Snabbguide – skillnaden mellan vektorer och listor

Koden nedan skiljer sig beroende på vad som lagras i vektorn eller listan. I exemplet väljer jag att lagra Sträng objekt. Men det kan lika gärna vara heltal, Turtle-objekt eller vad som helst.

	Vektor	Lista
Deklarera och skapa	`String [] sl = new String[10];` Man måste ange storleken när man skapar en vektor. Om man inte lagt in några värden så ligger det null på alla platser om vektorn är till för att lagra objekt. Om det är en int[ ] ligger det 0 på alla platser och för en boolean[ ] ligger det false på alla platser till att börja med	`ArrayList<String> sl = new ArrayList<String>();` Man anger ingen storlek till ArrayList och en ArrayList har aldrig några tomma platser (utåt sett) utan när s har skapats ovan så kommer den att vara helt tom innan vi lagt in något i den. (alltså inte innehålla null). Tom lista innebär att: • sl.get(0) => krashar • sl.isEmpty() => true • sl.size() => 0
ta reda på längden	`sl.length`	`sl.size()`

Sätta in ett element	`sl[i]="Hej";`
	Man går alltid via ett index för att sätta in ett värde. Men det behöver förstås inte vara ett fast värde, det kan lika gärna vara en strängvariabel som kommer var som helst ifrån:
	`String s = scan.next();`
	…
	`sl[i] = s;`
	Obs: ***`0<=i<=sl.length-1`***
		`sl.add("Hej");` `//Lägger strängen "Hej" sist i listan`
		`sl.add(i, "Hej")` `//Lägger strängen "Hej" på index i. Allt som låg på i och uppåt flyttas då upp ett steg. Inget skrivs över`
		Givetvis kan man även här byta ut "Hej" mot godtycklig strängvariabel.
		Observera att om man använder index för att lägga till måste indexet hålla sig inom gränserna, precis som för vektorn. Med skillnaden att man faktiskt får ange ett index som är lika med listans storlek (för att lägga något sist). ***`0<=i<=sl.size()`***

	Vektor	Lista

Exempel på att använda ett element	`String s = sl[i];//Sparar undan det som ligger på platsen i`
	`System.out.println(sl[i]);`
	`//skriver ut strängen på platsen i`
	Obs: ***`0<=i<=sl.length-1`***
		`String s = sl.get(i);//Sparar undan det som ligger på platsen i`
		`System.out.println(sl.get(i)); //skriver ut strängen på platsen i`
		`Obs: 0<=i<=sl.size()-1`

Ta bort ett element	I en vektor kan man bara skriva över värden, inte ta bort platser. Om man vill ta bort värdet på index i kan man t.ex. lägga det sista värdet på platsen i stället. Antag att n håller reda på hur många värden som är lagrade i vektorn just nu:
	`sl[i] = sl[n-1];`
	`sl[n - 1] = null;`
	`n--;`
	Om vektorn är sorterad behöver man istället flytta ner alla element ett steg (med en loop).
		`sl.remove(i);`
		
		remove(i) tar bort elementet på index i. Alla element som ligger efter justeras ner och får automatiskt ett nytt index. Observera att remove returnerar det borttagna värdet. Så man kan spara det om det behövs:
		
		`String s = sl.remove(i);`

loopa över listan	`for(int i = 0; i<sl.length;i++){`
	`  //Gör något med varje element`
	`  //Använd i som index`
	`}`
		`for(int i = 0; i<sl.size();i++){`
		`  //Gör något med varje element`
		`  //Använd i som index`
		`}`

loopa med for-each över listan	`for(String s: sl){`
	`  //Gör något med varje element`
	`s`
	`}`
		`for(String s: sl){`
		`  //Gör något med varje element s`
		`}`

LUNDS UNIVERSITET

	Vektor	Lista

Söka upp ett värde	`String s = … //Låt s vara det vi letar efter`
	`for(int i = 0; i<sl.length;i++){`
	`  if(sl[i].equals(s)){`
	`  //Nu har vi hittat det`
	`  }`
	`}`
	`//Nu vet vi att vi inte har hittat s i listan sl.`
	
	`Observera att:`
	• `Vi måste leta genom hela vektorn för att veta att det inte finns.`
	• `Vi måste använda equals när det är objekt i vektorn`
		`String s = … //Låt s vara det vi letar efter`
		`for(int i = 0; i<sl.size();i++){`
		`  if(sl.get(i).equals(s)){`
		`  //Nu har vi hittat det`
		`  }`
		`}`
		`//Nu vet vi att vi inte har hittat s i listan sl.`
		
		`Observera att:`
		• `Vi måste leta genom hela vektorn för att veta att det inte finns.`
		• `Vi måste använda equals när det är objekt i vektorn`
		• `Lösningen är väldigt lik den för vektorer. Kolla detta så ni inser vilka skillnader som finns.`

Byta plats på två värden	Låt säga att vi ska byta plats på värdena på index i och index k:
	
