



List, Linked, ArrayList

"Arrays... List... LinkedList.. ArrayList... Which one do we pick?"

INNEHÅLLSFÖRTECKNING

01

Översikt

03

ArrayList

02

List &
LinkedList

04

Eget arbete
&
Uppgifter



01

ÖVERSIKT

Datastruktur

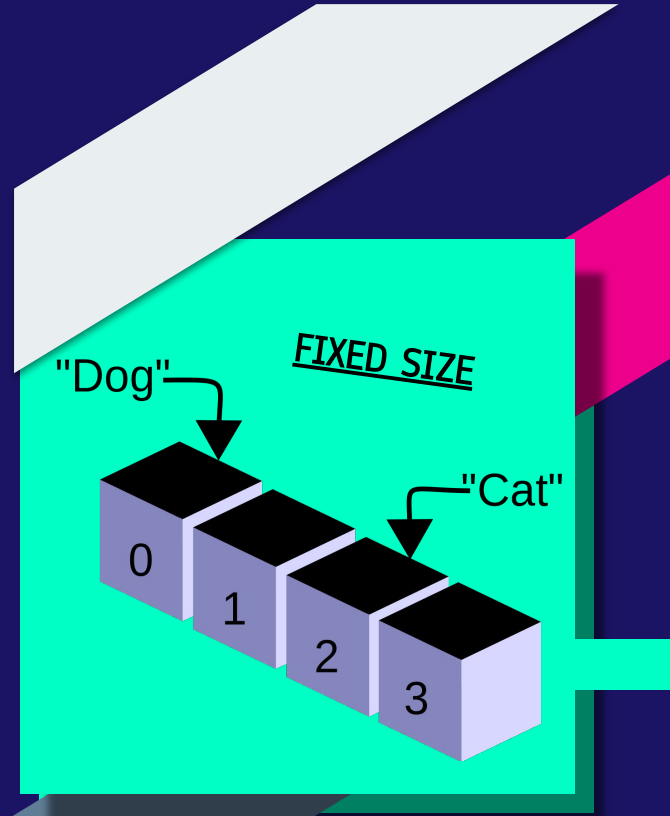


Arrays...?

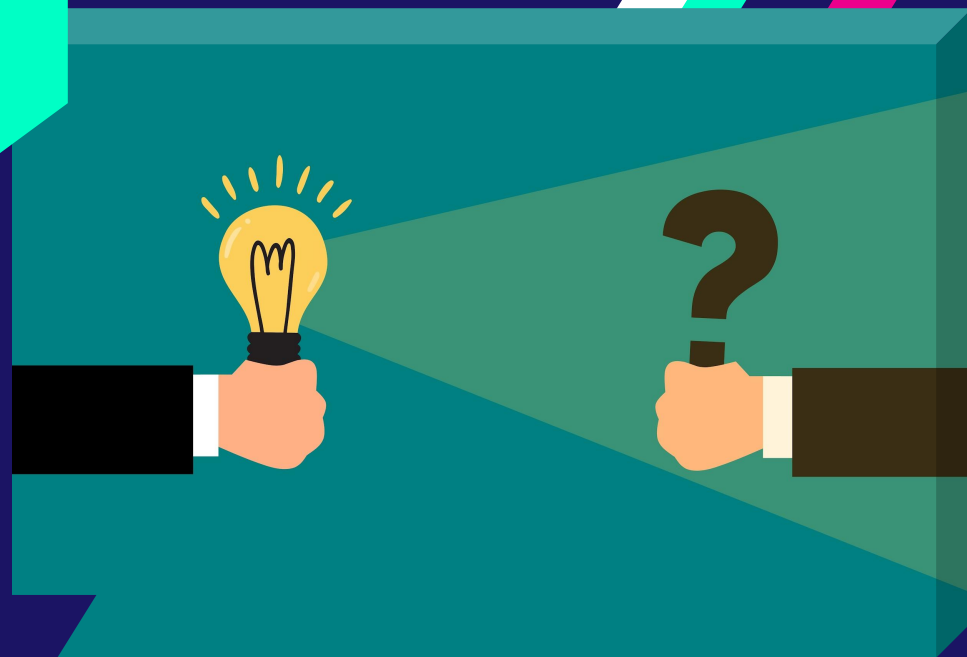
*En array är lik en lista...
en tabell med flera kolumner..
Kan t.om vara två dimensionel..*

