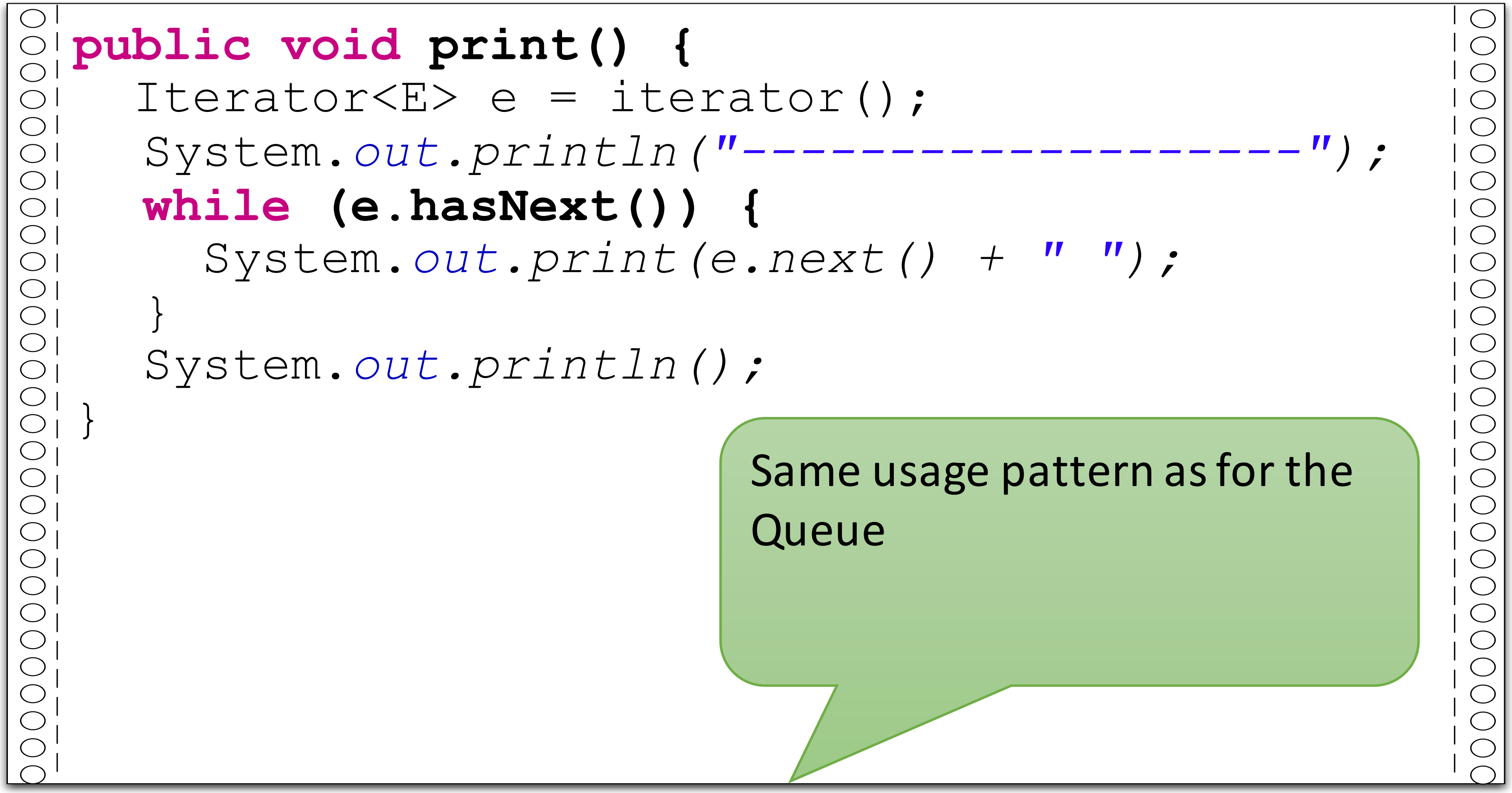
Iterator for LinList

```
class ListEnum implements Iterator<E> {
   Node<E> pos = start;
   public boolean hasNext() {
     return pos != null;
   }
   public E next() {
     if (pos != null) {
       E x = pos.data;
       pos = pos.link;
       return x;
     } else
       throw new NoSuchElementException();
   }
   public void remove() {
     throw new UnsupportedOperationException();
   }
 }
 public Iterator<E> iterator() {
   return new ListEnum();
 }
```

