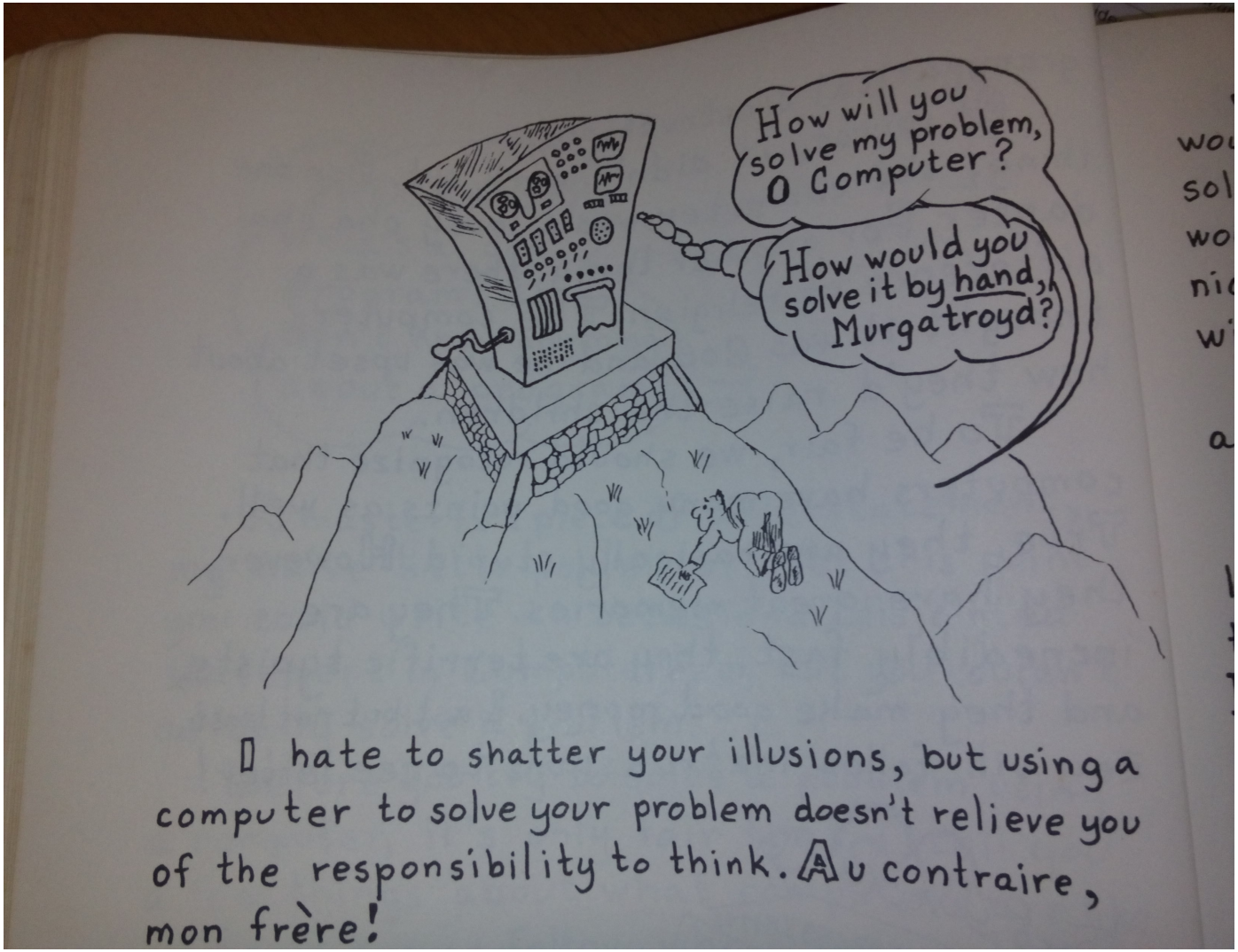
Programmieren



Arrays in Java

- Arrays sind eine Möglichkeit, um mehrere gleichartige Daten zu verwalten.
- Ein Array stellt eine Liste *fixer Größe* dar, wobei die Listenelemente von beliebigem Datentyp sein können.
- Alle Elemente eines Arrays haben aber denselben Datentyp.
- Arrays in Java sind eine spezielle Objektart.
- Bei der Erzeugung eines Array-Objekts wird seine Größe (=Anzahl der Elemente) festgelegt.
- Zum Programmieren mit Arrays gibt es eine spezielle Notation.

