Toegepaste Informatica

1TX

2022-2023

Back-End Development

Object-Oriented Programming

G. Jongen, J. Pieck, E. Steegmans, B. Van Impe

Object-Oriented Programming

OOP

Object-Oriented Programming

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code.
  - Data in the form of fields (often known as attributes or properties)
  - Code in the form of procedures (often known as methods)

- There exist a lot of OO programming languages
  - **Java**
  - C++
  - Python
  - C#
  - …

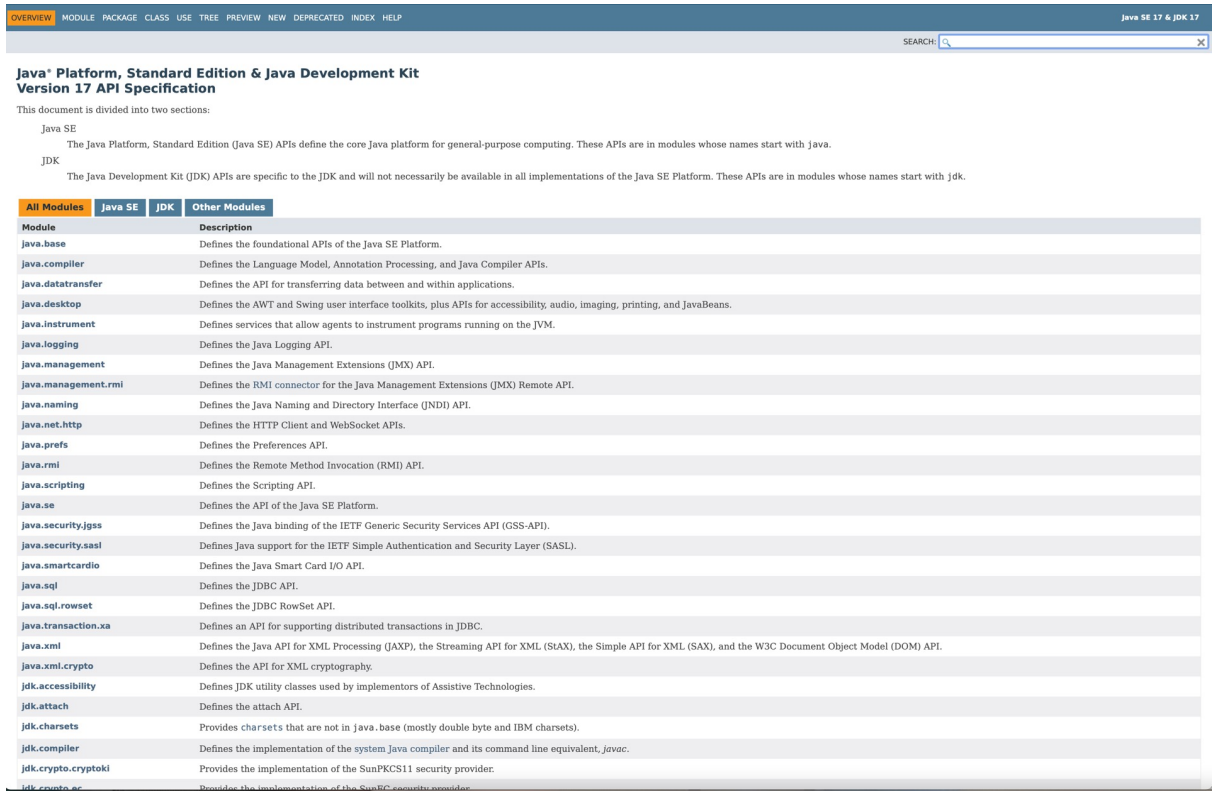
James Gosling

Version	Date
JDK Beta	1995
JDK 1.0	January 23, 1996[40]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8 (LTS)	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11 (LTS)	September 25, 2018[41]
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020[42]
Java SE 16	March 16, 2021
Java SE 17 (LTS)	September 14, 2021
Java SE 18	March 22, 2022
Java SE 19	September 20, 2022

minimum

Java API

- Java already has a lot of classes defined that you can use
  - as such you don't need to reinvent the wheel again and again and … :-)
- You can find all these classes in the Java API
  - https://docs.oracle.com/en/java/javase/17/docs/api/index.html



OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP — Java SE 17 & JDK 17

SEARCH:

Java® Platform, Standard Edition & Java Development Kit
Version 17 API Specification

This document is divided into two sections:

Java SE
The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK
The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules | Java SE | JDK | Other Modules

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.
java.rmi	Defines the Remote Method Invocation (RMI) API.
java.scripting	Defines the Scripting API.
java.se	Defines the API of the Java SE Platform.
java.security.jgss	Defines the Java binding of the IETF Generic Security Services API (GSS-API).
java.security.sasl	Defines Java support for the IETF Simple Authentication and Security Layer (SASL).
java.smartcardio	Defines the Java Smart Card I/O API.
java.sql	Defines the JDBC API.
java.sql.rowset	Defines the JDBC RowSet API.
java.transaction.xa	Defines an API for supporting distributed transactions in JDBC.
java.xml	Defines the Java API for XML Processing (JAXP), the Streaming API for XML (StAX), the Simple API for XML (SAX), and the W3C Document Object Model (DOM) API.
java.xml.crypto	Defines the API for XML cryptography.
jdk.accessibility	Defines JDK utility classes used by implementors of Assistive Technologies.
jdk.attach	Defines the attach API.
jdk.charsets	Provides `charsets` that are not in `java.base` (mostly double byte and IBM charsets).
jdk.compiler	Defines the implementation of the system Java compiler and its command line equivalent, *javac*.
jdk.crypto.cryptoki	Provides the implementation of the SunPKCS11 security provider.
jdk.crypto.ec	Provides the implementation of the SunEC security provider.

Class - Object

Demo Example Users

Overview Add User

Name	Age
Elke	44
Miyo	14
Eric	65
Yuki	12

OBJECT

name = “Eric”
age = 65

name = “Elke”
age = 44

name = “Miyo”
age = 14

name = “Yuki”
age = 12

CLASS

```
public class User {

  String name;
  int age;

}
```

CLASS DIAGRAM

User
age: int name: String

OBJECT

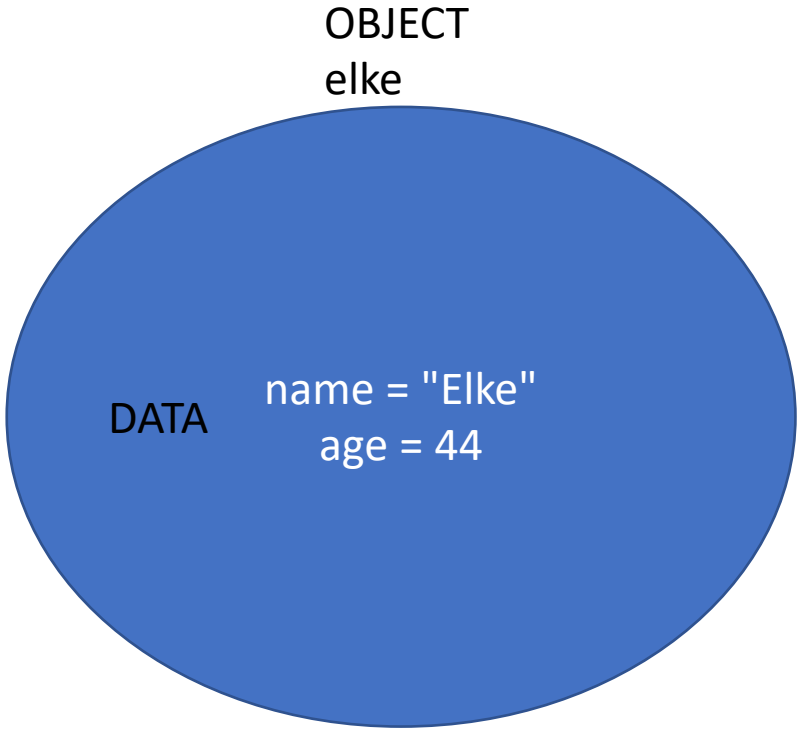
- An object holds specific values of attributes; these values can change while the program is running
- Objects created by many times.

CLASS

- Class specifies the structure (the number and types) of its objects' attributes – the same for all objects.
- Class is declared once.