*Java array kan dock inte växa...
Alternativ...?*

- *List*
- *LinkedList*
- *ArrayList*
- *Andra Datastrukturer*



Ifrågasätt och ta reda på mer...



Varför?

Växande Array...?

Interface List...?

Olika typer av list <ArrayList>





02

List & LinkedList

Definitioner

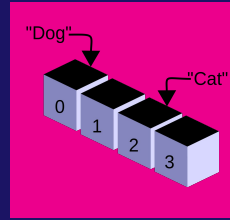


List

List är ett interface.

Interface kommer med metoder som vi kan arbeta direkt med.

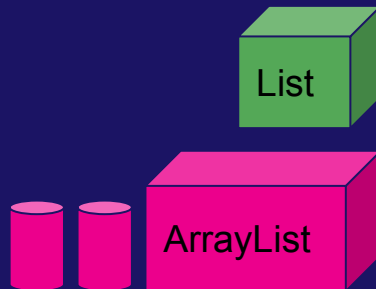
Interfaces har mindre funktionalitet redo än klasser.



LinkedList

Är en array som ärver från List.

Varför List?



Om vi arbetar direkt med 'List' så blir koden mer flexibel!

Om du implementerar ArrayList direkt så kan du inte ändra det senare i koden om du ångrar dig

Eftersom att List är ett interface - ett interface-barn till Collection klassen - så är det enbart en flexibel metod vilket i sin tur kan bli vad som helst!

Exempel:

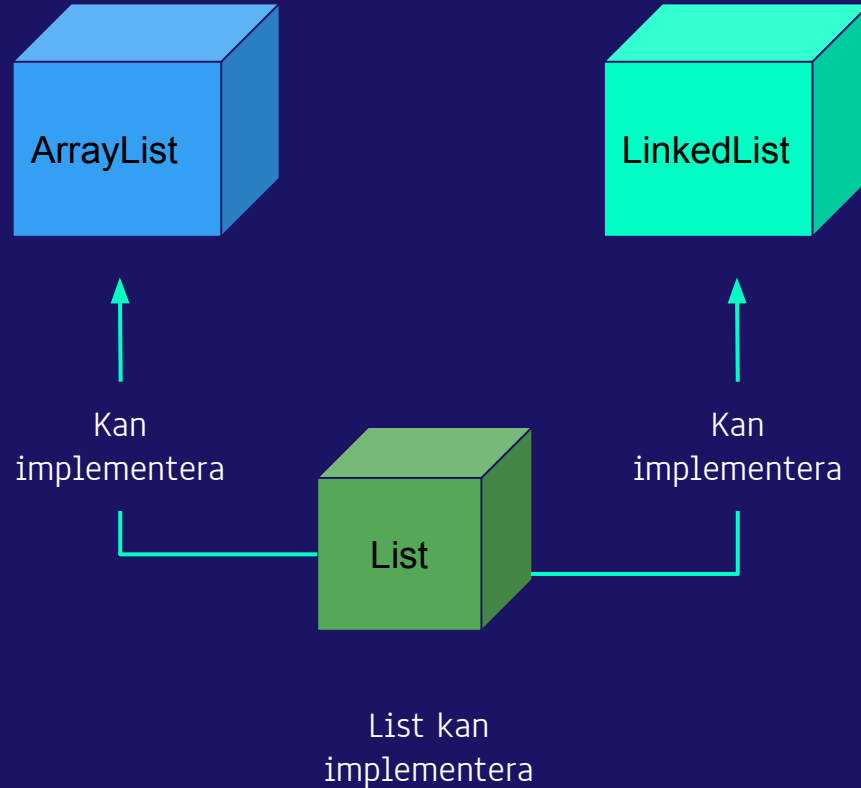
När du skriver List berättar du faktiskt att ditt objekt endast implementerar List-gränssnittet, men du anger inte vilken klass ditt objekt tillhör.