Definition einer Array-Variablen

Datentyp[] v*ariablenname*;

Z.B.: int[] a;
Auto[] fuhrpark;

- Die Elemente eines Arrays können von primitiven oder nicht-primitiven Datentyp sein.
- Bei einem nicht-primitiven Datentyp sind die Elemente des Arrays nicht die Objekte selbst, sondern Objekt-Referenzen.

Erzeugen eines Array-Objekts

new *Datentyp*[*Größe*]

Z.B.:

```
int[] zahlenliste;
zahlenliste = new int[24];
int n = 100;
Auto[] fuhrpark = new Auto[n];
```

- Der Aufruf des Konstruktors erzeugt ein Array-Objekt der angegebenen *Größe* mit dem angegebenen *Datentyp*.
- Die Elemente des Arrays werden mit Default-Werten initialisiert:
  - Objekt-Referenzen werden auf *null* gesetzt.
  - *int*-, *long*-, *double*-Werte werden auf 0 gesetzt.
  - *boolean*-Werte werden auf *false* gesetzt.

Erzeugen eines Array-Objekts

Ein Array kann auch gleich mit Werten befüllt werden:

Z.B.:

```
int [] zahlenarray = {3, 1, 4};
String [] stringarray = {"drei","eins","vier"};
Auto [] fuhrpark = new Auto []{a1, a2, a3 };
```

Zugriff auf ein Array-Objekt

- Auf die Elemente eines Arrays kann mit Hilfe ihres Indizes zu gegriffen werden:

```
int[] a = new int[10];
a[5] = 13;
a[6] = 2*a[5];
```

- In Java sind die Array-Elemente mit 0 beginnend nummeriert, also 0,1,2,…

- Auf die Größe eines Arrays kann mit

  <Array-Variable>.length

  zugegriffen werden (z.B. a.length).

- Das letzte Element eines Arrays a hat also den Index (a.length - 1).

Die *while*-Schleife in Java

```
while(Bedingung) {
    Anweisungsblock
}
```

- Die *Bedingung* ist ein Ausdruck, der *true* oder *false* liefert.
- Der *Anweisungsblock* der *while*-Schleife wird ausgeführt, wenn vor Ausführung des Anweisungsblocks die <Bedingung> *true* liefert.
- Nach Ende des Anweisungsblocks wird die <Bedingung> erneut überprüft, und der Anweisungsblock nochmals ausgeführt, wenn die Bedingung *true* ist.
- Das wiederholt sich, bis die Bedingung *false* ist.

Die *do-while*-Schleife in Java

```
do{
    Anweisungsblock
    }while(Bedingung)
```

- Funktionsweise wie *while*-Schleife.
- Läuft aber mindestens einmal durch, weil Überprüfung erst am Ende.

Information Hiding

- Die Klassen-Dokumentation enthält keine Implementierungsdetails:
  - Attribute und Methoden, die *private* sind.
  - Die Implementierung der Methoden.

Datentyp *char*

- *char* ist einer der primitiven Datentypen.
- In Variable vom Typ *char* kann man ein einzelnes Zeichen speichern.

  Beispiel:

  ```
  char x = 'u';
  ```

- Umwandlung in *String* funktioniert analog zu *int*:

  ```
  String s = "" + x;
  ```

Hardware

- Eingabegeräte: Tastatur, Maus, …
- Ausgabegeräte: Bildschirm, Drucker, …
- Prozessor (CPU): Evt. multi-core, …
  - Führt die durch ein Programm festgelegten Schritte aus.
- Hauptspeicher (RAM)
  - Beinhaltet die Daten (und die Programmschritte) die für die Programmausführung unmittelbar benötigt werden.
- Externer Speicher: Festplatte, …
  - Kann sehr große Datenmengen aufnehmen und permanent speichern.