CLASS DIAGRAM

- Class diagram is a visual presentation of all classes, their instance variables and their methods


OBJECT
elke
DATA
name = "Elke"
age = 44

OBJECT

```java
User elke = new User("Elke", 44);
User eric = new User();
eric.age = 65;
eric.name = "Eric";
User miyo = new User("Miyo", 14);
User yuki = new User("Yuki", 12);
```

CLASS

```java
public class User {

  public String name;
  public int age;

  public User(String name, int leeftijd) {
    this.name = name;
    age = leeftijd;
  }

  public User() {
  }

}
```

Instance Variables

Constructors

CLASS DIAGRAM

User
age: int name: String
User() User(name: String, age: int)

Class

```
public class User
{

}
```

- is a template for creating object, providing initial values for state (instance variables) and implementations of behaviour (methods)
- is declared once

Instance Variables

```
public String name;
public int age;
```

- store data of an object
- format

  ```
  accessmodifier type nameofinstancevariable
  ```

  - example

    ```
    public int age
    ```

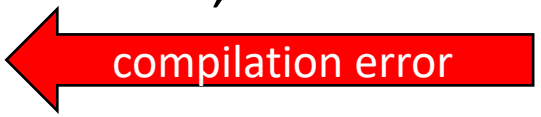
- each variable must be typed in Java!
  - Java is a strongly typed language

- JavaScript

```
let name = "Elke"
name = 1
```

- Java

```
String name = "Elke";
name = 1;
```



Constructor

```
public User(String name, int leeftijd) {
    this.name = name;
    age = leeftijd;
}
```

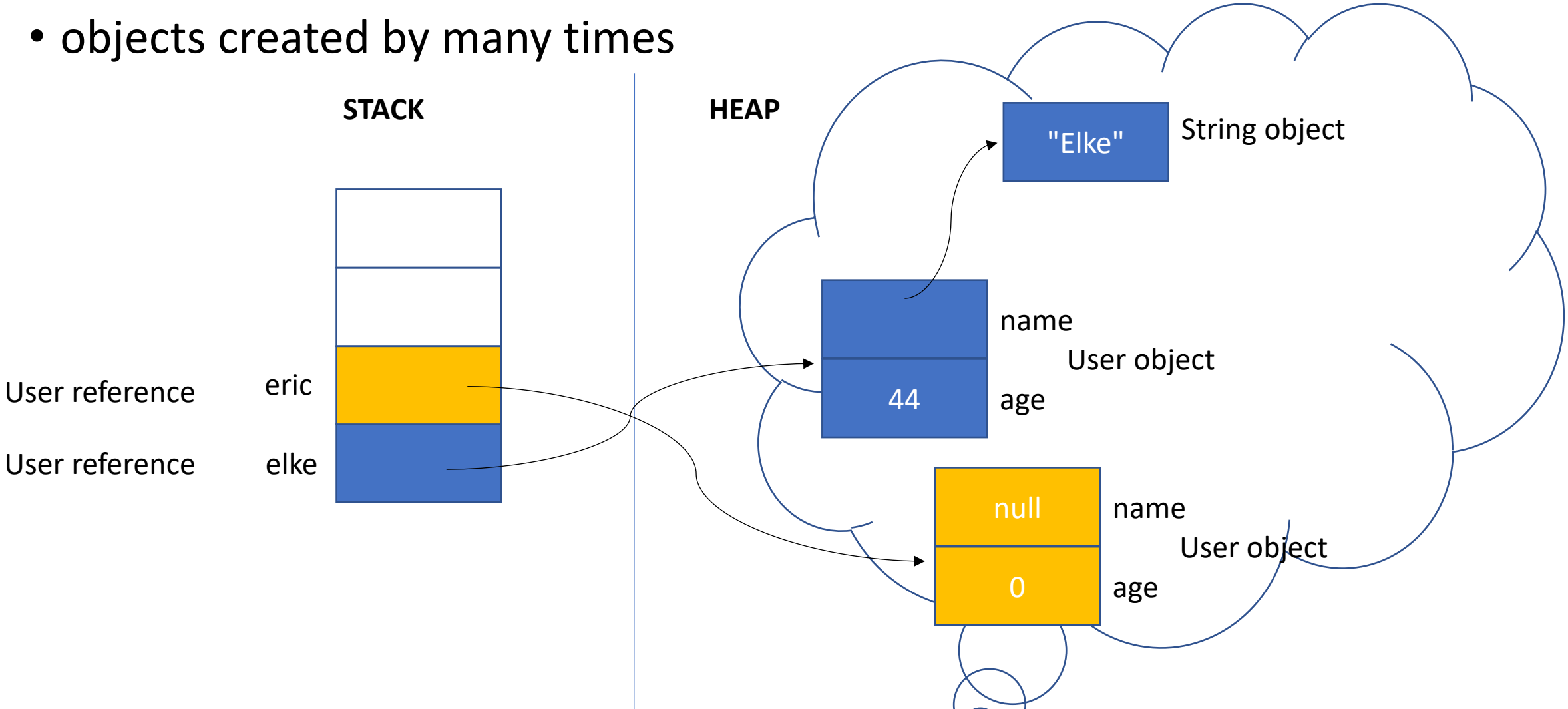
- is a method to create objects of this class
- there can be multiple constructors in a class
- the constructor with no parameters is called the default constructor
  - Then all instance variables are set to their default values
    - default value of int is 0
    - default value of String is null

```
public User() {
}
```

