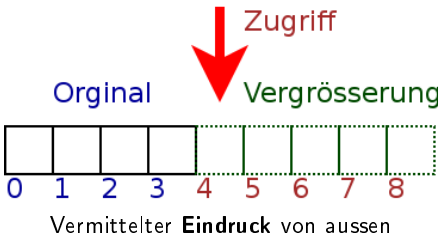
4 Selbstwachsendes Array

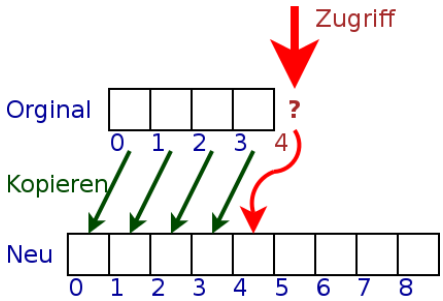
Bisher haben Sie schon einiges an Erfahrung mit Arrays gesammelt. Ist die Grösse eines Arrays-Objekts einmal festgelegt, lässt sie sich nicht mehr ändern. Hält man sich nicht daran, also wenn man über das Array hinausschreibt oder liest, bedankt sich die Java-VM mit einer `Exception`.

In dieser Serie aber soll ein Array von `int`-Zahlen implementiert werden, das die "Illusion" von adaptivem Wachstum erzeugt. Wird unverhofft mehr Platz im Array benötigt, behelfen wir uns mit einem einfachen Trick: Es wird ein **neues grösseres** Array alloziert und das alte Array an den Anfang des neuen Arrays **kopiert**.

Indem wir dem Benutzer der Klasse `GrowingArray` nur Methoden-Aufrufe anstelle von direktem Zugriff auf das Array zur Verfügung stellen, können wir in den Methoden die nötige Arbeit erledigen, falls mehr Platz als verfügbar beansprucht wird.



Vermittelter **Eindruck** von aussen



Realisation hinter den Kulissen

4.1 Implementation

a) Erstellen Sie ein neues Projekt z. B. "ArrayWork".

b) Fügen Sie dem Projekt zwei Klassen (Dateien) hinzu: `Test` (mit `main`-Methode) und `GrowingArray` (ohne `main`-Methode). Als package-Namen können Sie z. B. den Projektnamen in Kleinbuchstaben (`package arraywork;`) verwenden.

c) Fügen Sie in die Klasse `GrowingArray` Daten und Konstruktor wie folgt ein:

```
1 private int[] data;
2
3 public GrowingArray()
4 {   data = new int[10]; // default size 10
5 }
```

Das `private` vor dem Feld `data` ist Teil unseres "Tricks". Der Benutzer ausserhalb der Klasse soll nicht direkt auf die Datenstruktur zugreifen. Dadurch können wir diese Ändern, ohne dass sich am Zugriff von aussen etwas ändert. Das nennt sich **Information Hiding**.

d) Als nächstes Schreiben wir die Hilfsmethode: `private void resize(int minLength)`. Sie soll das Neu-Anlegen des Arrays erledigen, mit dem wir die Vergrösserung erreichen. Der Parameter `minLength`, gibt die neue Mindestgrösse an.

1. An erster Stelle soll die in `data` gespeicherte Referenz in eine neue Variable `int[] tmp` kopiert werden. Das geht mit einer einfachen Zuweisung.

2. Nun erstellen wir mittels `new` ein neues Array-Objekt. Die Referenz auf das neue Objekt weisen wir `data` zu. Als Länge für das neue Array wählen wir `minLength*2`. Durch die Reserve erreichen wir, dass ein Vergrössern des Arrays nicht zu oft nötig ist.
3. In einer `for`-Schleife wollen wir nun die Elemente vom alten Array (jetzt `tmp`) in das neue Array (`data`) kopieren. Die Schleife soll von `0` bis `tmp.length-1` (einschliesslich) laufen, also restlos alle Elemente von `tmp` abdecken. In der Schleife wird das jeweilige Element von `tmp` dem zugehörigen Element von `data` an der gleichen Stelle zugewiesen.

e) Nun folgt die Methode `public int get(int ind)` zum Auslesen einer Zahl aus dem Array. Der Parameter `ind` bezeichnet den Index des gewünschten Elements.