Insert inner class for iterators and iterator() operation to create instances

Node.java

Iterator in print operation

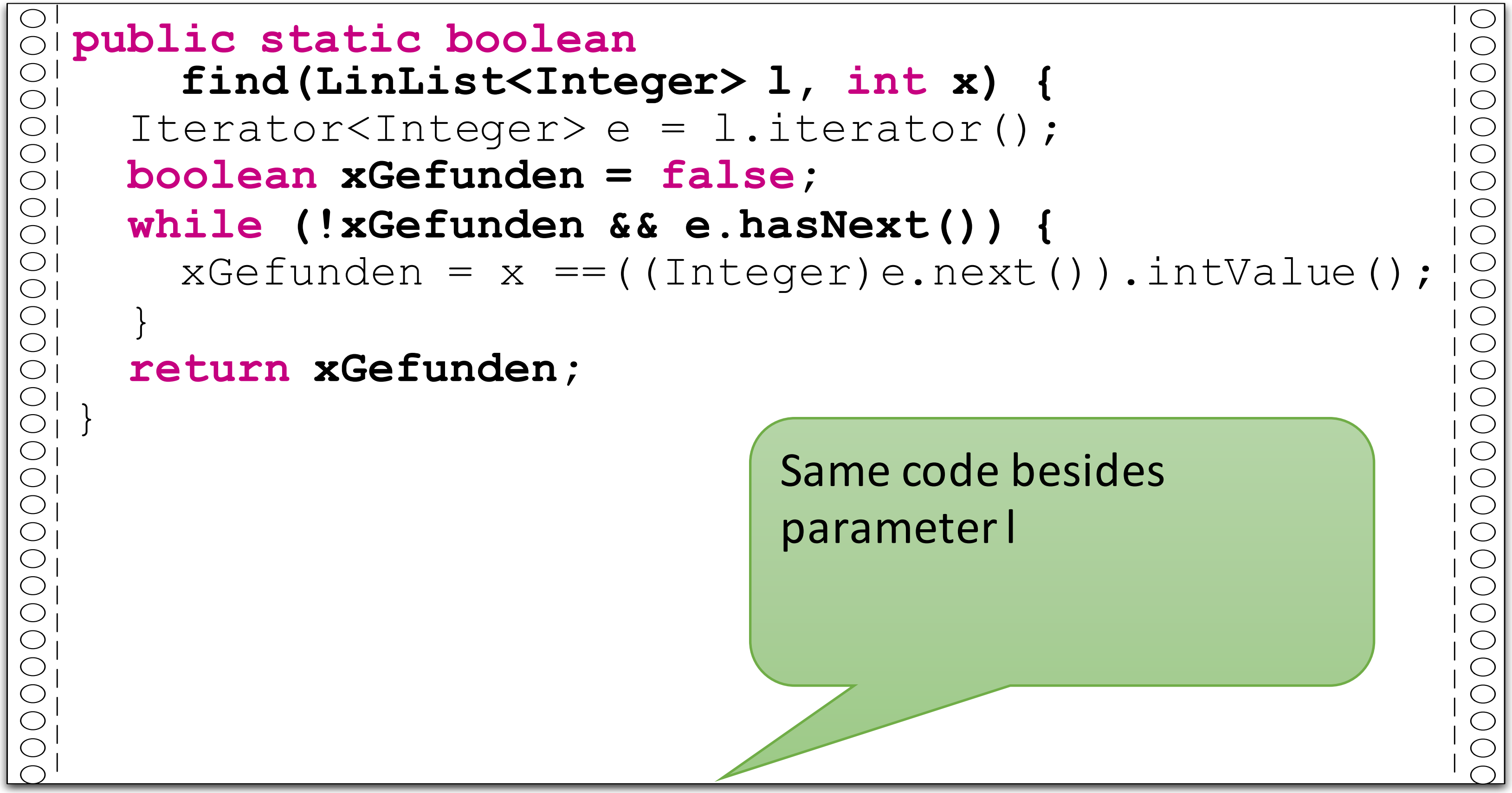


```java
public void print() {
   Iterator<E> e = iterator();
   System.out.println("--------------------");
   while (e.hasNext()) {
      System.out.print(e.next() + " ");
   }
   System.out.println();
}
```

Same usage pattern as for the Queue

Node.java

Iterator for find



```
public static boolean
    find(LinList<Integer> l, int x) {
  Iterator<Integer> e = l.iterator();
  boolean xGefunden = false;
  while (!xGefunden && e.hasNext()) {
    xGefunden = x ==((Integer)e.next()).intValue();
  }
  return xGefunden;
}
```

Same code besides parameter l

So far...

