Secure OOP with Java

Lecture - Unit 03

Claudia Maderthaner <claudia.maderthaner@fh-hagenberg.at>

Classes

- Basic unit of programming in object-oriented programming
- Building blocks of a Java application
- Blueprint for making objects
- There are other approaches like prototypes in JavaScript

Declaring a Class

```
[modifiers] class ClassName {
    [modifiers] DataType fieldName [= inital value];

    [modifiers] ReturnType methodName(ParameterType... param) {
        // method body
    }
}
```

```java
public class Human {

}
```

```java
public class Human {
    String name;

    public Human(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Fields

- Fields describe the **state** of an object
- Represent properties (or attributes) of objects of the class

Field Declaration

```
[modifiers] DataType fieldName [= inital value];
```

```
String name;
```

Field Initialization

```
String name = "Doe";
```

Default Initialization of Fields

- Numeric fields are initialized to zero
    - `byte`: `0`
    - `short`: `0`
    - `int`: `0`
    - `long`: `0L`
    - `float`: `0.0f`
    - `double`: `0.0`
    - `char`: `\u0000`
- Boolean fields are initialized to `false`
- Reference-type fields are initialized to `null`

Methods

- Methods describe the **behaviour** of an object
- Named block of code
- Every method must be declared in a class

Method Declaration

```
[modifiers] ReturnType methodName(ParameterType param) {
    // method body
}
```

```
public void setName(String name) {
    this.name = name;
}

public String getName() {
    return name;
}
```

Return Type

- Exactly one return type
  - primitive value
  - reference-typed value
  - `void`

```java
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
```

```java
public void setName(String name) {
    if (name == null) {
        return;
    }
    this.name = name;
}
```

```java
public String sayHello(String name) {
    if ((name.length() % 2) == 0) {
        return "Hello " + name;
    }
} // won't compile
```

Method Invocation

aka "calling a method"

```
human.setName("Scrooge McDuck");
String name = human.getName();
```

Parameter vs. Argument

Parameters

```
public void setName(String firstname, String lastname) {
    this.name = firstname + " " lastname;
}
```

Arguments

```
human.setName("Scrooge", "McDuck");
```

```
human.setName("Scrooge" + " " + "McDuck");
```

Call by Value

```
public void setAge(int age) {
    this.age = age;
}
```

→ the argument is a copy of the primitive value

```
public void setName(String name) {
    this.name = name;
}
```

→ the argument is a copy of the reference, not of the referenced object

Local Variables

Method Signature

- Uniquely identifies the method within a class
    - Method name
    - Parameter list

```
public void setName(String name) {
    this.name = name;
}
```

Method Overloading

- Same name
- Different type and/or number of parameters

```
public void setName(String name) {
    this.name = name;
}

public void setName(String firstname, String lastname) {
    this.name = firstname + " " lastname;
}
```

```java
public void setName(String name) {
    this.name = name;
}

public void setName(String firstname) { // won't compile
    name = firstname;
}


public String setName(String name) { // won't compile
    this.name = name;
    return name;
}
```

Objects

Object Creation

Constructor

- Special method with
  - the same name as its class
  - no return type
- Creates a new object (instance) of class
- Allocates heap memory space

```java
public class Human {
    String name;

    public Human(String name) {
        this.name = name;
    }
}
```

new Operator

```
Human scrooge = new Human("Scrooge", "McDuck");
```

Default Constructor

```java
public class Human {
    String name;