1. Nun müssen wir prüfen, ob der gewünschte Index grösser oder gleich `data.length` ist.
2. Falls dies der Fall ist, müssen wir das Array vergrössern! Dies ist dank unserer Hilfsmethode `resize` nun ein Einfaches. Wir geben `resize` den Parameterwert `ind+1` mit, so dass es auf jeden Fall Platz für das gewünschte Element hat.
3. Nun können wir mit `return` getrost den gewünschten Wert von `data[ind]` zurückliefern. Die Java-VM initalisiert den Wert für uns mit 0.

f) Als letzte Methode benötigen wir `public void set(int ind, int val)`, um das Schreiben der Werte zu ermöglichen.

1. Die ersten Zeilen, zum Prüfen, ob `ind` den zulässigen Bereich übersteigt und dementsprechendem Vergrössern des Arrays, können Sie aus `get` kopieren.
2. Danach folgt die eigentliche Zuweisung des zweiten Parameters `val` an den gewünschten Platz im Array `data[ind]`.

g) Das ist alles, was unsere schlaue Klasse benötigt. Überprüfen Sie ob Eclipse keine Fehler anzeigt.

4.2 Test

In Ihrer `public class Test` liegt die `main`-Methode schon bereit. Hier wollen wir die Klasse `GrowingArray` erstmal testen. Dazu wollen wir die Zahlen von 0 bis 123456788 schreiben und lesen.

a) Mit `GrowingArray ga = new GrowingArray();` erstellen Sie ein Objekt der zu testenden Klasse.

b) Erstellen Sie eine `for`-Schleife für 123456789 Iterationen.

c) Schreiben Sie mittels der `ga.set`-Methode in dieser `for`-Schleife stets den Wert der Laufvariablen z. B. `i` in das `i`-te Element von **ga**.

d) Wenn alles funktioniert enthält ga nun eine geordnete Liste aller Zahlen von 0 bis 123456788. *Nebenbei: Bis hierhin musste die Arraygrösse intern 26 Mal geändert werden.*

e) Ob alle Werte wirklich stimmen, wollen wir jetzt prüfen! Aber von Hand, dauert das viel zu lange :-).

f) Erstellen Sie eine identische `for`-Schleife (0 bis 123456788).

g) In jeder Iteration überprüfen Sie, ob der `i`-te Wert in ga dem Wert der Laufvariablen `i` entspricht.

h) Falls sich die Werte nicht entsprechen, geben Sie mit `System.out.println("...")` eine Fehlermeldung aus.

i) Läuft das Programm ohne Fehler durch, funktioniert ihr Code. **Herzliche Gratulation!**

j) **Zusatzfrage:** In welcher Grössenordnung alloziert die Klasse maximal Speicher in Abhängigkeit vom höchsten verwendeten Index m?

4.3 Vergleich

Diese Teilaufgabe ist **freiwillig**. Sie bietet die Gelegenheit, Ihre selbstgebaute Klasse mit vordefinierten Klassen zu vergleichen.

Falls ihr `GrowingArray` noch nicht wie gewünscht läuft, sie aber zwischendurch etwas Abwechslung brauchen, können Sie diese Aufgabe auch ohne `GrowingArray` probieren.

a) Fügen Sie

```
import java.util.ArrayList;
import java.util.Vector;
import java.util.LinkedList;
```

b) Nun wollen wir die Main-Methode etwas anpassen. Fügen Sie folgende Objekte hinzu:

```
int[] arr = new int[12345680];
GrowingArray ga = new GrowingArray();
ArrayList<Integer> al = new ArrayList<Integer>();
Vector<Integer> vec = new Vector<Integer>();
LinkedList<Integer> ll = new LinkedList<Integer>();
```

c) Definieren Sie folgende Variablen: `long time, duration;`

d) Verwenden Sie nur die `for`-Schleifen für das Beschreiben des Arrays. Kommentieren Sie den Rest mit `/* ... */` aus.

