Session 12: Recap

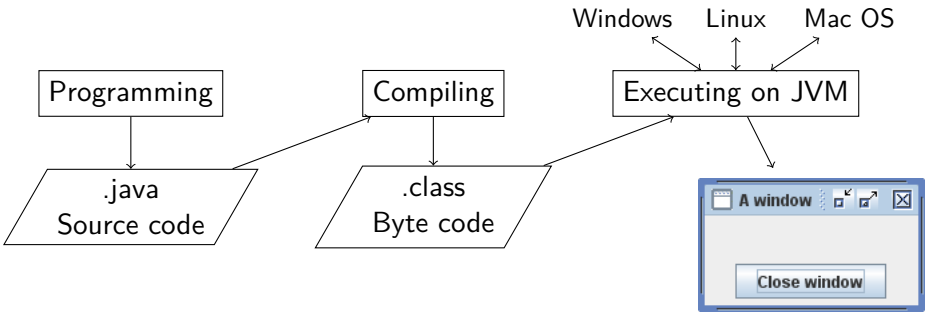
Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

January 11, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Running a Java-Program



Recap: What's a Java Program?

A sequence of statements

- Statements are:
  - Code blocks: `{ /*MORE STATE MENTS */}`
  - Assignment statement: `Dog d = new Dog();`, or `int i = 15;`
  - If statement: `if (i < 15) CODE BLOCK else CODE BLOCK`
  - While statement: `while (i < 15) CODE BLOCK`
    - Do while statement: `do CODE BLOCK while (i < 15)`
  - For statement: `for (int i = 0; i < 5; i++) CODE BLOCK`
  - Switch statement: `switch (month) { case 1: str = "January"; break; /*... */}`

Recap: What's a Java Program?

Expressions

- Many statements make use of expressions
- Expressions are
  - Literal values `5`, `true`, `'c'`
  - Variables `i`
  - Function calls `add(5, c)`
  - Operations `5 + c`

    Operators: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `<`, `>`, `<=`, `>=`, `=!`, `==`, `instanceof`, `&&`, `||`, ...

Recap: What's a Java Program?

Expressions

- Many statements make use of expressions
- Expressions are
  - Literal values `5`, `true`, `'c'`
  - Variables `i`
  - Function calls `add(5, c)`
  - Operations `5 + c`

    Operators: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `<`, `>`, `<=`, `>=`, `=!`, `==`, `instanceof`, `&&`, `||`, ...
- Expressions
  - are evaluated at run time
  - have a clearly defined type at compile time

Recap: Variables and Functions

Variables

- Placeholders for values
- Can change their value
- Are only accessible within a function block (scope)

Recap: Variables and Functions

Variables

- Placeholders for values
- Can change their value
- Are only accessible within a function block (scope)

Functions `static int myFunction(String s, boolean b) { /*... */}`

- Allow encapsulating functionality
- Dealing with complexity
- Signature: Name and typed arguments
- Return value
  - The result of a function
  - `void`: There is no return value

Recap: Types

- Variables and expressions have a type
  - Primitive types: `int`, `boolean`, `char`, `double`, `byte`, `short`, `long`, `float`
  - Reference types: `String`, `T[]` (arrays), `Horse` (classes we define), ...

Recap: Types

- Variables and expressions have a type
  - Primitive types: `int`, `boolean`, `char`, `double`, `byte`, `short`, `long`, `float`
  - Reference types: `String`, `T[]` (arrays), `Horse` (classes we define), ...
- Arrays
  - Store a list of things of the same type
  - Items are enumerated and addressed with their index number

Example

```
1 int[] iArray = new int[2];
2 iArray[0] = 14;
3 iArray[1] = 3;
4 int j = iArray[0]; // j becomes 14
```

Recap: Classes and Objects

- Core ingredients of object-oriented programming
- Define our own types to reflect our domain of interest
- Encapsulate data and behaviour
- Allows writing modularized programs

Recap: Classes and Objects

- Core ingredients of object-oriented programming
- Define our own types to reflect our domain of interest
- Encapsulate data and behaviour
- Allows writing modularized programs

Example

```
public class Address {
  String street;
  String houseNumber;
  int postcode;
  String city;
  String country;

  public String getStreetPart() {
    if (country.equalsIgnoreCase("Germany")) {
      return this.street + " " + this.houseNumber;
    } else if (country.equalsIgnoreCase("France")) {
      return this.houseNumber + ", " + this.street;
    }
  }
}
```

