Abstrakt datatyp

- ***Datatyp*** för en variabel

Betecknar i ett programmeringsspråk den mängd värden variabeln får anta. T ex kan en variabel av typ boolean anta värdena true och false.

- ***Abstrakt datatyp (ADT)***

Abstrakt modell tillsammans med operationer definierade på modellen. Finns inte i verkligheten men är "abstraherad " från verkligheten.

- ***Datastruktur***

Används för att *representera* ADT. Är *samlingar av variabler av någon datatyp* som *hör ihop* på något sätt. Datastrukturer skapas genom att man ger namn åt (deklarerar) samlingar. Enklaste mekanismen för detta i Java är fält (array). "Länkade" strukturer via referensvariabler är annan möjlighet.

- ***En del böcker använder beteckningen datastruktur som synonym med ADT.***

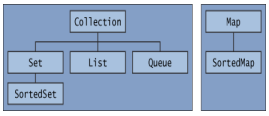
List, Kö, Stack är exempel på ADT)

"Ording och reda" på datastrukturer i java med hjälp av olika interface

En datastruktur är:

- data (representerad på olika sätt)
- Metoder

- Finns i java.util http://java.sun.com/j2se/1.5.0/docs/api/



Collection- generell samling data
List- ordnad samlig av data
Set- ordnad samlig av data som inte får innehålla dubbletter
Map- Samling av data i par (objekt+ nyckel)
Queue- samlig data som tillåter access efter en bestämd ordning

Interface Collection

- Basen är ett interface Collection som representerar en samling element utan något vetskap om hur elementen är positionerade. Collection interfacen innehåller följande

boolean isEmpty();
boolean add(E x);
boolean contains(Object x);
boolean remove(Object x);
void clear();
int size();
Iterator iterator();

Interface List

Listor är en ordnad samling ("rad") av object

public interface List<E> extends Collection

void add(int idx, E element);
boolean add(E element);
E get(int idx);
E set(int idx, E element);

Iterator<E> iterator();
...
...
}

Queue

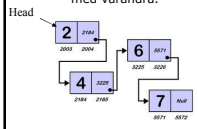
En ordnad samling av objekt
Man kan bara komma åt första objekten i kö
Man kan lägga till bara sist i kö

.....
void enqueue(Object e);
Object dequeue ();

List-klasser i java

- Det finns två konkreta generiska klasser i Javas API för listhantering. Båda implementerar alltså interfacet List
ArrayList som implementerats med array

LinkedList, som använder länkad struktur. Enskilda ”Nod” av data som länkas med varandra.



One Dimensional array

```
Initialization   int a[ ] = new int [12];
Value            1 2 3 4 5 6 7 8 9 10 11 12
Index            a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
System.out.print(a[5]);          Output: 6
```


En ”SimpleDataStructure”

```
public class SimpleDataStructure

{
    // data hålls i en array
      private int theSize;
      private Object [ ] theItems;

// konstruerare
public SimpleDataStructure()
{
  theItems=new Object[10];
  theSize=0;
}

public void add ( Object obj)
{
  theItems [thesize++]=obj;
}
```

```
// ta bort objekt om det hittas

public boolean remove( Object obj)
      {
        int idx=-1;
        boolean  found =false;
      for(int  i=0;i<theItems.length;i++)
          if(obj.equals(theItems[i])){

            idx=i;
            found=true;
            break;
          }

        for( int i = idx; i < theSize - 1; i++ )
          theItems[ i ] = theItems[ i + 1 ];
        theSize--;

        return found;
      }
```


Ett program

```
public class Program
{
...main()....{

SimpleDataStructure bank=new SimpleDataStructure ();
bank.add( new Customer());
bank.add( new Customer());
.....................................
bank.add( new Donkey());


}
}
```