När du skriver ArrayList anger du att din objektklass är en storleksändringsbar array.

Så den första versionen gör din kod mer flexibel i framtiden.

```
List<> myList = new ArrayList OR LinkedList<>();
```

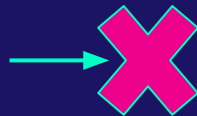
Exempel



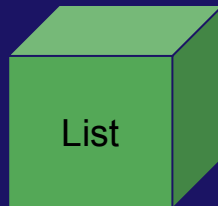
Ärver vi från *List*, så kan vi sedan byta till *LinkedList* om vi ändrar oss senare!

Exempel

```
ArrayList<> myList = new ArrayList<>(); // ONLY
```



ArrayList kan ej
implementera
LinkedList



Här låser vi in oss från
förändring.

ArrayList kan inte bli något
annat än ArrayList.

LinkedList

ArrayList

Låt oss börja med LinkedList...

Arrays?

Vi har arbetat med en ordinarie
Array tidigare.

```
Int[] ageList = { 0, 1, 2 }
```

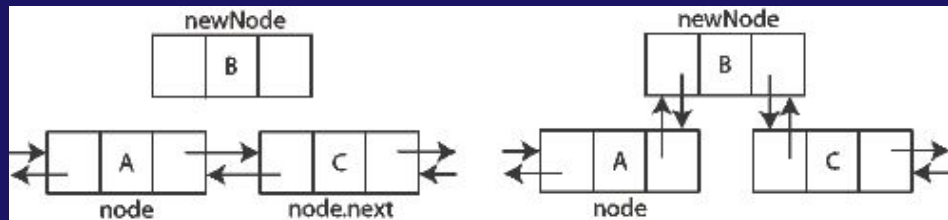
Nu är den dock begränsad



LinkedList?

LinkedList är en dubbellänkad lista, som Javadoc nämner:

“LinkedList is a doubly linked list, Doubly-linked list implementation of the List and Deque interfaces, Implements all optional list operations, and permits all elements (including null).”



LinkedList



Märk av att vi har datan i mitten...
Detta är alltså ett elemen!

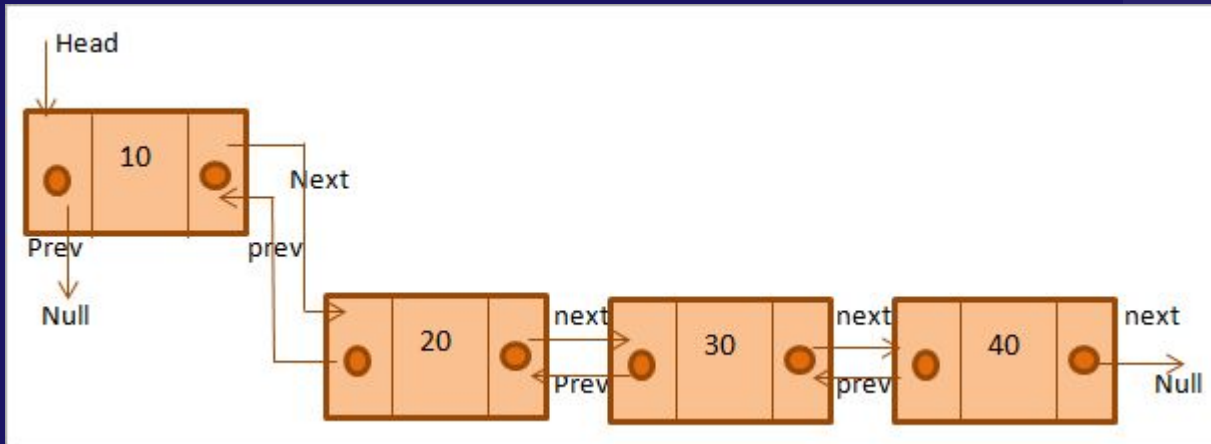
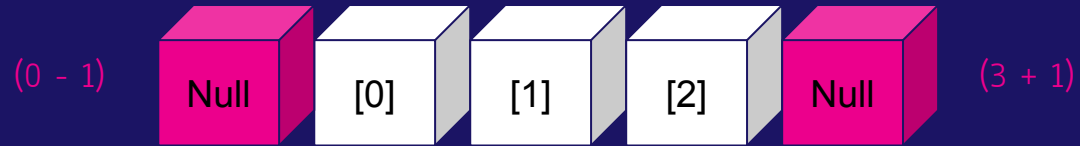
Pointers som pekar på 'next' element men också 'prev'.