Object

```
User elke = new User("Elke", 44);
User eric = new User();
```

- objects created by many times



Main method

- is the starting point of a Java program
- all code within this method is executed when you run this main method

```
public static void main(String[] args) {

  User elke = new User("Elke", 44);
  System.out.println(elke.age);

  User eric = new User("Eric", 65);
  System.out.println(eric.name);

  User miyo = new User("Miyo", 14);
  System.out.println(miyo);

  User yuki = new User("Yuki", 12);
  System.out.println("User with name "
     + yuki.name + " is " +
     yuki.age + " years old");

}
```

```
44
Eric
be.ucll.ti.demo.User@7344699f
User with name Yuki is 12 years old
```

Variables – Methods
List - ArrayList

public/private

```
public String
name;
private int age;
```

- Defines the scope of an instance variable, a method, ...
  - public
    - Means that it can be used everywhere
      - Inside or outside the class, a method, ...
  - private
    - Means that it can only be used within a specific scope
      - Only inside the class, ...

User
-age: int +name: String
+User() +User(name: String, age: int)

Getter

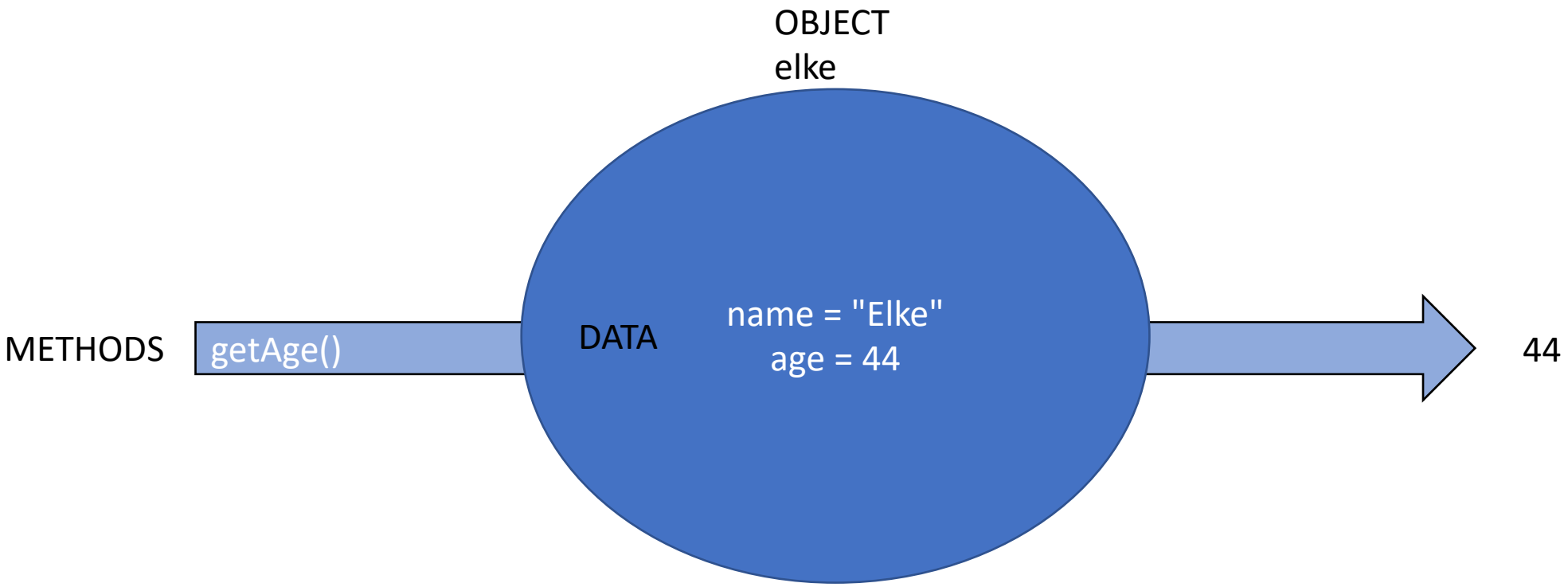
- public method to get the value of a private instance variable
- return keyword returns the wanted result when this getter is called
  - in this example the return value is of type int (public int … in the heading of the method)

```
private int age;

public int getAge() {
  return this.age;
}

public User(String name, int age) {
  this.name = name;
  if (age >= 0)
    this.age = age;
}
```