Recap: Classes and Objects

- Core ingredients of object-oriented programming
- Define our own types to reflect our domain of interest
- Encapsulate data and behaviour
- Allows writing modularized programs

Example

```
public class Address {
  String street;
  String houseNumber;
  int postcode;
  String city;
  String country;

  public String getStreetPart() {
    if (country.equalsIgnoreCase("Germany")) {
      return this.street + " " + this.houseNumber;
    } else if (country.equalsIgnoreCase("France")) {
      return this.houseNumber + ", " + this.street;
    }
  }
}
```

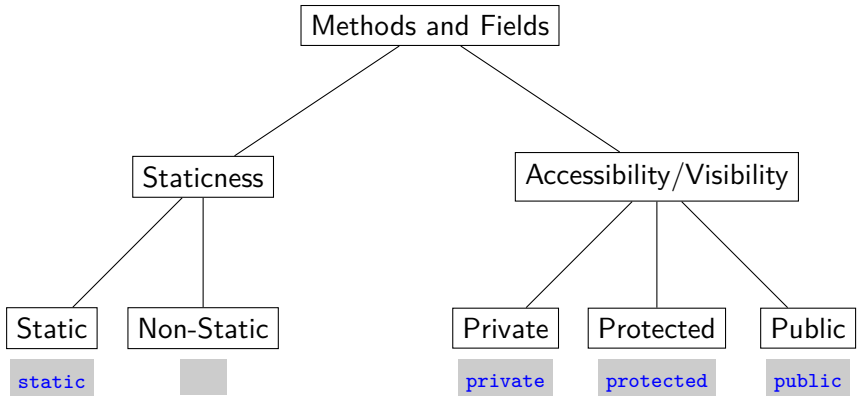
```
Address a = new Address();
a.street = "Universitätsstraße"
a.houseNumber = "22";

a.country = "Germany";
a.getStreetPart();
  // returns "Universitätsstraße 22"

a.country = "France";
a.getStreetPart();
  // returns "22, Universitätsstraße"
```

Recap: Classes and Objects

Staticness and Visibility



Recap: Classes and Objects

Inheritance

- A class may inherit from one other class
- Keyword: `extends`, used in the class declaration: `public class Horse extends Animal { ... }`
  - Horse: sub class
  - Animal: super class
- Useful to model domain hierarchies, e.g., animal species

Recap: Classes and Objects

Inheritance

- A class may inherit from one other class
- Keyword: `extends`, used in the class declaration: `public class Horse extends Animal { ... }`
  - Horse: sub class
  - Animal: super class
- Useful to model domain hierarchies, e.g., animal species
- Objects of sub class can use methods/access fields defined in super class
- Sub class may override method defined in super class

Recap: Classes and Objects

Interfaces, Abstract Classes, Abstract Methods

- Abstract classes
  - Cannot be instantiated
  - But can be inherited
- Abstract methods
  - Can only appear in abstract classes
  - An abstract method does not have a body
  - An inheriting class must provide the method body

Recap: Classes and Objects

Interfaces, Abstract Classes, Abstract Methods

- Abstract classes
  - Cannot be instantiated
  - But can be inherited
- Abstract methods
  - Can only appear in abstract classes
  - An abstract method does not have a body
  - An inheriting class must provide the method body
- Interface: A class, but
  - There are no fields
  - All methods are abstract
  - It is abstract

Interfaces vs. Abstract Classes

- Similar concepts: No instantiation, abstract methods
- Difference: A class can inherit from only one class, but can implement multiple interfaces

Interfaces vs. Abstract Classes

- Similar concepts: No instantiation, abstract methods
- Difference: A class can inherit from only one class, but can implement multiple interfaces

Multiple inheritance

- We often want to inherit from multiple classes
  - E.g., the class Dog inherits from Animal and Pet
- This is not allowed in Java
  - Because if Animal and Pet both define a method `eat()`, we don't know which is executed
- But class Dog could inherit from Animal and implement the interface Pet

```
public class Dog extends Animal implements Pet { /*... */}
```

Another Use Case for Interfaces

- Sorting things in a collection is a well-studied task in computer science
- Different approaches exist, with different time and space requirements