Exempel på ett element AKA [0]

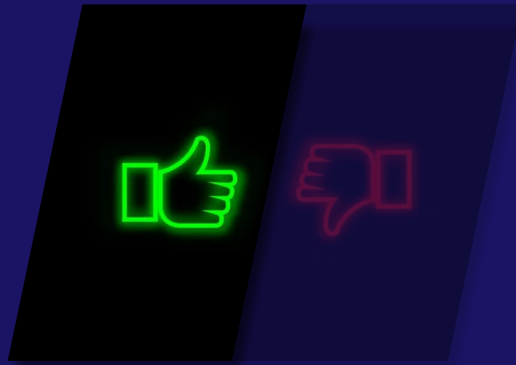
LinkedList

Början och slutet pekar på 'Null'.
Det är här den börjar och tar slut.

```
int [] = { 0 , 1 , 2 }
```



LinkedList



PRO's

- Elements to be added or removed are too large, go for LinkedList
- Adding the elements to LinkedList is unlimited while in an ArrayList, it is restricted to a specific value as it is constrained
 - (ArrayList can also be resized but resizing is a bit costly operation, and you can go with LinkedList in this case)
- Is better at manipulating data
- Supports NULL elements

LinkedList



CON's

- Eftersom det finns en extra pekare i den dubbellänkade listan, dvs den föregående pekaren, krävs ytterligare minnesutrymme för att lagra denna pekare tillsammans med nästa pekare och dataobjekt.
- Alla operationer som tillägg, radering, etc. kräver att både föregående och nästa pekare manipuleras, vilket medför driftskostnader.



BAD

Undvik detta

När vi instansierar på detta sätt
så stänger vi in oss för framtida
förändringar...

Undvik detta!

```
public class InstanceAlarm {  
  
    public void notRecommendedWay() {  
        ArrayList<String> test = new ArrayList<>();  
        // Not a good practice...!  
    }  
}
```

The Good!

Skriv så här!

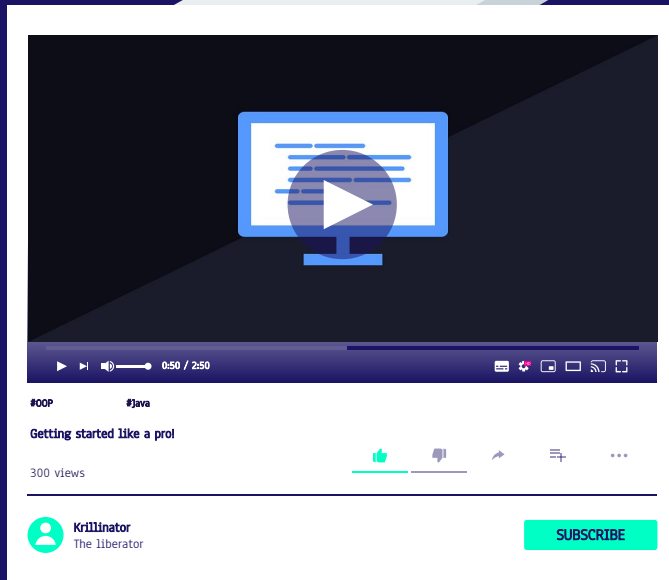
```
public void arrayLists() {  
    List<Integer> scores = new ArrayList<>();  
  
    scores.add(0);  
    scores.add(25);  
  
    System.out.println(scores);  
}
```