En ”SimpleDataStructure” generic

```
public class SimpleDataStructure <AnyType>
implements Collection <AnyType>
{
        private int theSize;
        private AnyType [ ] theItems;

  public void add ( AnyType obj)
  {
    theItems[thesize++]=obj;
  }

public boolean remove( AnyType obj){
        // idx- hitta index of objekt obj
.....
for( int i = idx; i < size( ) - 1; i++ )
        theItems[ i ] = theItems[ i + 1 ]

theSize--;

return found;
}

}
```

```
public class Program
{
...main()....{
SimpleDataStructure <Customer> bank=
new SimpleDataStructure <Customer> ();


bank.add( new Customer());
bank.add( new Customer());
.....................................
bank.add( new Donkey()); //  compile error


}
```


Generiska klasser i Java 5.0

- Typparametrar anges inom tecknen < och > i klassrubriken:
public class ArrayList**<E>** { ... }
- Typparametern kan användas i klassen för att t ex deklarera attribut, parametrar och returtyper för metoder

```
public class ArrayList <E> {
private E[] elements;
public boolean add(E x) { ... } // lägg in x sist
public E get(int i) { ... } //hämta element på plats i
...
}
```

- Vi skapar instanser genom att ge parametern E ett ”verkligt datatyp”
ArrayList <Integer> intList = new ArrayList <Integer> ();
I intList kan endast Integer-objekt läggas in

Interface Comparable som ”generic”

- I java biblioteket finns interfacen Comparable som används för att definiera objekt som kan jämföras och rangordnas

```
public interface Comparable<AnyType>
{
  public int compareTo( AnyType other);
}
```

En klass som implementear en ”generic” interface, enkel fall

```
class Ball implements Comparable <Ball>
{
   private int radius;
 public Ball ( int value)
{
  radie=value;
}
public void move (){.....}
public int compareTo( Ball other)

{
  if (radius>other.radius)
     return 1;
 else if (radius<other.radius)
   return -1;
else
  return 0

}
```


En klass som implementear en ”generic” interface, Svår fall

AnyType ersäts med Person

```
class Person implements Comparable<Person>
{
   String namen;
   int age;
 // andra metoder  för person
public int compareTo( Person other)
{
 if( name.compareTo(other.getName())==0)  /jämför namn
   { if (adress .....)   // jämför adress


  }


}
}
```


Interface Comparator som alternativ

```
import java.util.*;
      class AgeComparator implements
      Comparator<Person>{

public int compare(Person emp1, Person emp2)
{

            int emp1Age = emp1.getAge();
            int emp2Age = emp2.getAge();

           if(emp1Age > emp2Age)
               return 1;
            else if(emp1Age < emp2Age)
               return -1;
            else
               return 0;
        }

    }
```

```
import java.util.*;
      class NamenComparator implements
      Comparator<Person>{

public int compare(Person emp1, Person
      emp2){

            String emp1Name = emp1.getName();
            String emp2Age = emp2.getName();


       return emp1Name.compareTo(emp2Name)
      }

      }
```


En ”generic” algoritm- findMax metoden

```
public static <AnyType extends Comparable<? super AnyType>>
   AnyType findMax ( AnyType [ ] a )

{
        int maxIndex = 0;

        for( int i = 1; i < a.length; i++ )
            if( a[ i ].compareTo( a[ maxIndex ] ) > 0 )
                maxIndex = i;

        return a[ maxIndex ];
}
```


Iteratorer

```
* Iterator interface for Collections.
*/
public interface Iterator
{
    * Tests if there are more items in the collection.
    * @return true if there are more items in the collection.
    boolean hasNext( );

    * Obtains the next item in the collection.
    * @return the next (as yet unseen) item in the collection.

     Object next( );

    * Remove the last item returned by next.
    * Can only be called once after next.
      void remove( );
}
```


List-klasser i java

- Det finns två konkreta generiska klasser i Javas API för listhantering. Båda implementerar alltså interfacet List
ArrayList som implementerats med array
LinkedList, som använder länkad struktur i implementationen

Man bör bara använda ArrayList om alla insättningar görs sist i listan. Insättningar i andra positioner samt remove orsakar flyttningar av element.