

# Föreläsning 4

ADT

Collection

ArrayList

Generics

Iterator

Stack

# Abstraktion

När man skriver en klass i java anger man synligheten för instansvariabler och metoder.

Synligheten ***private*** anges som regel för ***instansvariabler*** och för metoder som endast är avsedda att användas inom klassen. Instansvariablerna kan på så sätt endast användas av kod inom klassen.

Synligheten ***public*** anges för de ***metoder som ska kunna anropas från kod i andra klasser.***

De metoder som är ***public***-deklarerade utgör användargränsnittet (***interface***) till objektet. Och det är som regel endast dessa metoder som en användare behöver känna till.

Användaren är abstraherad från hur objektet är byggt (***implementerat***) men kan använda objektet.

Exempel: Det går utmärkt att använda ett JButton-objekt utan att känna till:

- \* instansvariabler som finns i klassen
- \* hur metoder är implementerade

# ADT – Abstract Data Type

*Primitiva datatyper* i java är t.ex. *int*, *long* och *double*.

*En abstrakt datatyp* håller någon form av data och har ett användargränssnitt (publika metoder) för att använda datan. Hur datan lagras och klassen är implementerad behöver användaren ej nödvändigtvis känna till.

Ett exempel på en abstrakt datatyp är klassen *String*.

- Klassens data består av en sekvens av tecken.
- Klassen har ett antal publika metoder att anropa, t.ex. `charAt(pos)`, `length()` och `substring(start,end)`. När du använder objekt av typen `String` så är det endast de publika metoderna du behöver känna till.

Med hjälp av en klass bygger man den abstrakta datatypen i java. Det är vanligt att man definierar funktionaliteten i en abstrakt datatyp med hjälp av ett *interface*. Sedan låter man klassen implementera interfacet.

# Collection – ett interface

**Collection** är ett gränssnitt vilket definierar grundläggande funktionalitet för klasser i vilka man kan lagra objekt.

Method Summary	
boolean	<a href="#">add</a> ( <a href="#">E</a> o) Ensures that this collection contains the specified element.
boolean	<a href="#">addAll</a> ( <a href="#">Collection</a> <? extends <a href="#">E</a> > c) Adds all of the elements in the specified collection to this collection.
void	<a href="#">clear</a> () Removes all of the elements from this collection.
boolean	<a href="#">contains</a> ( <a href="#">Object</a> o) Returns true if this collection contains the specified element.
boolean	<a href="#">containsAll</a> ( <a href="#">Collection</a> <?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	<a href="#">equals</a> ( <a href="#">Object</a> o) Compares the specified object with this collection for equality.
int	<a href="#">hashCode</a> () Returns the hash code value for this collection.
boolean	<a href="#">isEmpty</a> () Returns true if this collection contains no elements.
<a href="#">Iterator</a> < <a href="#">E</a> >	<a href="#">iterator</a> () Returns an iterator over the elements in this collection.
boolean	<a href="#">remove</a> ( <a href="#">Object</a> o) Removes a single instance of the specified element from this collection, if it is present.
boolean	<a href="#">removeAll</a> ( <a href="#">Collection</a> <?> c) Removes all this collection's elements that are also contained in the specified collection.
boolean	<a href="#">retainAll</a> ( <a href="#">Collection</a> <?> c) Retains only the elements in this collection that are contained in the specified collection.
int	<a href="#">size</a> () Returns the number of elements in this collection.
<a href="#">Object</a> []	<a href="#">toArray</a> () Returns an array containing all of the elements in this collection.
<a href="#">T</a> []	<a href="#">toArray</a> ( <a href="#">T</a> [] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