Software

- Anwendungsprogramme
  - Powerpoint, selbstgeschriebene Programme, …
- Betriebssystem (Windows, Linux, MacOS)
  - Stellt die Basisfunktionen des Computers und seiner Komponenten zur Verfügung:
    - Eingabe
    - Ausgabe
    - Zugriffe auf den Haupt- und externen Speicher
    - Programmstart
    - …
  - In Java stehen diese Basisfunktionen z.B. über Methodenaufrufe zur Verfügung.

Der Prozessor

- Gängige Prozessoren verstehen nur relativ einfache Programmschritte:
  - Hole die Daten X aus dem Hauptspeicher
  - Führe die arithmetische Operation Y aus
  - Schreibe die Daten in den Hauptspeicher
  - Wenn das Ergebnis größer als 0 ist, mache an der Stelle Z des Programms weiter

- Ein Programm in einer höheren Programmiersprache (wie z.B. Java) muss in solch einfache Programmschritte umgewandelt werden.

Compiler

- Um ein Programm in einer höheren Programmiersprache (z.B. Java) auf einem Computer auszuführen, wird das Programm in eine computergerechte Form umgewandelt.
  - Komplizierte Anweisungen werden in einfache Instruktionen für den Prozessor umgewandelt.
  - Speicheradressen für Objekte, Attribute, Variablen, … werden festgelegt.
  - Die Werte von Datentypen werden in eine Binärdarstellung umgewandelt.
- Diese Aufgabe erledigt im wesentlichen der Compiler.

Speicher und Adressen

- Um Daten im Speicher (Haupt- oder externer Speicher) abzulegen, muss der genaue Platz im Speicher, die *Adresse*, angegeben werden.
- Ebenso muss beim Lesen von Daten aus dem Speicher die die *Adresse* angegeben werden.

- In einer höheren Programmiersprache wie Java werden Adressen meist nicht explizit angegeben, sondern es werden Variablen- , Parameter- und Attributnamen verwendet.
- Diese werden automatisch in die richtigen Speicheradressen übersetzt.

Speicher und Adressen

- Von jedem Java-Objekt sind die Datentypen seiner Attribute bekannt.
- Daher kann der Speicherplatzbedarf berechnet werden.
- Wenn ein neues Objekt angelegt wird, wird dieser Speicherplatz (an irgendeiner Stelle des Hauptspeichers) reserviert.
- Der Konstruktor initialisiert dann die Attributwerte des Objektes.

Primitive vs. Object Types

- Objekt-Referenzen werden verwendet, wenn ein Attribut (oder lokale Variable) keinen primitiven Datentyp, sondern ein Objekt bezeichnet.

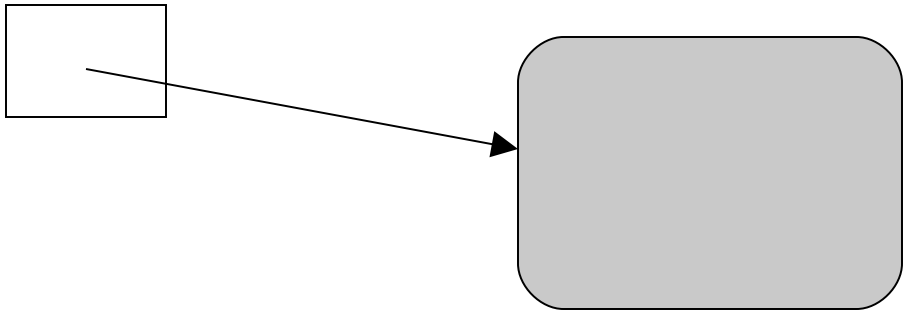
- Der Wert des Attributs ist dann nicht das Objekt selbst, sondern eine Referenz auf das Objekt („welches Objekt ist gemeint?“).
Das hat insbesondere Auswirkungen beim Zuweisen von Objekten (eigentlich Objekt-Referenzen) und dem Kopieren von Objekten.

