Session 6: Arrays and Strings

Softwaretechnologie: Java I

Nils Reiter
`nils.reiter@uni-koeln.de`

November 16, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Exercise 5

Today

- New things
  - Arrays to store collections/sequences of things
  - Strings to store sequences of characters
- New concepts
  - Arrays and strings are reference types
  - First signs of object orientation

Section 2

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`

Array

- Stores a collection of values
- Number of values is fixed
- All values are of the same type

Introduction

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

Array

- Stores a collection of values
- Number of values is fixed
- All values are of the same type
- Syntax: square brackets `[]`
  - `int[] arr = new int[5]; //five int values`

Using Arrays

- Array components are enumerated (0-base)

```
arr[0] //the first component of arr
```

```
arr[2] //the last component of arr, if arr has 3 components
```

Using Arrays

- Array components are enumerated (0-base)

```
arr[0] //the first component of arr
```

```
arr[2] //the last component of arr, if arr has 3 components
```

- Components can be used in expressions, similar to variable names

```
arr[0] = 5;
```

```
int b = arr[2] + 4;
```

Array Length

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

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

Array Length

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

```java
int a = 5;
a = a + (int) Math.random();
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

```java
int a = 5;
a = a + (int) Math.random();
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

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

```java
public static void main(String[] args) {
  // ...
}
```

args.length

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

Array Creation

Reference Type

- With `new`
  - `int[] a = new int[5];`
  - Filled with `0`
- As literal
  - `int[] a = new int[] {1, 2, 3};`
    - In this case, the type can be inferred, so we can skip `new int[3]`: `int[] a = {1, 2, 3};`
  - `someFunction(new int[] {1,2,3})` – literal array as argument

demo

Session 3
October 26

Primitive Data Types and Objects

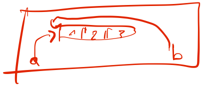
- Two kinds of types
  - Primitive data types: Built into the language
    - Type names are reserved keywords in Java
    - Convention: Lower cased
  - Non-primitive data types ("reference types"): Established in the library
    - Type names are defined by library authors
    - Convention: Upper cased
    - Reference types can also be defined by us
      (in the form of classes, to be discussed later)

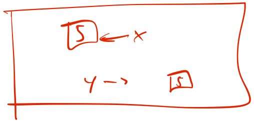
Array is a Reference Type

```java
// Primitive type
int x = 5;
int y = x;
y = y + 2; // y now contains 7,
           // x still 5

// Reference type
int[] a = {1,2,3};
int[] b = a;
a[0] = 0; // a and b are identical
```

- 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





Comparing Reference Types

```java
int[] a = {1,2,3};
int[] b = {1,2,3};

if (a == b) {
  System.out.println("Arrays are equal");
} else {
  System.out.println("Arrays are not equal");
}
```

- Which output do we get?

Comparing Reference Types

```java
int[] a = {1,2,3};
int[] b = {1,2,3};

if (a == b) {
  System.out.println("Arrays are equal");
} else {
  System.out.println("Arrays are not equal");
}
```

- 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

```java
int[] a = {1,2,3};
int[] b = {1,2,3};

if (a == b) {
  System.out.println("Arrays are equal");
} else {
  System.out.println("Arrays are not equal");
}
```

- 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 mostly correct, more on this next week

demo

Array Copying

```java
// Reference type
int[] a = {1,2,3};
int[] b = a; // does not create a copy of a
b[0] = 0;

int[] c = a.clone(); // creates a copy
c[2] = 10; // no change in a
```

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

.length

Methods and Fields

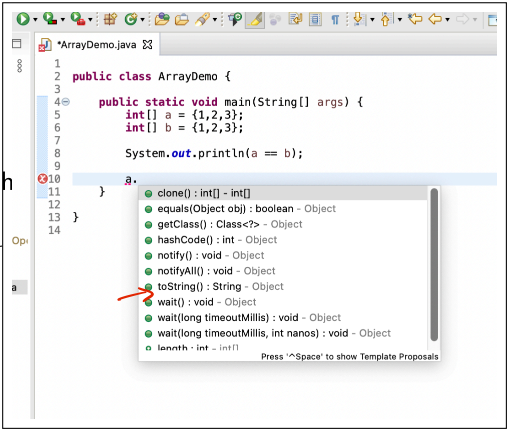
Keine runden Klammern

- `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

Code

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 th
  - Method: A function with benefits
- Other methods are displayed by Eclipse



*ArrayDemo.java

```
public class ArrayDemo {

    public static void main(String[] args) {
        int[] a = {1,2,3};
        int[] b = {1,2,3};

        System.out.println(a == b);

        a.
    }

}
```

```
clone() : int[] - int[]
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeoutMillis) : void - Object
wait(long timeoutMillis, int nanos) : void - Object
length : int - int[]
Press '^Space' to show Template Proposals
```

Array Patterns

Frequently used pattern:

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

Array Patterns

Frequently used pattern:

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

Two-dimensional array:

```java
int[][] matrix = new int[17][25];
int[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 3

Strings

Introduction

- Represents character sequences
- A reference type
- Internally: An array of `char` -values (mostly)

```java
String s = "Hi there!"; // String literal with double quotes
```

String Operations

Several regular operators have been re-defined for strings

- Concatenation