Class `HashQueue<E>`
(implements Interface `IQueue<E>`)
Class `LinList<E>`

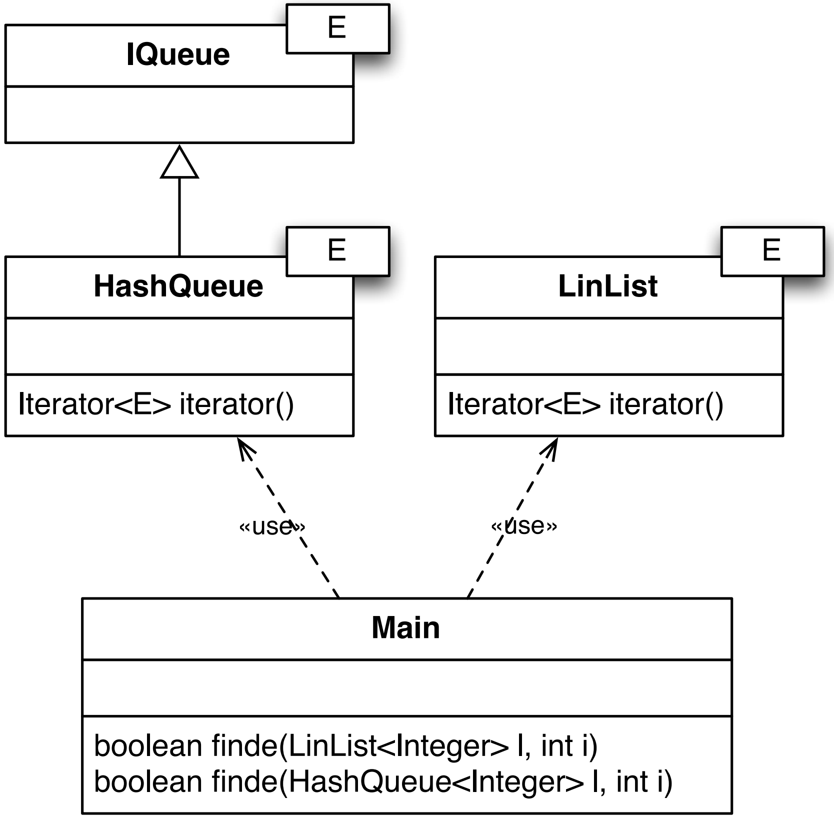
Both have method

```
Iterator<E> iterator()
```

(both implement it by an inner class which does the actual implementation)

Two find methods which for a given
a) HashQueue looks for an element using its iterator
b) LinList looks for an element using its iterator

Current status graphical



Goal

- Reuse find
  - Write once, use for both HashQueue and LinList

Solution

Reuse java.lang interface

```
interface Iterable<T> {
    Iterator<T> iterator();
      // generiert Iterator-Objekt
}
```

Let classes implement it

```
class LinList<E> implements Iterable<E> {
   public Iterator<E> iterator() {
            return new ListEnum(); }
    ...
}
```