- Nur Attribute (und Variablen) von einem primitiven Datentyp enthalten den entsprechenden Wert.

- Alle Attribute von anderem Datentyp enthalten als Wert nur Objekt-Referenzen.

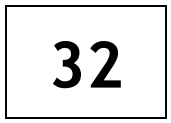
Primitive types vs. object types (1)

```
SomeObject obj;
```



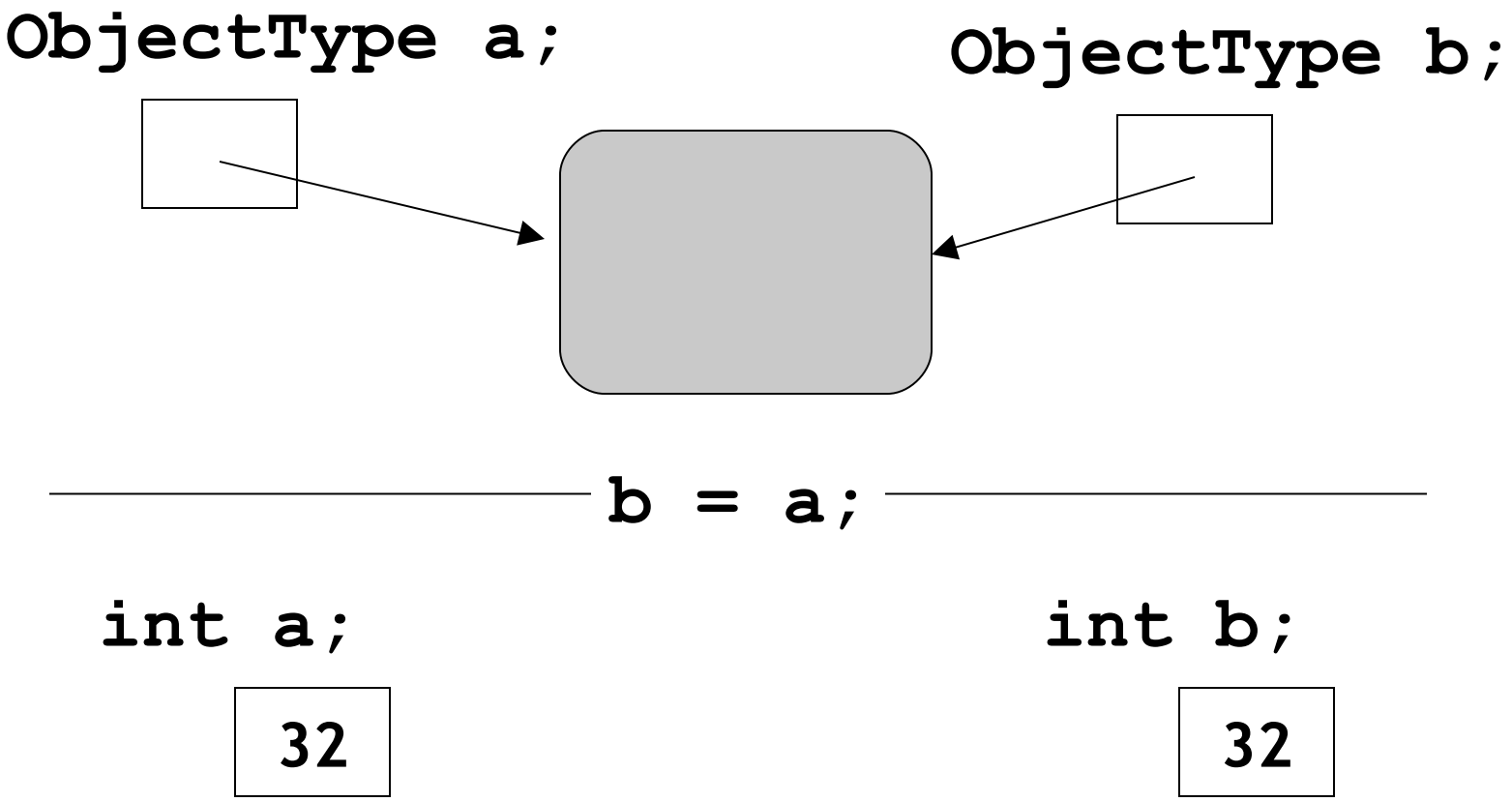
object type

```
int i;
```

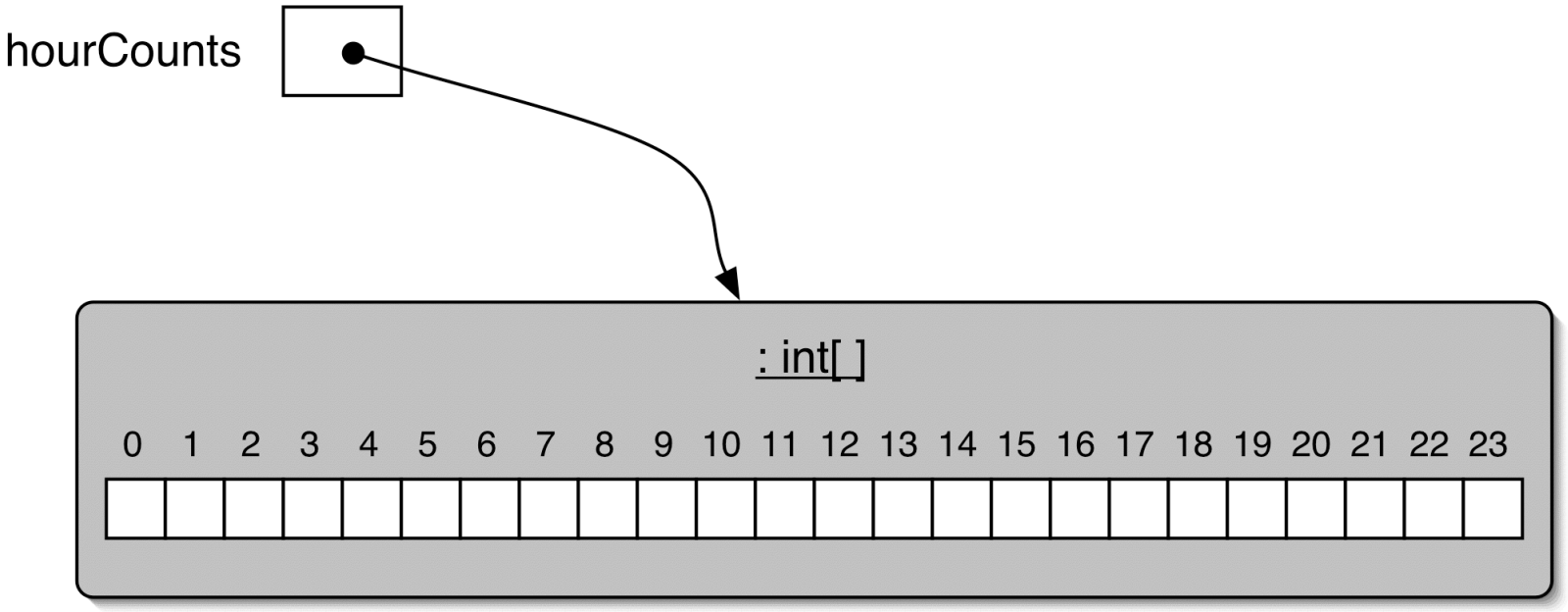


primitive type

Primitive types vs. object types (2)



Arrays


hourCounts
: int[ ]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23