e) Verwenden Sie nur 12345678 als maximalen Wert. Das Programm dauert sonst zu lange.

f) Vervielfältigen Sie diese `for`-Schleife für alle 5 obenstehend genannten Objekte. Bei Listen heisst der dem `set`-Befehl entsprechende Aufruf `add`.

g) Wenn Sie die jeweiligen `for`-Schleifen wie folgt "einpacken", zeigt Ihnen Ihr Programm nach jeder Berechnung die verflossene Zeit in Milisekunden an.

```
time = System.currentTimeMillis();
... // for-loop
duration = System.currentTimeMillis() - time;
System.out.println("Vector: "+duration);
```

h) Vergleichen Sie die Resultate und erstellen Sie eine Rangliste. Machen Sie dazu je $3-5$ Messungen.

i) Was stellen Sie fest?

4.4 Einsatz als Stack

Für alle, die noch etwas tüfteln möchten, haben wir zum Schluss folgende **freiwillige** Aufgabe:

Ein *"reiner"* **Stack** erlaubt nur den Zugriff auf das oberste Element. Als wäre es bei einem Stapel Papier unmöglich Seiten aus dem Stapel zu zupfen. Ein neues Element kann oben auf Stack gelegt werden (`push`) und das jeweils oberste Element kann entfernt werden (`pop`).

Nebenbei: *Der Stack in der Speicherverwaltung (siehe Übungsvorlesung letzte Woche) ist kein "reiner" Stack in diesem Sinne. Es muss dort möglich sein, auf das zweitoberste, drittoberste, ... Element zuzugreifen. Dies aber nur am Rande (aus Rache, weil mir das jahrelang niemand gesagt hat).*

Erstellen Sie im gleichen Projekt eine weitere Klasse "`ArrayStack`".
Die Klasse soll folgende Struktur haben:

```
1 public class ArrayStack
2 {   private GrowingArray ga;
3     private int top;
4
5     public ArrayStack() ...
6     public void push(int v) ...
7     public int pop() ...
8 }
```

Hinter den Kulissen (`private`) wird unser `ArrayStack` als Speicher für den Stack eingesetzt. Sie können selbstverständlich alternativ auch `ArrayList`, `Vector` oder `LinkedList` einsetzen/ausprobieren.

Aufgabe: Implementieren Sie den Konstruktor und die beiden Methoden `push` und `pop`. Wenn auf den leeren Stack ein pop ausgeführt wird, können Sie eine Fehlermeldung ausgeben und -1 zurückliefern.

Tipp: In der Variable `top` speichern Sie stets die Position (also den Array-Index) des obersten Stack-Elements. Der Wert von `top` muss dazu bei jeder `push`- und `pop`-Operation entsprechend angepasst werden.

Um die Array-Grösse müssen Sie sich **keine Sorgen** machen, das erledigt die verwendete Klasse (`GrowingArray` oder was auch immer). Dieses Verhalten der positiven "Ignoranz" nennt sich **Abstraktion**. **Abstraktion** hilft sich auf das **Wesentliche** zu fokussieren, anstatt von unergründlichen Tiefen verschlungen zu werden.

Ändern Sie die `main`-Methode zu folgendem simplen Testcode:

```
1 ArrayStack as = new ArrayStack();
2 as.push(1);
3 as.push(2);
4 as.push(3);
5 System.out.println(as.pop());
6 System.out.println(as.pop());
7 System.out.println(as.pop());
8 System.out.println(as.pop());
```

Wenn ihr Stack korrekt funktioniert müsste die Ausgabe lauten:

```
3
2
1
Error, pop on empty stack
-1
```

Viel Glück & Erfolg!