In [ ]:

```
// git@github.com:janDigeser/StruProUeb.git

// github.com/janDigeser/StruProUeb

// jan.konstantin.digeser@uni-jena.de
// [StruPro]

public interface Vergleichbar{
    int compareTo(Object other);
}
```

In [ ]:

```
public class Entry<T> implements Vergleichbar{
    public static int PREC = 6;
    double timeStamp;
    T data;

    public Entry(double ts, T dt){
        timeStamp = ts;
        data = dt;
    }

    @Override
    public int compareTo(Object other){
        Entry othr = (Entry) other;
        if (timeStamp < othr.timeStamp)
            return -1;
        else if (timeStamp > othr.timeStamp)
            return 1;
        return 0;
    }

    @Override
    public String toString(){
        return String.format(Locale.US, "Entry( %."+PREC+"f" , timeStamp) + ", "+ data
+" )";
    }
}
```

In [ ]:

```
Entry.PREC = 3
```

PriorityQueue

In [ ]:

```
public abstract class PriorityQueue{
    abstract public void add(Vergleichbar e);
    abstract public boolean isEmpty();
    abstract public Vergleichbar poll();
}
```

In [ ]:

```
public class MyPriorityQ extends PriorityQueue{

    private LinkedList<Vergleichbar> queue;

    public MyPriorityQ(){
        queue = new LinkedList<>();
    }

    @Override
    public void add(Vergleichbar e){

        if (queue.isEmpty()){
            queue.add(e);
            return;
        }
        ListIterator<Vergleichbar> iter = queue.listIterator(0);
        while(iter.hasNext()){
            Vergleichbar cur = iter.next();   // nächstes element in Liste
            if (e.compareTo(cur) <= 0){       // falls aktuelles element größer oder gl
eich ist
                iter.previous();              // direkt davor einfügen
                iter.add(e);
                return;                        // und abbrechen
            }
        }
        iter.add(e);


    }

    @Override
    public boolean isEmpty(){ return queue.isEmpty();}

    @Override
    public Vergleichbar poll(){
        return queue.poll();
    }
}
```

In [ ]:

```
PriorityQueue q = new MyPriorityQ();
for(int i = 0; i < 20; i++){
    double val = (Math.random()*20);
    q.add(new Entry(val, String.valueOf(val)));
}
```

In [ ]:

```
while(!q.isEmpty())
    System.out.println((Entry) q.poll());
```

In [ ]:

```
public class Pair implements Vergleichbar{

    private Integer first, second;

    public String toString(){
        return "( " + first + " " + second +" )";
    }

    public Pair(Integer a, Integer b){
        first = a;
        second = b;
    }

    public int compareTo(Object obj){
        Pair other = (Pair) obj;
        if (first - other.first == 0){
            return second - other.second;
        }
        return first - other.first;
    }

}
```

In [ ]:

```
PriorityQueue p = new MyPriorityQ();
for(int i = 0; i < 20; i++){
    p.add(new Pair((int) (Math.random()*20),(int) (Math.random()*20)));
}
```

In [ ]:

```
while(!p.isEmpty())
    System.out.println((Pair) p.poll())
```

Generics

Schreiben von Klassen, die mit unterschiedlichen Typen arbeiten können

Interfaces und ABCs aus der Java-Bibliothek

Eigene Klassen einfacher zu benutzen mit Interfaces von Java (Iterable, Comparable, Iterator)

ABCs: (AbstractCollection, AbstractList, ...)

Interface (https://docs.oracle.com/javase/8/docs/api/java/util/List.html)

ABC (https://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html)

In [ ]:

```
// Eigene Generische Klasse

public class MyList<T> implements Iterable<T>{


    public Iterator<T> iterator(){
        return new Iterator<T>(){

            Node current = head;

            public boolean hasNext(){
                return current != null;
            }

            public T next(){
                Node tmp = current;
                current = current.child;
                return tmp.data;
            }

        };
    }


    private class Node{
        Node child;
        T data;

        Node(T dt, Node n){
            data = dt;
            child = n;
        }
    }

    Node head;

    MyList(){
        head = null;
    }

    public void add(T elem){
        head = new Node(elem, head);
    }

}
```

In [ ]:

```
MyList<Integer> list = new MyList<>();
```

In [ ]:

```
for(int i = 20; i > 0; i--){
    list.add(i);
}
```

In [ ]:

```
for(Integer elem : list) // Interface Iterable
    System.out.println(elem);
```

In [ ]:

```
// Exakt das selbe wie:
for(Iterator<Integer> iter = list.iterator(); iter.hasNext() ; ){
    Integer elem = iter.next();

    f();
}
```

In [ ]:

```

```