# Collection – ett interface

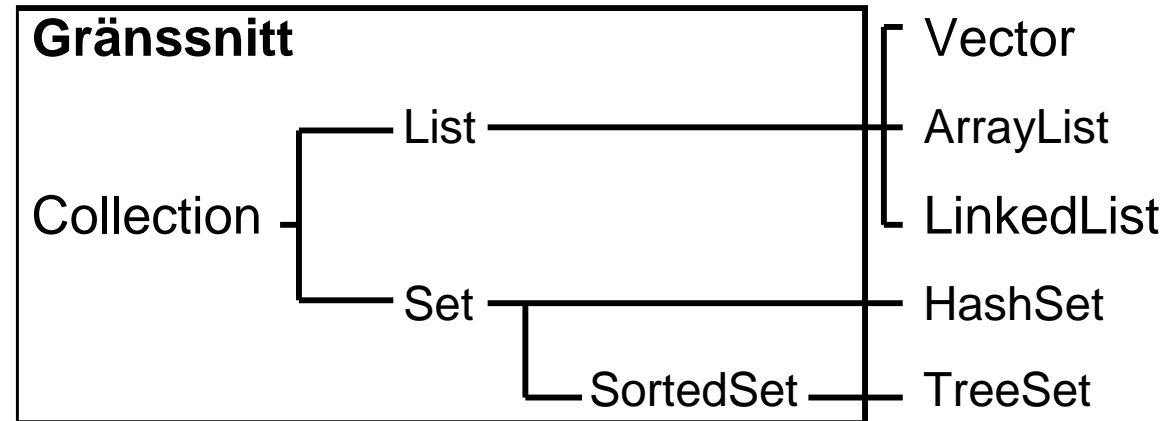
**Collection** är ett gränssnitt vilket definierar grundläggande funktionalitet för klasser i vilka man kan lagra objekt.

Method Summary	
boolean	<a href="#"><code>add(E o)</code></a> Ensures that this collection contains the specified element.
	<a href="#"><code>addAll(Collection c)</code></a> Adds all of the elements in the specified collection to this collection.
void	<a href="#"><code>clear()</code></a> Removes all of the elements from this collection.
boolean	<a href="#"><code>contains(Object o)</code></a> Returns true if this collection contains the specified element.
boolean	<a href="#"><code>containsAll(Collection c)</code></a> Returns true if this collection contains all of the elements in the specified collection.
boolean	<a href="#"><code>equals(Object o)</code></a> Compares the specified object with this collection for equality.
int	<a href="#"><code>hashCode()</code></a> Returns the hash code value for this collection.
boolean	<a href="#"><code>isEmpty()</code></a> Returns true if this collection contains no elements.
<a href="#"><code>Iterator&lt;E&gt;</code></a>	<a href="#"><code>iterator()</code></a> Returns an iterator over the elements in this collection.
	<a href="#"><code>remove(Object o)</code></a> Removes a single instance of the specified element from this
boolean	<a href="#"><code>remove(Object o)</code></a> Removes a single instance of the specified element from this collection, if it is present.
	<a href="#"><code>retainAll(Collection c)</code></a> Retains only the elements in this collection that are contained in the specified collection.
int	<a href="#"><code>size()</code></a> Returns the number of elements in this collection.
	<a href="#"><code>toArray()</code></a> Returns an array containing all of the elements in this collection.
<code>&lt;T&gt; T[]</code>	<a href="#"><code>toArray(T[] a)</code></a> Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

# Collection - samlingar

Interfacen *List*, *Set* och *SortedSet* ärver *Collection*.

*List* och *SortedSet* innehåller ytterligare några metoder för flexiblare funktionalitet.



En lista, *List*, lagrar en ordnad sekvens av element. Ett element kan förekomma flera gånger i listan. Varje element kan nås med ett index


En mängd, *Set*, innehåller ett antal unika element, dvs. inga element förekommer mer än en gång.

I *SortedSet* är elementen i mängden ordnade (sorterade).

# Objektsamlingar

Implementeringar av subgränssnitt till **Collection** utgör tillsammans med implementeringar av gränssnittet **Map** en uppsättning klasser i vilka man kan lagra objekt, objektsamlingar.

Tabellen nedan innehåller de vanligaste klasserna i java. Dock saknas två av Javas ursprungliga klasser, **Vector** och **Hashtable**. Dessa finns fortfarande i Java, framför allt av historiska skäl.

 JAVA		Implementations			
		Hash Table	Resizable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

Fr.o.m. Java 1.5 stödjer Collection-gränssnittet och dess subgränssnitt **generics**. Det innebär att man kan ange vilken typ av element som containerklassen ska innehålla.

Gränssnitten och klasserna finner du som regel i paketet **java.util**.