Nu börjar vi med att instansiera ett gränssnitt istället!

Med detta 'interface' så följer vi 'best practices' och gör koden lättare att förändra om vi senare hade velat!

SOUT: [0, 25]

LIST & LinkedList – demo



LIST interface
LinkedList



Problem

Vad är best practice...?
Hur skapar vi lists..?



Solution

```
List<String> list = new LinkedList
```

Så skapar vi en abstrakt variant där
vi arbetar direkt med BEST PRACTICE!



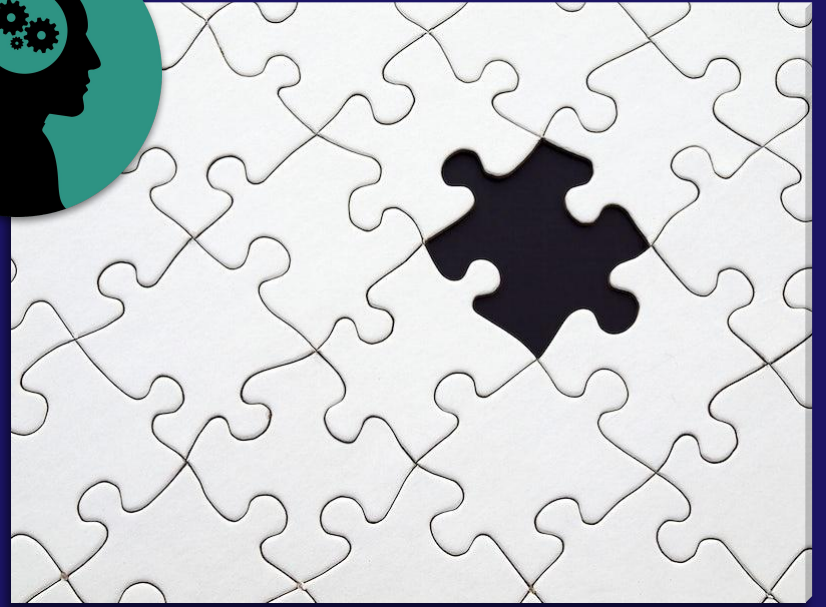


LinkedList Methods

For many cases, the **ArrayList** is more efficient as it is common to need access to random items in the list, but the **LinkedList** provides several methods to do certain operations more efficiently:

Method	Description	Try it
addFirst()	Adds an item to the beginning of the list.	Try it »
addLast()	Add an item to the end of the list	Try it »
removeFirst()	Remove an item from the beginning of the list.	Try it »
removeLast()	Remove an item from the end of the list	Try it »
getFirst()	Get the item at the beginning of the list	Try it »
getLast()	Get the item at the end of the list	Try it »

Frågor?






03

ArrayList

ArrayList

Likt LinkedList så kan vi
Ärva från List och arbeta
Oss ut från kedjan vilket
Följer best practices!