User
-age: int +name: String
+User() +User(name: String, age: int) +getAge(): int


OBJECT
elke
METHODS
getAge()
DATA
name = "Elke"
age = 44
44

Demo Example Users

Overview Add User

Name	Age	Years of membership
Elke	44	9
Miyo	14	4
Eric	65	[illegible]
Yuki	12	2

2020, 2022

List

```java
import java.util.ArrayList;
import java.util.List;

private List<Integer> membershipYears
= new ArrayList<Integer>();
```

- List<E>
  - provides the facility to maintain the *ordered collection*
  - with E the type of elements in the list
  - it contains the index-based methods to insert, update, delete and search the elements
  - is found in the java.util package
    - a package is like a map on your computer
    - you need to import the class when you want to use it

- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html

ArrayList

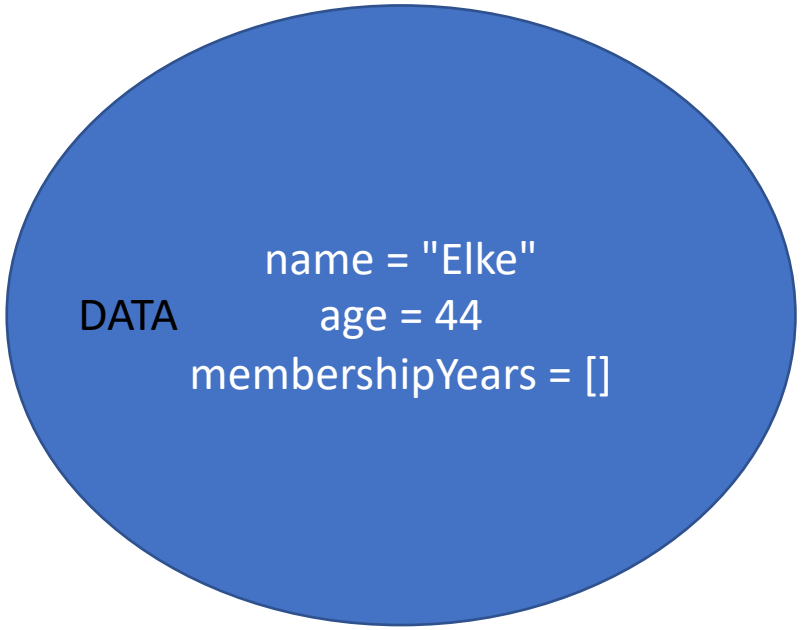
```java
import java.util.ArrayList;
import java.util.List;

private List<Integer> membershipYears
= new ArrayList<Integer>();
```

- ArrayList<E>
  - is a resizable array
  - with E the type of elements in the list
  - which also can be found in the java.util package
  - already has a lot of predefined methods
    - add(Object)
    - size()
    - get(index)
    - …

- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

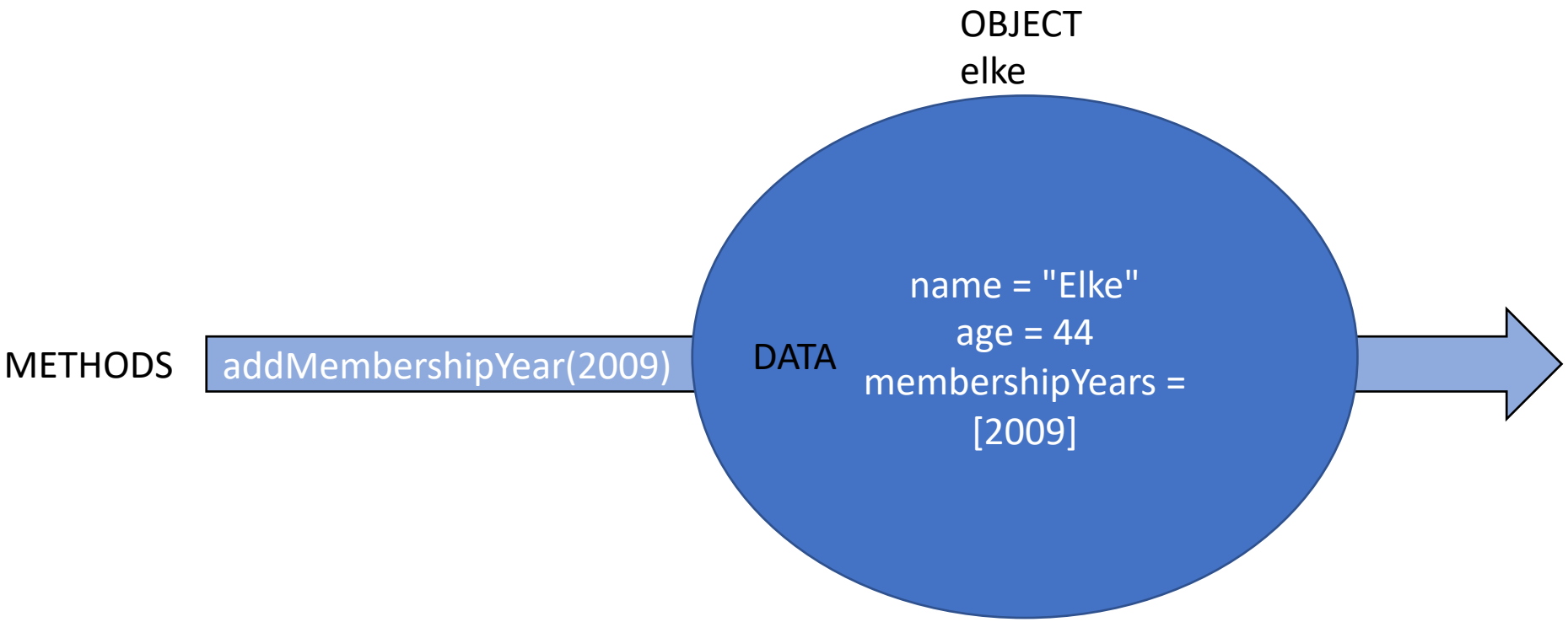
OBJECT
elke



Method with parameter

- addMembershipYear
  - name of method
- (int year)
  - Parameter
    - type: int
    - name: year
- void
  - no return value expected