# List-implementationer

Gränssnittet **List<E>** innehåller, förutom metoderna i Collection, bl.a. metoderna:

Method Summary	
<code>void</code>	<code><a href="#">add</a>(int index, <a href="#">E</a> element)</code> Inserts the specified element at the specified position in this list.
<code><a href="#">E</a></code>	<code><a href="#">get</a>(int index)</code> Returns the element at the specified position in this list.
<code>int</code>	<code><a href="#">indexOf</a>(<a href="#">Object</a> o)</code> Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
<code>int</code>	<code><a href="#">lastIndexOf</a>(<a href="#">Object</a> o)</code> Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
<code><a href="#">E</a></code>	<code><a href="#">remove</a>(int index)</code> Removes the element at the specified position in this list
<code><a href="#">E</a></code>	<code><a href="#">set</a>(int index, <a href="#">E</a> element)</code> Replaces the element at the specified position in this list with the specified element.
<code><a href="#">List</a>&lt;<a href="#">E</a>&gt;</code>	<code><a href="#">subList</a>(int fromIndex, int toIndex)</code> Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.

Klasser som implementerar **List** (och därmed Collection) är bl.a. *ArrayList*, *LinkedList* och *Vector*.

Den underliggande strukturen, *datastrukturen*, för att lagra elementen är dold för användaren. Klassernas namn ger dock information om datastrukturen som används.



# ArrayList utan Generics

Om man använder en ArrayList utan att ange elementtyp

**List list = new ArrayList();** // endast metoder som tillhör List kan användas

så innebär detta att

- Alla typer av objekt kan lagras i samma lista (olämpligt!).  
**list.add( new Person( "Andersson" ) );**  
**list.add( new Orange( 0.23 ) );**
- När man använder t.ex. get-metoden för att hämta referens till ett objekt i listan så måste man typkonvertera referensen så man får en referens till korrekt typ.

**Person p = (Person)list.get( 1 );**

get-metoden returnerar nämligen en referens av typen Object.

Om det endast finns Person-objekt i listan så är typkonverteringen väldigt onödigt. Den borde inte behövas.

Om det däremot finns olika typer av objekt så kan typkonverteringen misslyckas (man försöker kanske typkonvertera en Orange-referens till Person-referens), varvid det kastas ett **ClassCastException** och exekveringen av programmet avslutas.

NoGenerics.java

# ArrayList med Generics

Om man använder en ArrayList där man anger elementtyp

```
List<Person> list = new ArrayList<Person>();
```

så innebär detta att

- Endast objekt av typen Person (och subklasser till person) kan lagras i listan.

```
list.add( new Person( "Andersson" ) );
```

Kompilatorn upptäcker och meddelar misstag (kompileringen avbryts)

```
list.add( new Orange( 0.23 ) ); // GÅR EJ - kompileringsfel!
```

- När man använder t.ex. get-metoden för att hämta referens till ett objekt i listan så behövs ingen typkonvertering. Det finns ju endast Person-objekt i listan.

```
Person p = list.get( 1 );
```

Eftersom typkonverteringen inte behövs längre så slipper vi denna form av ClassCastException.

Generics.java

# Iterator

Gränssnittet Iterator innebär att en klass måste implementera tre metoder.

Method Summary	
boolean	<a href="#">hasNext</a> () Returns true if the iteration has more elements.
<a href="#">E</a>	<a href="#">next</a> () Returns the next element in the iteration.
void	<a href="#">remove</a> () Removes from the underlying collection the last element returned by the iterator. Throws <a href="#">UnsupportedOperationException</a> if the remove operation is not supported by this Iterator.

Med en Iterator går man i genom objekten i en objektsamling:

```
ArrayList<Shape> shapes = new ArrayList<Shape>();
Shape shape;
:
Iterator<Shape> iter = shapes.iterator();
while( iter.hasNext() ) {
    shape = iter.next();
    :
}

for( Iterator<Shape> iter = shapes.iterator(); iter.hasNext(); ) {
    shape = iter.next();
    :
}
```

IteratorEx.java

# ADT - Stack

En **Stack** är en objektsamling med speciella regler för insättning och borttagning av element. I en stack lagrar och kommer man åt elementen enligt **LIFO**-principen – "last-in, first-out". Principen innebär att det endast är det senast lagrade elementet man har åtkomst till. Om man tar bort detta element så har man åtkomst till det näst senast lagrade elementet.