```
String s1 = "Hi";
String s2 = "there";
String s = s1 + s2; // s now contains "Hithere"
```

- `+` is the only regular operator you can use with strings

String Operations

Several regular operators have been re-defined for strings

- Concatenation

```
String s1 = "Hi";
String s2 = "there";
String s = s1 + s2; // s now contains "Hithere"
```

  - `+` is the only regular operator you can use with strings
- Length: `s.length() //returns 7 (as an int)`

String Operations

Several regular operators have been re-defined for strings

- Concatenation

```
String s1 = "Hi";
String s2 = "there";
String s = s1 + s2; // s now contains "Hithere"
```

  - `+` is the only regular operator you can use with strings
- Length: `s.length() //returns 7 (as an int)`
- Convert case
  - `s2.toLowerCase(); //returns "hi"`
  - `s2.toUpperCase(); //returns "HI"`

Strings and Other Types

- All primitive types can be converted into a string
  - `System.out.println()` does this, as we have seen
- Conversion done implicitly:

```java
int i = 2022;
String s = "Hallo";
System.out.println(s + i); // implicit conversion of i,
                           // then concatenation
```

Strings and Other Types

- All primitive types can be converted into a string
  - `System.out.println()` does this, as we have seen
- Conversion done implicitly:

```java
int i = 2022;
String s = "Hallo";
System.out.println(s + i); // implicit conversion of i,
                           // then concatenation
```

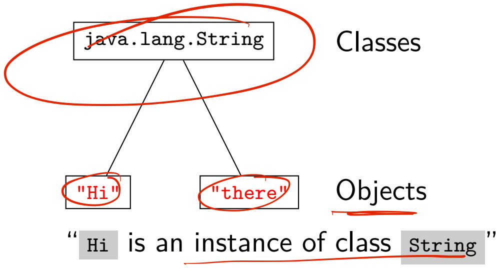
- Explicit conversion
  - Many functions `String.valueOf(ARG)`
  - Take all primitive types as arguments

int / boolean

The class String

- `java.lang.String`: Our first class
- Classes and Objects:
  Object-oriented programming

More on classes and objects: Next week(s)



`main` Function

```java
public class MyProgram
  public static void main(String[] args) {
    // do stuff
  }
}
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?

`main` Function

```java
public class MyProgram
  public static void main(String[] args) {
    // do stuff
  }
}
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?

- Command line: `java MyProgram ARG1 ARG2 ...`
  - ARG1 and ARG2 are available as arguments in `main`

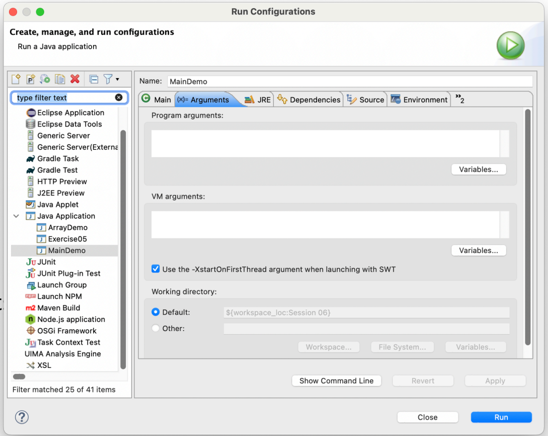
`main` Function

```java
public class MyProgram
  public static void main(String[] args) {
    // do stuff
  }
}
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?

- Command line: `java MyProgram ARG1 ARG2 ...`
  - ARG1 and ARG2 are available as argument
- Eclipse: Run → Run Configurations ⟶



Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(Externa
Gradle Task
Gradle Test
HTTP Preview
J2EE Preview
Java Applet
Java Application
ArrayDemo
Exercise05
MainDemo
JUnit
JUnit Plug-in Test
Launch Group
Launch NPM
Maven Build
Node.js application
OSGi Framework
Task Context Test
UIMA Analysis Engine
XSL

Filter matched 25 of 41 items

Name: MainDemo

Main | Arguments | JRE | Dependencies | Source | Environment | »2

Program arguments:

Variables...

VM arguments:

Variables...

[✓] Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

[● Default:] ${workspace_loc:Session 06}

○ Other:

Workspace... File System... Variables...

Show Command Line Revert Apply

Close Run

demo

What can we do with Strings?

String s = " ";

...and how do we find out?

- Javadoc
  - `char charAt(int index);` charAt(5)
  - `int compareTo(String anotherString)`
  - `String concat(String str)`
  - `boolean endsWith(String suffix)` "Hallo".endsWith("lo")
  - `boolean isEmpty()`
  - `String substring(int beginIndex, int endIndex)`
  - ...

java.lang.String

What can we do with Strings?

...and how do we find out?

java.lang.String

- Javadoc
  - `char charAt(int index);`
  - `int compareTo(String anotherString)`
  - `String concat(String str)`
  - `boolean endsWith(String suffix)`
  - `boolean isEmpty()`
  - `String substring(int beginIndex, int endIndex)`
  - ...
- How to use them? `INSTANCE.METHOD(ARGUMENTS)`
  - Eclipse suggests possible methods/fields in a small window
  - Methods are associated with the specific instance before the `.`

Section 4

Exercise