Others, e.g., HashQueue, implement it too

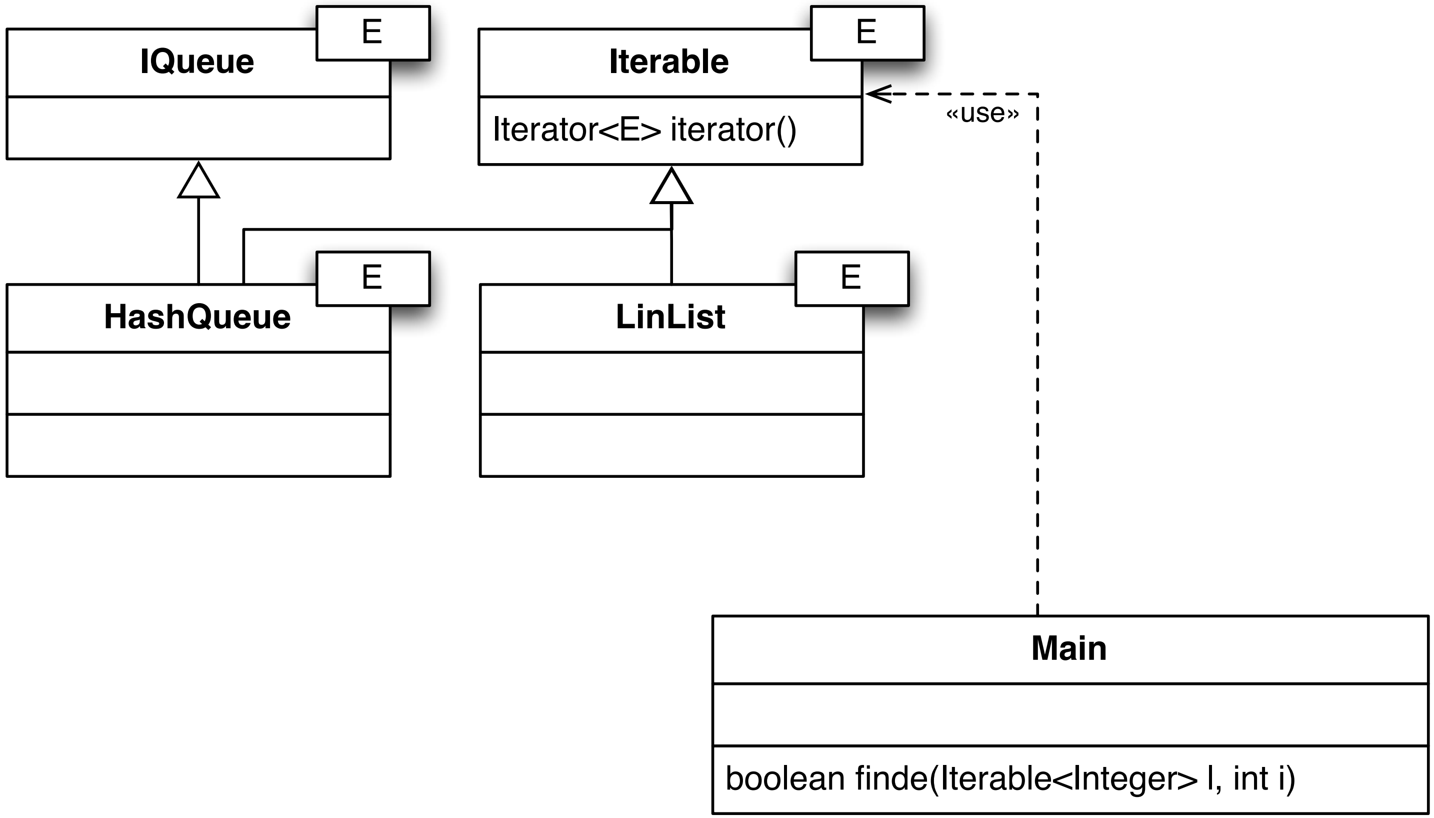
Then…

`Find`-Method independent of concrete datastructure implementation

Only use `Iterable`

```
public static boolean finde
       (Iterable<Integer> l, int x){

    Iterator<Integer> e = l.iterator();
   …
}
```

Class diagram of solution



Further benefits

- For each datastructure implementing Iterable we can use foreach in Java or C++
- API of `Iterable`:
  - “Implementing this interface allows an object to be the target of the "foreach" statement.”

Erweitertes For

To iterate over program structures use

```
arr: irgendein Array
for (int i = 0; i < arr.length; i++) {
   // arr[i]
}
```

Or better the short form (less error-prone)

```
for (Typ var : arrayname) {
   // var benutzen
}
```

Example

Print all command line arguments:

```
public static void main (String [] args) {
  for (String arg : args) {
        System.out.println(arg);
  }
}
```

Nested foreach:

```
int [] [] matrix = new int[3][4];
for (int [] zeile : matrix) {
  for (int elem : zeile) {
        System.out.print(elem + " ");
  }
  System.out.println();
}
```