En vanlig bild av detta är en stapel av tallrikar. Den senast placerade tallriken är den man tar när man behöver en tallrik.



## *Operationer på en stack*

<b>push( elem )</b>	lägger till ett element i stacken
<b>pop() : elem</b>	tar bort ett element från stacken och returnerar elementet
<b>peek() : elem</b>	returnerar elementet som är på tur att tas bort
<b>empty() : boolean</b>	returnerar true om stacken är tom och annars false
<b>size() : int</b>	returnerar antalet element på stacken

# Stack för int-variabler

*Operationer på en stack som lagrar int-variabler*

<b>push( int )</b>	lägger till ett element i stacken
<b>pop() : int</b>	tar bort ett element från stacken och returnerar elementet
<b>peek() : int</b>	returnerar elementet som är på tur att tas bort
<b>empty() : boolean</b>	returnerar true om stacken är tom och annars false
<b>size() : int</b>	returnerar antalet element på stacken

Att avgöra:

- Hur ska int-variablerna lagras i klassen?
- Hur många element ska stacken rymma?

```
public class IntStack {  
    private int[] elements;  
    private int size = 0; // antal element i stacken  
  
    private IntStack( capacity ) {  
        elements = new int[ capacity ];  
    }  
  
    public void push( int element ) { ... }  
    :  
}
```

IntStack.java

# Stack för Object-referenser

*Operationer på en stack som lagrar Object-referenser*

<b>push( Object )</b>	lägger till ett element i stacken
<b>pop() : Object</b>	tar bort ett element från stacken och returnerar elementet
<b>peek() : Object</b>	returnerar elementet som är på tur att tas bort
<b>empty() : boolean</b>	returnerar true om stacken är tom och annars false
<b>size() : int</b>	returnerar antalet element på stacken

Att avgöra:

- Hur ska referenserna lagras i klassen?
- Hur många element ska stacken rymma?

```
public class ObjectStack {  
    private Object[] elements;  
    private int size = 0; // antal element i stacken  
  
    private ObjectStack( capacity ) {  
        elements = new Object[ capacity ];  
    }  
  
    public void push( Object element ) { ... }  
    :  
}
```

ObjectStack.java

# Stack - interface för en generisk objekt-stack

```
1 package f6;
2
3 public interface Stack<T> {
4     /**
5      * Placerar ett element i stacken.
6      * @param element elementet att lägga på stacken
7      */
8     public void push(T element);
9
10    /**
11     * Returnerar det element som senast placerades i stacken. Elementet tas bort från
12     * @return det element som senast placerades i stacken
13     */
14    public T pop();
15
16    /**
17     * Returnerar det element som senast placerade i stacken. Elementet är kvar i stack
18     * @return det element som senast placerades i stacken
19     */
20    public T peek();
21
22    /**
23     * Returnerar true om stacken inte innehåller några element och false om det finns
24     * @return
25     */
26    public boolean empty();
27
28    /**
29     * Returnerar antalet element som finns i stacken.
30     * @return antalet element som finns i stacken
31     */
32    public int size();
33 }
```

Stack.java

# Generisk stack

*Operationer på en stack som lagrar Object-referenser*

<b>push( T )</b>	lägger till ett element i stacken
<b>pop() : T</b>	tar bort ett element från stacken och returnerar elementet
<b>peek() : T</b>	returnerar elementet som är på tur att tas bort
<b>empty() : boolean</b>	returnerar true om stacken är tom och annars false
<b>size() : int</b>	returnerar antalet element på stacken

Att avgöra:

- Hur ska referenserna lagras i klassen?
- Hur många element ska stacken rymma?

```
public class ArrayStack<T> implements Stack<T>{  
    private T[] elements;  
    private int size = 0; // antal element i stacken  
  
    private ArrayStack( capacity ) {  
        elements = (T[])(new Object[ capacity ]);  
    }  
  
    public void push( T element ) { ... }  
    :  
}
```

ArrayStack.java



# Frågor?

