Recap: Loops

- Repeating things

Repeat as long as something holds

Pattern

```
1 while (EXPRESSION) {
2   // some code
3 }
```

Example

```
1 String userInput;
2 while (userInput != "exit") {
3   // some code
4   // ask user for command
5   userInput = ...
6 }
```

Repeat for specified number of times

Pattern

```
1 for (INIT; CONDITION; UPDATE) {
2   // some code
3 }
```

Example

```
1 for (int i = 0; i < 17; i++) {
2   // some code
3 }
```

demo

Exercise 5

UNIVERSITÄT
ZU KÖLN

Session 6: Arrays ~~and Strings~~

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

November 20, 2024

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Arrays

Introduction

- So far: Single variables store single values
  - `int i = 5; //one int value in one int variable`

Introduction

- So far: Single variables store single values
  - `int i = 5; //one int value in one int variable`
- New: Array (German, rarely used: “Feld”)
  - Stores a collection of values – a data structure
  - Fixed number of values
  - All values have the same type

Introduction

- So far: Single variables store single values
  - `int i = 5; //one int value in one int variable`
- New: Array (German, rarely used: “Feld”)
  - Stores a collection of values – a data structure
  - Fixed number of values
  - All values have the same type
  - New syntactic element: square brackets `[]`

Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-based: The first component has the index number 0

Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-based: The first component has the index number 0

Example

```
1 arr[0] // the first component of arr
2 arr[2] // the third component of arr
```

Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-based: The first component has the index number 0

Example

```
1 arr[0] // the first component of arr
2 arr[2] // the third component of arr
```

- Array components can be used in expressions, similar to variable names

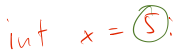
Example

```
1 (arr[0]) = 5;
2 int b = (arr[2]) + 4;
```

Array Creation

- With `new`
  - `int[] a = new int[5];`
  - Filled with 0s

Array Creation



- With `new`
  - `int[] a = new int[5];`
  - Filled with 0s
- As literal – similar to writing fixed numbers in the code
  - `int[] a = new int[] {1, 2, 3};`
  - `someFunction(new int[] {1,2,3})` – literal array as argument

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

- There is no way to increase the length
  - ...except to create a new array and copy items from the old to the new

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

- There is no way to increase the length
  - ...except to create a new array and copy items from the old to the new
- Because the length is important, there is a way to access it: `arr.length`

demo

ArrayDemo

Array as a Type

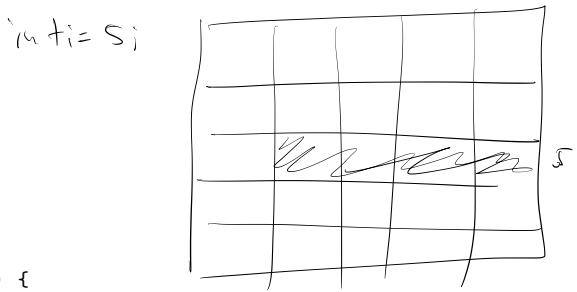
- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`

Array as a Type

- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`
- Length is not part of the type
  - I.e., not known at compile time

Array as a Type

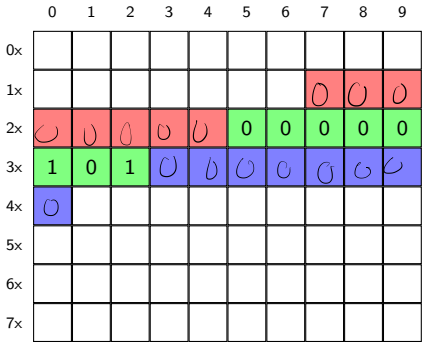
- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`
- Length is not part of the type
  - I.e., not known at compile time




```
1 public static void main(String[] args) {
2   // ...
3 }
```

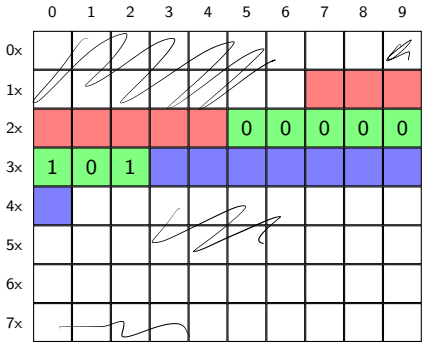
- As `main` is a function, `args` is an argument of type `String[]`
  - ➜ A collection of character sequences

