Arrayer

Arrayer

Ibland (ofta?) vill man ha flera variabler av samma typ.

String name = “Bill”

String name1 = “Bob”

String name2 = “Billy-Bob”

String name3 = “Björn”

osv…

det här blev ju jobbigt :(

Arrayer

Istället för att ha massa “lösa” variabler att hålla reda på så kan vi istället samla in dessa i en Array

String[] name; //så här deklarerar vi en Array

Vi kan dock **inte** använda den innan vi har allokerat minne för den

String[] name = new String[5]; //nu har vi initierat en Array och allokerat minne för 5 element

Arrayer

Om vi redan vet VAD vi ska stoppa in i Arrayen så kan vi göra det redan vid deklarationen

String[] names = {“Bill”, “Bob”, “Billy-Bob”, Björn”}

En array är en typ av “lista” som innehåller **adressen** till flera variabler av samma typ. Det är mao en **referenstyp** ungefär som **String** är

Arrayer

String[] names = {“Bill”, “Bob”, “Billy-Bob”, ”Björn”}

Arrayen ovan innehåller 4 platser, för att komma åt en specifik plats så anger vi dess index.

första indexet är alltid 0.

exempelvis **names[0]** innehåller en variabel av typen String med värdet “Bill”

```
System.out.print(names[0]);     //skriver ut Bill dvs variabeln som finns på index 0
System.out.print(names[3]);     //skriver ut Björn dvs variabeln som finns på index 3
```

Arrayer

OBS!

Arrayen innehåller bara **adressen** till variabler och inte själva variabeln!

```
int[] values1 = {1,2,3,4}

int[] values2 = values1;
```

så blir values2 INTE en kopia av values! Utan ännu ett namn (en ny referens) som pekar på **samma** Array!

Dvs om vi skriver:

```
values2[0] = 100;
```

innebär att value1[0] = 100 **och** value2[0] = 100;

Variabler

vi tar värdet från tal1 och kopierar till en Ny variabel tal2

```
int tal1 = 5;
int tal2 = tal1;
```

Nu har vi **två olika** variabler med värdet 5


```
tal1 = 10;
```

Vi ändrar värdet på tal1 till 10

Nu har vi **två olika** variabler med **olika värden**

Arrayer

int[] tal = {5, 10, 15, 20, 25}

Vi deklarerar en array och fyller upp med tal

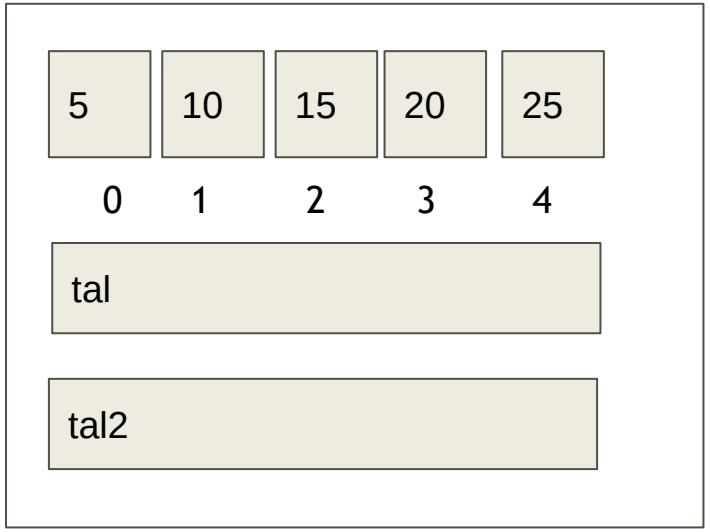
tal2 = tal;

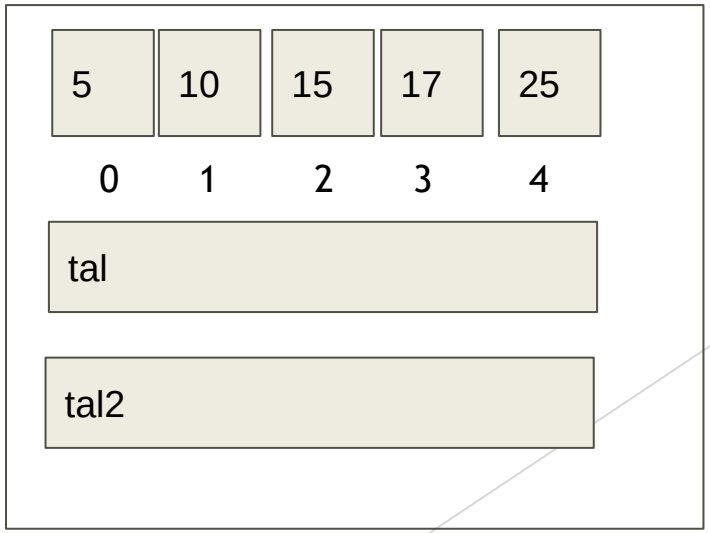
skriver vi bara **tal** kommer vi åt “adresslistan”

tal[3] = 17;

För att komma åt själva variabeln måste vi ange dess index

både tal[3] och tal2[3] har nu värdet 17 eftersom det är samma variabel





Arrayer

Vill man ha en ny kopia av variablerna i en ny array så använd:

int [] values2 = values1.**clone()**;

eller int values2 = Arrays.copyOf(values1, values1.length);

(import java.util.Arrays)

Arrayer

Utskrifter är lite knepigare än vanliga variabler.

System.out.println(values); Ger inte så mycket

(testa gärna! Vad är det vi ser?)

System.out.println(Arrays.toString(values));

Arrayer

Loopa genom en array med en vanlig hederlig for-loop;

```
Int[] values = new int[1000] ;
….
for (int i=0; i<values.length;i++){
        System.out.println(values[i]);
}
```

Eller en for-each loop:

```
Int[] values = new int[1000] ;
….
for (int val: values){
        System.out.println(val);
}
```

Arrayer

Det går även att skapa multidimensionella Arrayer…

dvs Arrayer som innehåller Arrayer

```
int[][] matrix = {
        {1,2,3,4},
        {5,6,7,8},
        {9,10,11,12}
}
```

x = matrix[2][2] ger oss då värdet 11 (rad två värde två)

Listor

Nackdelen med Arrayer är de inte alltid är så flexibla eftersom vi redan från början måste tala om hur många element de innehåller och inte kan förändra det i efterhand annan än genom att skapa en kopia och flytta över värdena - vilket kan bli väldigt långsamt för långa Arrayer.

I java har vi bland annat alternativen java.util.ArrayList<> och java.util.LinkedList<>

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

ArrayList

Använda en ArrayList:

```
ArrayList<String> names = new ArrayList<String>(); //Obs parentes!
```

names.add(”Dolly”) ;	[”Dolly”]
names.add(”Joan”);	[”Dolly”,”Joan”]
names.add(1,”Emmylou”);	[”Dolly”,”Emmylou”,”Joan”];

ArrayList

```
//names = [”Dolly”,”Emmilou”,”Joan”];

names.set(2,”Teri”);                          [”Dolly”,”Emmilou”,”Teri”];

String name = names.get(1);                   // name =”Emmilou”;

names.remove (0);                             [”Emmilou”, ”Teri”];
System.out.println(names);
names.size();                                          //  2
```