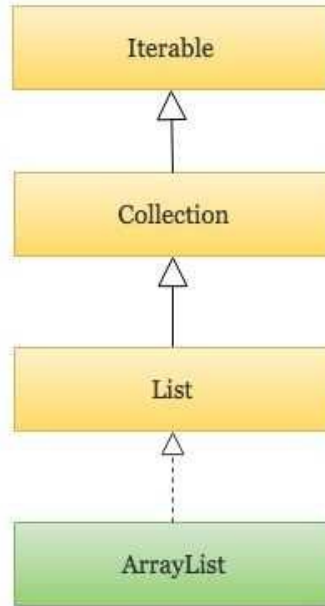
Implements 

Extends 

Interface

Class

Java ArrayList Class
Hierarchy



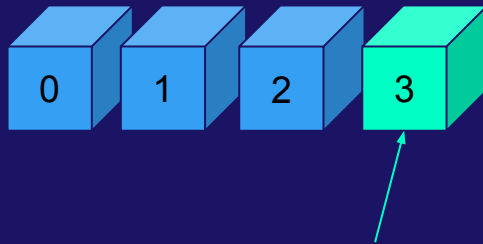
Hur fungerar ArrayList



ArrayList

`.add()`

lägger till index 3,
utökar storleken på arrayen!



En ArrayList är en array som kan ändras i storlek,
även kallad en dynamisk array.

Den växer sin storlek för att rymma nya element
och krymper storleken när elementen tas bort.
Precis som linkedlist



ArrayList



PRO's

- ArrayList is better for storing data
- Consumes less memory in comparison to LinkedList
- ArrayList elements can be directly or randomly accessed while in LinkedList, the elements can be accessed only sequentially
- ArrayList is faster in get()
- Iterating over ArrayList is faster.
- Compact in memory == more cache friendly
- Java 6 implemented ArrayDeque (more about that soon)
- Supports NULL elements

ArrayList



CON's

- adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap
- If you add more elements than the capacity of the underlying array, a new array (1.5 times the size) is allocated, and the old array is copied to the new one.
- InitialSize of an arrayList is small (10 elements)
 - To avoid the high cost of resizing when you know you're going to add a lot of elements, construct the ArrayList with a higher initial capacity

Setup

Skriv så här!

För att komma igång börja med att specificera `List<DataTyp>`

Därefter kan vi lägga till värden med `.add`

```
public void arrayLists () {  
    List<Integer> scores = new ArrayList<>();  
    scores.add(25);  
}
```

Get

Skriv så här!

Här så hämtar vi värde med 'get'

```
public void arrayLists () {  
    List<Integer> scores = new  
    ArrayList<>();  
  
    scores.get(0);  
  
    System.out.println(scores);  
}
```

Remove

Skriv så här!

Här tar vi bort med index 0

Det finns även en 'removeAll' om man vill rensa allt!

```
public void arrayLists () {  
    List<Integer> scores = new  
    ArrayList<>();  
  
    scores.remove(0);  
  
    System.out.println(scores);  
}
```


Set

Skriv så här!

Här ändrar vi på index 0 till värdet 25!

```
public void arrayLists() {  
    List<Integer> scores = new  
    ArrayList<>();  
  
    scores.set(0, 25);  
  
    System.out.println(scores);  
}
```

Init Value

Skriv så här!

```
public void arrayLists() {  
  
    List<Integer> scores = new  
    ArrayList<>(Arrays.asList(9, 15, 24));  
  
    System.out.println(scores);  
}
```