Arrays in Memory




```
1 byte[] bArray = new byte[3];
2 bArray[1] = (byte) 5; // green cells
```

Arrays in Memory




```
1 byte[] bArray = new byte[3];
2 bArray[1] = (byte) 5; // green cells
```

- Exact address of each component can be computed
- If starting address and length of each component are known
- "Direct access" (= fast)

Session 2
October 16

Primitive Data Types and Objects

Two kinds of types

- Primitive data types: Built into the language
  - Type names are reserved keywords in Java
    - I.e., it's not allowed to use them as an identifier
  - Convention: Lower cased
- Non-primitive data types ("reference types"): Established in the library
  - Type names are defined by library authors
  - Convention: Upper cased
  - Reference types can also be defined by us
    (in the form of classes, to be discussed later)

Array is a Reference Type

```
1  // Primitive type
2  int x = 5;
3  int y = x;
4  y = y + 2; // y now contains 7,
5             // x still 5
6
7  // Reference type
8  int[] a = {1,2,3};
9  int[] b = a;
10 a[0] = 0; // b[0] is now also 0!
11           // because a and b are
12           // references to the same
13           // memory region
```

- Primitive types: Values (of memory regions) are passed
- Reference types: References (to memory regions) are passed
  - If you change a reference type within a function, it's changed outside of the function
- Everything from now on is a reference type

demo

ReferenceTypeDemo

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?

int a = 5
int b = 5
if (a == b) ...

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?
- If reference types are compared with `==` & co., we compare the memory location
  - Not the content
- To compare the content: `Arrays.equals(a1, a2)`

javadoc

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?
- If reference types are compared with `==` & co., we compare the memory location
  - Not the content
- To compare the content: `Arrays.equals(a1, a2)` javadoc
  - ⚠ Using some functions requires importing them first
    - Eclipse suggestions are most often correct

Array Copying

```
1 // Reference type
2 int[] a = {1,2,3};
3 int[] b = a; // does not create a copy of a
4 b[0] = 0;
5
6 int[] c = a.clone(); // creates a copy
7 c[2] = 10; // no change in a
```

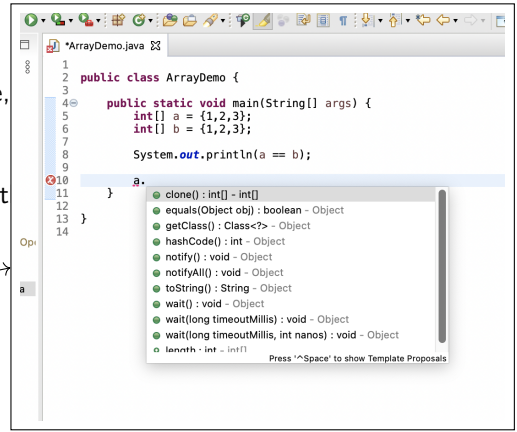
- Copying an array: `someArray.clone()`
  - This is a method (note the parentheses)

Methods and Fields

- `length` is stored with an array
  - Calling `someArray.length` does not execute code, it's just a variable access
- `clone()` is a function associated with this array
  - Calling `someArray.clone()` runs this function in the context of this array
  - Method: A function with benefits

Methods and Fields

- `length` is stored with an array
  - Calling `someArray.length` does not execute code,
- `clone()` is a function associated with this array
  - Calling `someArray.clone()` runs this function in t
  - Method: A function with benefits
- Other methods are displayed by Eclipse



Array Patterns

(Code snippets that are frequently needed)

Printing out an array:

```
1 System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Array Patterns

(Code snippets that are frequently needed)

Printing out an array:

```
1 System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Iterating over an array:

```
1 for (int i = 0; i < array.length; i++) {
2   // access each array element with array[i]
3 }
```

Array Patterns

(Code snippets that are frequently needed)

Printing out an array:

```
System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Iterating over an array:

```
for (int i = 0; i < array.length; i++) {
  // access each array element with array[i]
}
```

Two-dimensional array (= a matrix or table):

```
int[][] matrix = new int[17][25];
matrix[0][0] = 15;
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    // cells can be accessed with matrix[i][j]
  }
}
```

Section 2

Exercise