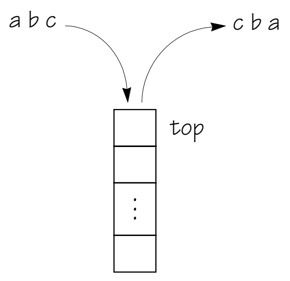
Linjära datastrukturer

Abstrakta datatyper:
- Stackar
- Köer
- Prioritetsköer

Representerade med:
- Länkade listor
- Arrayer

Stack (LIFO-kö)



```
class Stack {

  public Stack();
  public boolean empty()
  public void push(Object X)
  public Object pop()
  public Object peek()

}


Stack a = new Stack();

a.push(2);
a.push(3);
a.push(4);
a.pop() == 4
a.pop() == 3
```

Stack (Länkad lista)

```
class StackNode {
    Object    item;
    StackNode link;
  }

class Stack {

  private StackNode top;

  public boolean empty() {
    return (top == null);
  }

  public void push(Object X) {
    StackNode newNode = new StackNode();
    newNode.item = X;
    newNode.link = top;
    top = newNode;
  }

  public Object pop() {
    StackNode tempNode = top;
    top = top.link;
    return tempNode.item;
  }

  public Object peek() {
    return top.item;
  }

} //end class Stack
```

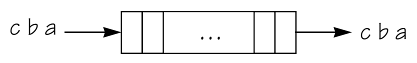
Stack (Sekvensiell)

```
class Stack {

  private int count;
  private int capacity;
  private int capacityIncrement;

  private Object[] itemArray;

  public Stack() {
    count = 0;
    capacity = 10;
    capacityIncrement = 5;
    itemArray = new Object[capacity];
  }

  public boolean empty() {
    return (count == 0);
  }

  public void push(Object X) {

    if (count == capacity) {
      capacity += capacityIncrement;
      Object[] tempArray = new Object[capacity];
      for (int i = 0; i < count; i++) {
        tempArray[i] = itemArray[i];
      }
      itemArray = tempArray;
    }

    itemArray[count++] = X;
  }
```

Stack (forts.)

```
  public Object pop() {

    if (count == 0) {
      return null;
    } else {
      return itemArray[--count];
    }
  }

  public Object peek() {

    if (count == 0) {
      return null;
    } else {
      return itemArray[count-1];
    }
  }


} // end Stack class
```

Kö (FIFO-kö)



```
public class Queue {

  public Queue();
  public boolean empty()
  public void insert(Object x)
  public Object remove()

}
```

Cirkulär kö

```
class Queue {

  private int      front;
  private int      rear;
  private int      count;
  private int      capacity;
  private int      capacityIncrement;

  private Object[] itemArray;

  public Queue() {
    front    =  0;
    rear     =  0;
    count    =  0;
    capacity = 10;
    capacityIncrement = 5;
    itemArray = new Object[capacity];
  }

  public boolean empty() {
    return (count == 0);
  }

  public Object remove() {

    if (count == 0) {
      return null;
    } else {
      Object tempItem = itemArray[front];
      front = (front + 1) % capacity;
      count--;
      return tempItem;
    }
  }
```

Cirkulär kö (forts.)

```
  public void insert(Object newItem) {

    if (count == capacity) {
      capacity += capacityIncrement;
      Object[] tempArray = new Object[capacity];

      for( int i = 0; i < rear; i++ )
        tempArray[i] = itemArray[i];
      for( int i = front; i < count; i++ )
        tempArray[i + capacityIncrement] =
          itemArray[i];

      front += capacityIncrement;
      itemArray = tempArray;

    }

    itemArray[rear] = newItem;
    rear = (rear + 1) % capacity;
    count++;
  }

} // end Queue class
```

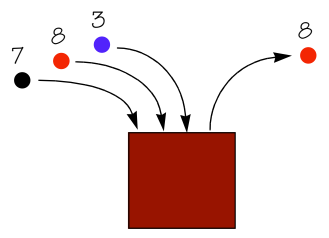
Länkad kö

```
class QueueNode {
  Object    item;
  QueueNode link;
}

class Queue {

  private QueueNode  front;
  private QueueNode  rear;
  private int        count;

  public boolean empty() {
    return (count == 0);
  }

  public void insert(Object newItem) {
    QueueNode temp = new QueueNode();

    temp.item = newItem;
    temp.link = null;
    if (rear == null) {
      front = rear = temp;
    } else {
      rear.link = temp;
      rear = temp;
    }
    count++;
  }
```

Länkad kö (forts.)

```
  public Object remove() {

    if (count == 0) return null;
    else {
      Object tempItem = front.item;
      front = front.link;
      if (front == null) rear = null;
      count--;
      return tempItem;
    }
  }

} // end Queue class
```

Prioritetskö



```
class PriorityQueue {

  public PriorityQueue();
  public int size();
  public void insert(ComparisonKey x);
  public ComparisonKey remove();

}
```

Diverse

```
interface ComparisonKey {
  int compareTo(ComparisonKey value);
  String toString();
}

class ListNode {
  ComparisonKey item;
  ListNode link;
}
```

Länkad prioritetskö

```
class PriorityQueue {
  private int       count;
  private ListNode  itemList;

  public int size() {
    return count;
  }

  private ListNode sortedInsert(ComparisonKey
    Item, ListNode P) {
    if( (P==null)||(Item.compareTo(P.item)>=0) ) {
      ListNode N = new ListNode();
      N.item = Item;
      N.link = P;
      return(N);
    } else {
      P.link = sortedInsert(Item, P.link);
      return(P);
    }
  }

  public void insert(ComparisonKey Item) {
    itemList = sortedInsert(Item, itemList);
    count++;
  }

  public ComparisonKey remove() {
    if (count == 0) return null;
    else {
      ComparisonKey K = itemList.item;
      itemList = itemList.link;
      count--;
      return(K);
    }
  }
} // End PriorityQueue Class
```

Implementation med heltal

```
class PQItem implements ComparisonKey {

  private int key;

  PQItem(int value) {
    key = value;
  }

  public String toString() {
    return Integer.toString(key);
  }

  public int compareTo(ComparisonKey value) {
    int a = this.key;
    int b = ((PQItem)value).key;
    if( a == b ) return 0;
    else if( a > b ) return 1;
    else return -1;
  }

}
```

Implementation med strängar

```
class PQItem implements ComparisonKey {

  private  String  key;

  PQItem(String value) {
    key = value;
  }

  public String toString() {
    return key;
  }

  public int compareTo(ComparisonKey value) {
    String a = this.key;
    String b = ((PQItem)value).key;
    return a.compareTo(b);
  }

}
```