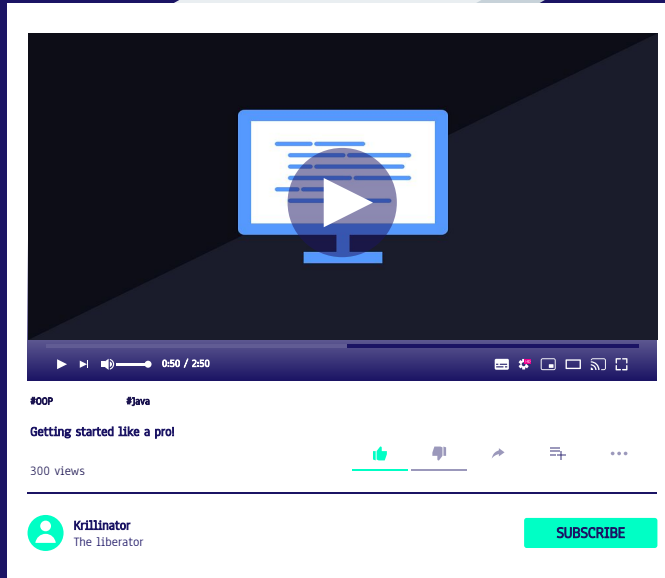
```
Arrays.asList(9, 15, 24)
```

Detta är inte en nödvändighet utan görs bara om man behöver extra värden i från början!

Vi avslutar med en SOUT

Gör bara detta om du har ett högt antal.
Annars sätter vi 'init' värdet till 3 - ILLA

ArrayList – DEMO



ArrayList
List

Add / Get / Remove / Set

Sammanfattning



Who wins?

- It's an efficiency question. LinkedList is fast for adding and deleting elements, but slow to access a specific element. ArrayList is fast for accessing a specific element but can be slow to add to either end, and especially slow to delete in the middle.
- <https://docs.google.com/spreadsheets/d/1lXAm9dF53e0lHIhYa1ecCZOzBhV8R0iteRPAiLHTN2Y/edit#gid=1595957819>

IF IN DOUBT

USE ARRAYLIST



04

Uppgifter
&
Eget arbete

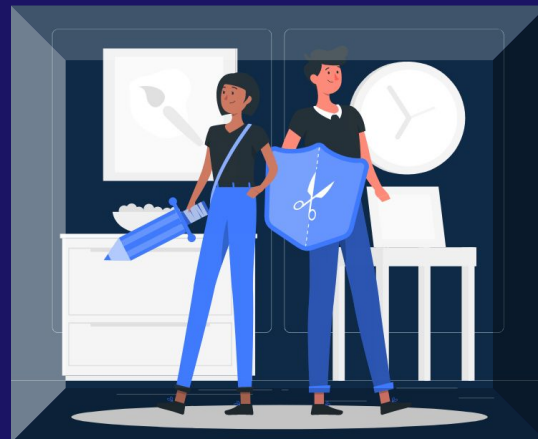
Välkommen till Uppgifterna!

Uppgifterna är till för att testa dina färdigheter och kunskaper för att både öva och repetera på det vi har arbetat med under föreläsningarna.

Dessa är **INTE** obligatoriska.

Men är **starkt rekommenderat** att arbeta med!

Uppgifter

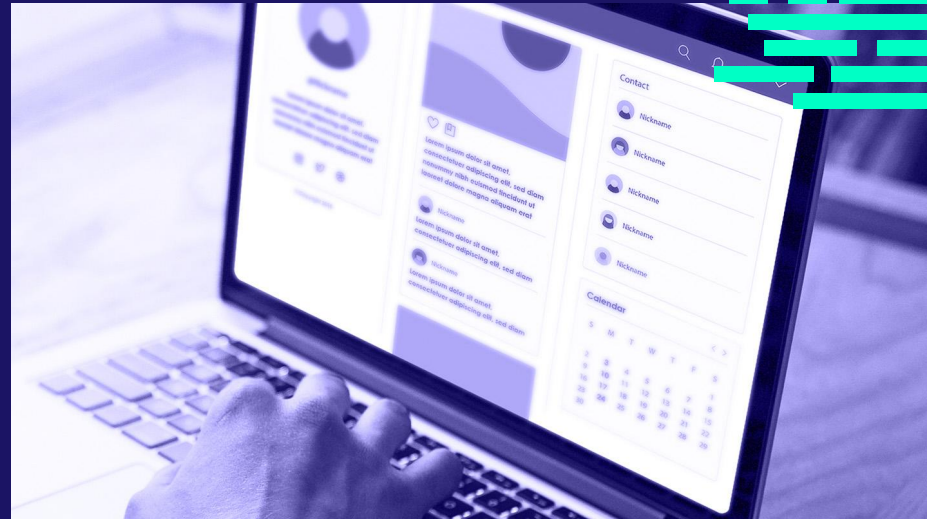


UPPGIFT

Create an **all new** project and label it: *JavaB_2_Övning*
Skapa main funktion!