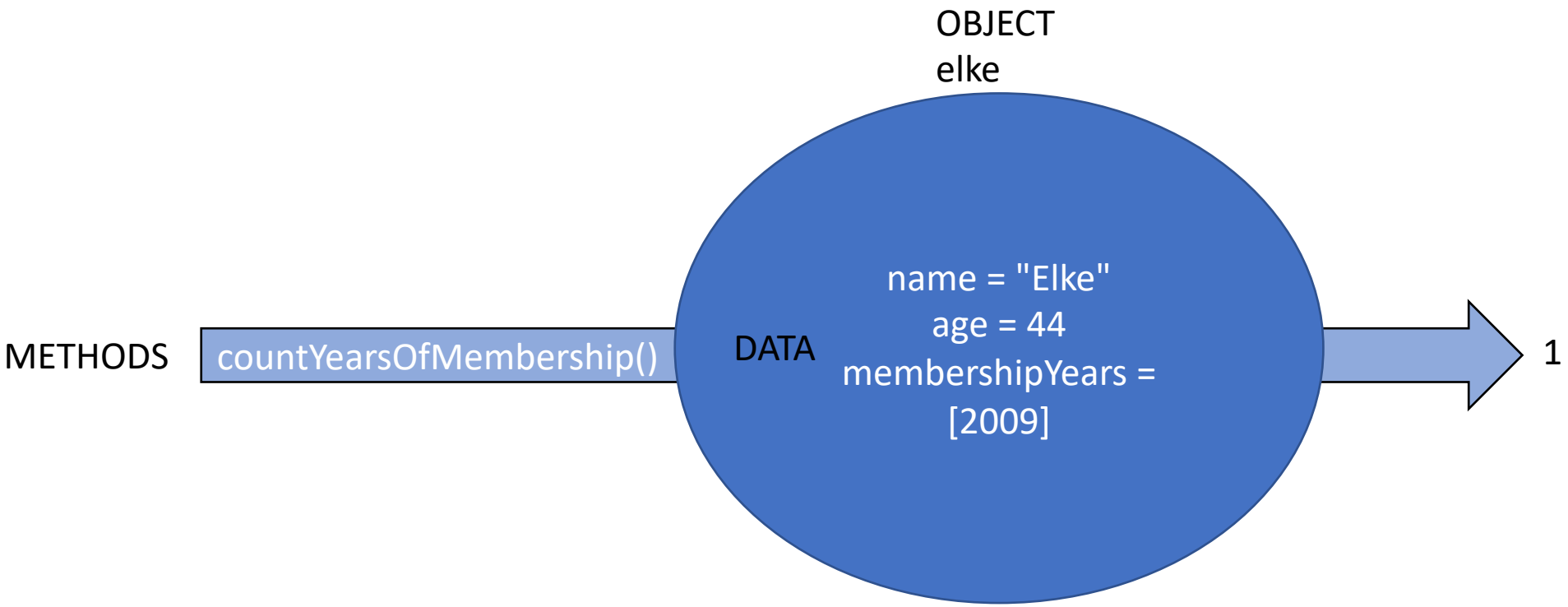
```
public void addMembershipYear (int year) {
  membershipYears.add(year);
}
```


OBJECT
elke
METHODS
addMembershipYear(2009)
DATA
name = "Elke"
age = 44
membershipYears =
[2009]

```java
public int countYearsOfMembership () {
  return membershipYears.size();
}
```

Method with return value

- int
  - type of what will returned as result of the method
- return
  - returns the value behind this keyword


OBJECT
elke
METHODS
countYearsOfMembership()
DATA
name = "Elke"
age = 44
membershipYears =
[2009]
1

Iteration

- for loop
  - use it to iterate over elements of a list

```
public int countMembershipYearsAfter1999 () {
  int result = 0;
  for(Integer year: membershipYears) {
    if (year > 1999)
      result++;
  }
  return result;
}
```

CLASS

```java
package demo;

import java.util.ArrayList;
import java.util.List;

public class User {

  private String name;
  private int age;
  private List<Integer> membershipYears = new ArrayList<Integer>();

  public User(String name, int age) {
    this.name = name;
    if (age >= 0)
      this.age = age;
  }

  public int countMembershipYearsAfter1999 () {
    int result = 0;
    for(Integer year: membershipYears) {
      if (year > 1999)
        result++;
    }
    return result;
  }

  public int countYearsOfMembership () {
    return membershipYears.size();
  }

  public void addMembershipYear (int year) {
    membershipYears.add(year);
  }

  public int getAge() {
    return this.age;
  }

  public String getName () {
    return name;
  }

}
```

initialiseert 'membershipYear' lijst

vraagt name & age van gebruiker

voegt alle jaren na 1999 van lijst 'membershipYears' toe aan result

stuurt terug hoeveel getallen id lijst staan

'void' duid aan daje iets hebt da geen return waarde heeft
en hier zeggen ze dat ze een jaar toevoegen aan de lijst 'membershipYear'

functie voor leeftijd te vragen

CLASS DIAGRAM

User
-age: int -name: String -membershipYears: List<Integer>
+User(name: String, age: int) +getAge(): int +getName(): String +countYearsOfMembership(): int +addMembershipYear(year: int) +countMembershipYearsAfter1999(): int

References

- Java API
  - https://docs.oracle.com/en/java/javase/17/docs/api/index.html

- Java Basic Tutorials
  - https://www.tutorialspoint.com/java/index.htm
  - https://www.w3schools.com/java/default.asp