	`String temp = sl[i];`
	`sl[i] = sl[k]; //Nu kan vi skriva över i eftersom vi sparat undan värdet`
	`sl[k] = temp; //Nu kan vi lägga det värde som tidigare låg på i på platsen k.`
		Låt säga att vi ska byta plats på värdena på index i och index k. Om vi ska översätta exemplet från vektorn rakt av så blir det:
		`String temp = sl.get(i);`
		`sl.set(i, sl.get(k));`
		`sl.set(k, temp);`
		Men det är egentligen lite onödigt då set returnerar det överskrivna elementet. Detta går alltså att göra kortare:
		`String temp = sl.set(i,sl.get(k));`
		`sl.set(k,temp);`

Ändra storlek	Man kan inte ändra storlek på en befintlig vektor. Då måste man skapa en ny vektor med den nya storleken och kopiera över alla gamla värden till den nya	En ArrayList har automatiskt den storlek som behövs. Exempelvis: Har man stoppat in tre element i listan så är storleken tre. Tar man bort ett så minskar storleken automatiskt med ett.

Snabbguide – matriser

Koden nedan skiljer sig beroende på vad som lagras i matrisen. I exemplet väljer jag att lagra Sträng objekt. Men det kan lika gärna vara heltal, Turtle-objekt eller vad som helst.

	Matris
Deklarera och skapa	`String [][] sl = new String[10][10];` Man måste ange storleken när man skapar en matris. Först anger man antalet rader och sen antalet kolumner. Om man inte lagt in några värden så ligger det null på alla platser om vektorn är till för att lagra objekt (som här). Om det är en int[ ] ligger det 0 på alla platser och för en boolean[ ] ligger det false på alla platser till att börja med
ta reda på antalet rader	`sl.length`
ta reda på antalet kolumner	`sl[0].length => ger antalet kolumner på rad 0 (om alla rader har samma antal kolumner räcker detta).` `sl[i].length => ger antalet kolumner på rad i`

Sätta in ett element	`sl[i][k]="Hej";`
	Man går alltid via ett index för att sätta in ett värde. Man anger alltid först radens index och sen kolumnens index. Index börjar alltid på noll. Det man sätter in behöver förstås inte vara ett fast värde, det kan lika gärna vara en strängvariabel som kommer var som helst ifrån:
	`String s = scan.next();`
	…
	`sl[i][k] = s;`
	Obs: ***0<=i<=sl.length-1 och 0<=k<=sl[i].length-1***

	Matris

Exempel på att använda ett element	`String s = sl[i][k];//Sparar undan det som ligger på raden i , kolumnen k`
	`System.out.println(sl[i][k]); //skriver ut strängen på platsen i,k`
	
	Obs: ***0<=i<=sl.length-1 och 0<=k<=sl[i].length-1***

Ta bort ett element	I en matris kan man bara skriva över värden, inte ta bort platser. Om man tillåter att det ligger null i matrisen kan man givetvis skriva över med null (funkar bara för objektstyper, inte primitiva typer) men då måste man ta hänsyn till det när man använder matrisen. Så att man inte använder element som är null.

loopa över matrisen	`for(int i = 0; i<sl.length;i++){`
	`  for(int k = 0; k<sl[i].length; k++){`
	`  //Gör något med varje element`
	`  //Använd i som rad-index och k som kolumn-index. Men döp helst`
	`  //indexen till något bättre än i och k.`
	`  }`
	`}`

loopa med for-each över listan	`for(String[] rad: sl){`
	`  for(String s: rad){`
	`  //Gör något med varje element s`
	`  }`
	`}`

LUNDS UNIVERSITET

	Matris

Söka upp ett värde	`String s = … //Låt s vara det vi letar efter`
	`for(int i = 0; i<sl.length;i++){`
	`  for(int k = 0; k<sl[i].length; k++){`
	`  if(sl[i][k].equals(s)){`
	`  //Nu har vi hittat det`
	`  }`
	`}`
	`//Nu vet vi att vi inte har hittat s i listan sl.`
	
	`Observera att:`
	• `Vi måste leta genom hela matrisen för att veta att det inte finns.`
	• `Vi måste använda equals när det är objekt i matrisen`

Byta plats på två värden	Låt säga att vi ska byta plats på värdet på platsen i, k med värdet på platsen row, col:
	
	`String temp = sl[i][k];`
	`sl[i][k] = sl[row][col]; //Nu kan vi skriva över i,k eftersom vi sparat undan värdet`
	`sl[row][col] = temp; //Nu kan vi lägga det värde som tidigare låg på i,k på platsen row,col`

Ändra storlek	Man kan inte ändra storlek på en befintlig matris. Då måste man skapa en ny matris med den nya storleken och kopiera över alla gamla värden till den nya. Det är inte givet var de gamla värdena då ska ligga…det beror ju på vad man skriver för program och varför man skulle behöva ändra storlek.

LUNDS UNIVERSITET