Inside your 'src' folder
right-click and choose: 'package'
Name it: com.YOURNAME.demo

Part 0



```

1      // -Uppgift #1- //
2
3      /* INSTRUCTIONS
4         Create a new class - TestLists
5
6         Within the new class, create:
7             public void arrayListMethod() {}
8             public void linkedListMethod() {}
9
10        Inside of arrayListMethod,
11        instantiate a new like so:
12        'list<Integer> scoreList = new ArrayList<>()'
13
14        Do the same with linkedListMethod.
15
16        In both methods, add a new element.
17
18        Finish by printing the respective Arrays
19        inside the console to see the effect!
20
21    */
22
23

```

Uppgift #1

Creating new classes and methods to test, is great for debugging and practicing OOP!

Part #1 is only the setup. But here we learn how to add and setup the different lists!

Notice how we write 'list' first, there's a point to this...

```

1          // -Uppgift #2- //
2
3      /* INSTRUCTIONS
4
5          Try out the following commands for both
6          ArrayList & LinkedList.
7
8          .add()
9          .remove()
10         .set()
11         .get()
12
13         System.out.println() the arrays so you
14         Can see the results!
15
16     */
17
18     // HINT & Examples
19     hint(" myList.add() ")      // Adds element
20     hint(" myList.remove() ")  // Removes element
21     hint(" myList.set() ")     // Changes element
22     hint(" myList.get() ")     // Fetches element
23

```

Uppgift #2

Practicing manipulation of Arrays is a great way to get a better understanding of how we can work with said datastructures!

```

1          // -Uppgift #3- //
2
3      /* INSTRUCTIONS
4
5          Inuti LinkedList-metoden
6
7          Det kommer inte att gå att skriva:
8          scoreList.addFirst()
9
10         Men om vi ändrar om till:
11         Lista◇ till LinkedList◇
12
13         linkedList list = new LinkedList◇();
14         Då kommer detta gå - testa!
15
16
17     */
18
19     hint("We can get built in methods with
20     LinkedList as a type. But this limits the
21     array against future changes...")
22
23

```

Uppgift #3

The problem with writing List is that we don't get all the methods for LinkedList respective ArrayList. However, it's lightweight and dynamic because of it!

```

1          // -Uppgift #4- //
2
3      /* INSTRUCTIONS
4
5          Create a Student class.
6          Student has attributes:
7              String name;
8              int age;
9
10         Create a for loop inside ArrayList
11         method.
12
13         During each iteration, .add() a
14         new student inside of the arrayList!
15     */
16
17     // HINT & Examples
18     hint(" if you want unique values, add index
19     as an age ")
20     hint(" name could be player + index")
21
22     studentList.add(index, index)    // Add index
23

```

Uppgift #4

Another example where we work with for loops to populate an arrayList.

This time you should have:

```
List<Student> studentList
```

MINNS DU?

```
// Varför har vi en 'list' vs  
// 'ArrayList'?  
//  
// Example:  
// list<> myList = new ArrayList<>()  
// varför inte:  
// ArrayList<> myList = new ArrayList
```

THANKS!

Do you have any questions?
kristoffer.johansson@sti.se

sti.learning.nu/

CREDITS: This presentation template was created by
Slidesgo, including icons by Flaticon, and
infographics & images by Freepik.

You can also contact me VIA Teams (quicker response)
Du kan också kontakta mig VIA Teams (Snabbare svar)