Another Use Case for Interfaces

- Sorting things in a collection is a well-studied task in computer science
- Different approaches exist, with different time and space requirements
- Computer science: We sort collections of numbers
  - Because numbers have an obvious true ordering (5 comes before 10)
  - And for any two numbers, we can specify, which comes first

Another Use Case for Interfaces

- Sorting things in a collection is a well-studied task in computer science
- Different approaches exist, with different time and space requirements
- Computer science: We sort collections of numbers
  - Because numbers have an obvious true ordering (5 comes before 10)
  - And for any two numbers, we can specify, which comes first
- Real world: We want to sort other things, e.g., entries in an address book!
  - If we can specify which of two entries comes first, can we use the sorting algorithm from CS?
  - Sure: Just implement this interface, and provide it to the (generic) sort function

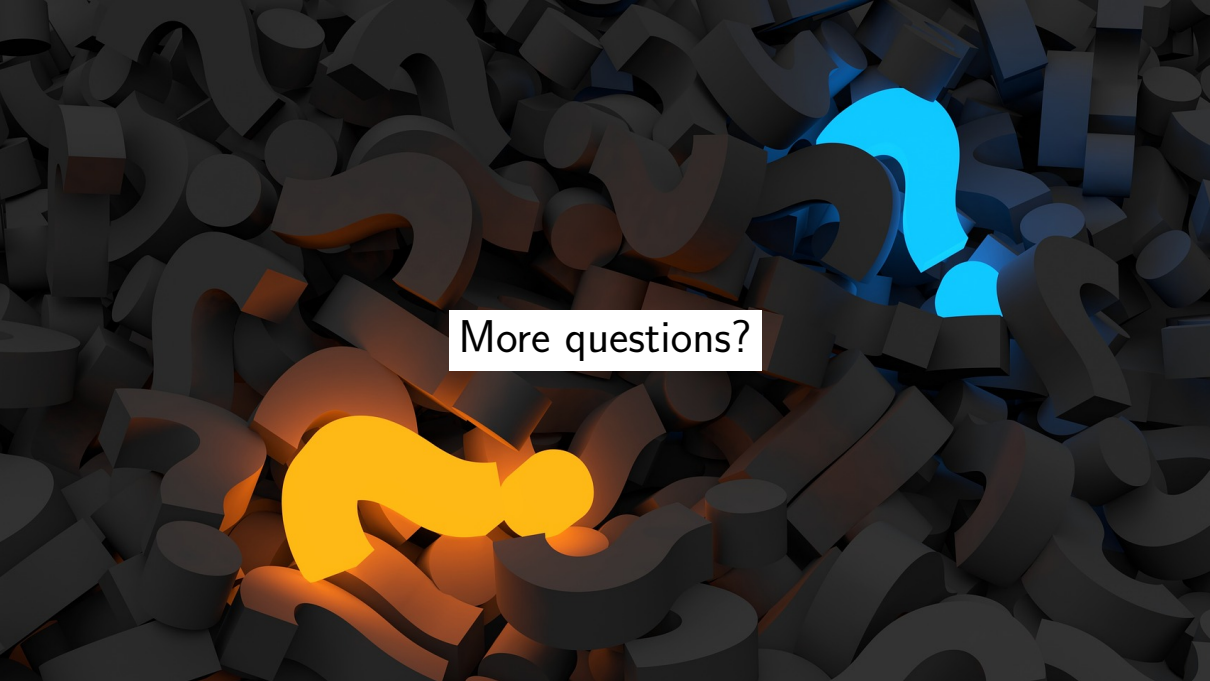
```
public interface CompareTwoItems { public int compare(Object e1, Object e2) { /*... */} }
```

Another Use Case for Interfaces

- Sorting things in a collection is a well-studied task in computer science
- Different approaches exist, with different time and space requirements
- Computer science: We sort collections of numbers
  - Because numbers have an obvious true ordering (5 comes before 10)
  - And for any two numbers, we can specify, which comes first
- Real world: We want to sort other things, e.g., entries in an address book!
  - If we can specify which of two entries comes first, can we use the sorting algorithm from CS?
  - Sure: Just implement this interface, and provide it to the (generic) sort function

```
public interface CompareTwoItems { public int compare(Object e1, Object e2) { /*... */} }
```

- This is how java.util.Arrays: sort() works


More questions?

Section 2

Exercise