    public Human() {
        // no op
    }
}
```

```java
public class Human {
    String name;
}
```

The default constructor is only added when there is no other constructor available.

Explicit Default Constructor

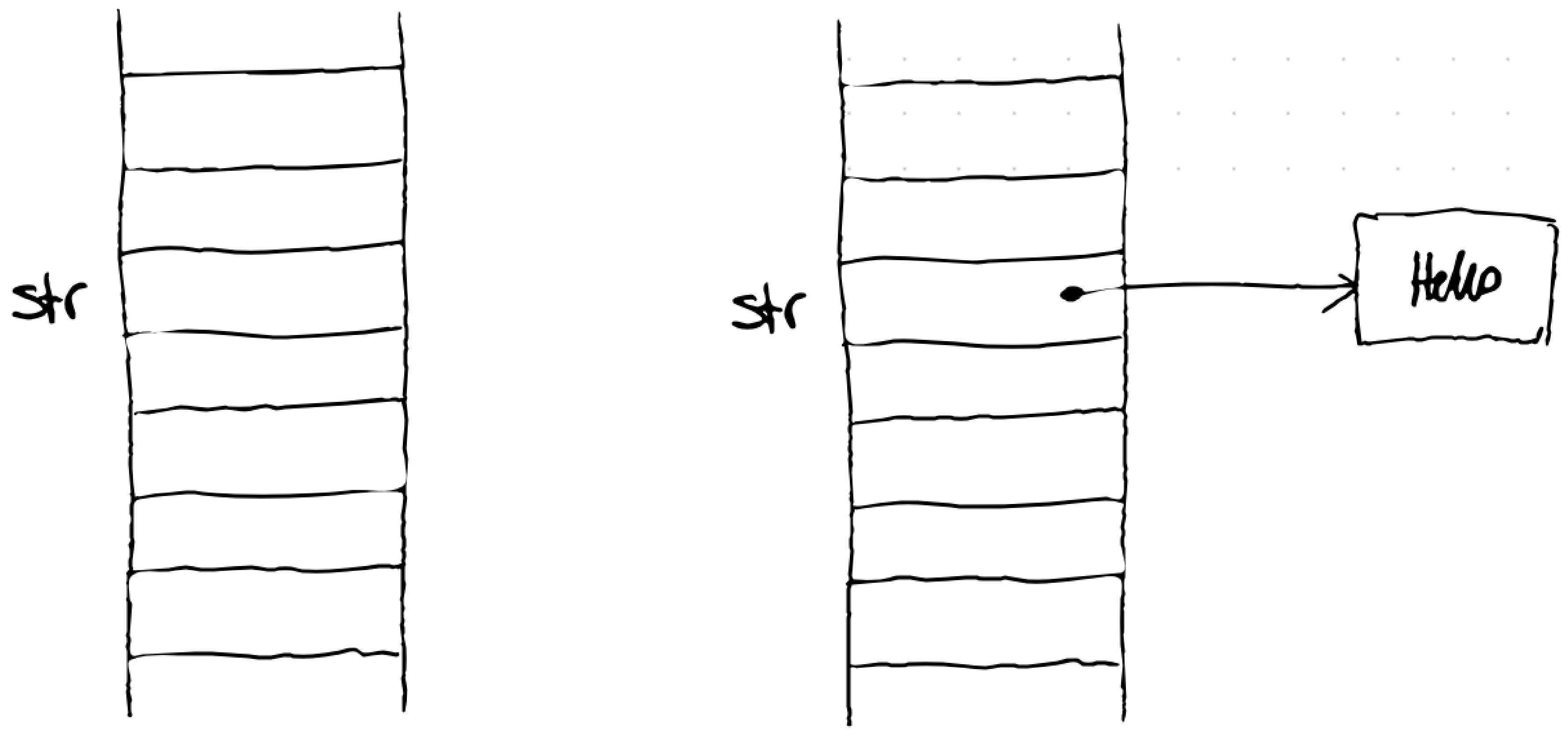
```java
public class Human {
    String name;

    public Human() {
        // no op
    }

    public Human(String name) {
        this.name = name;
    }
}
```

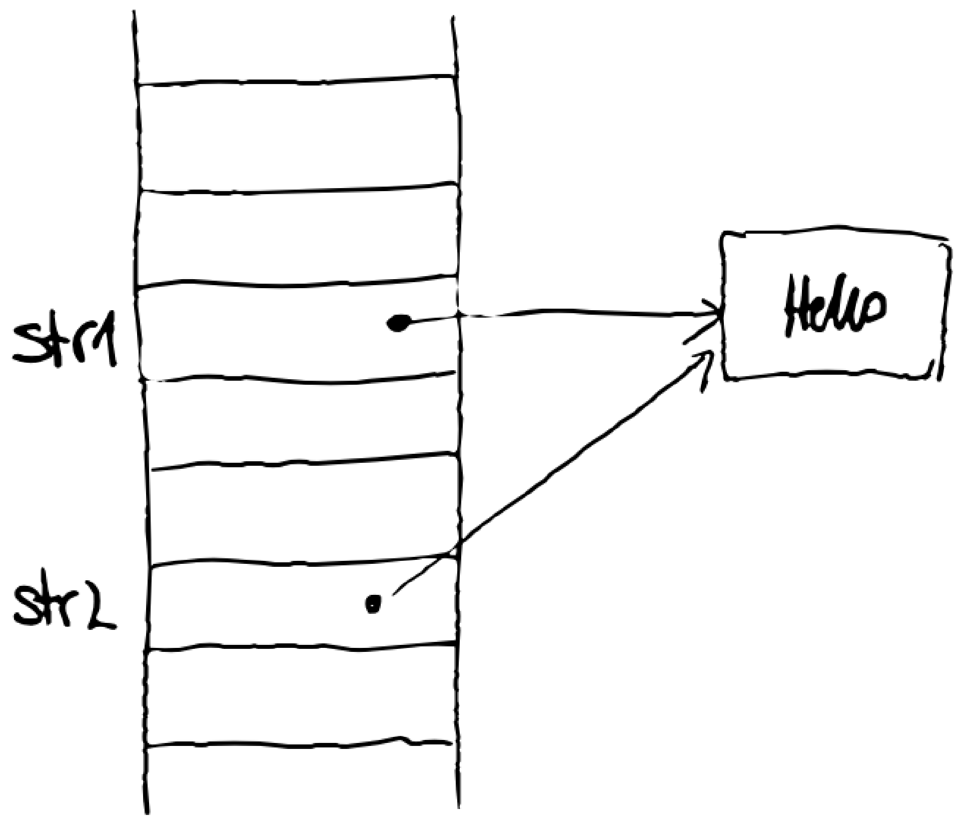
this Constructor

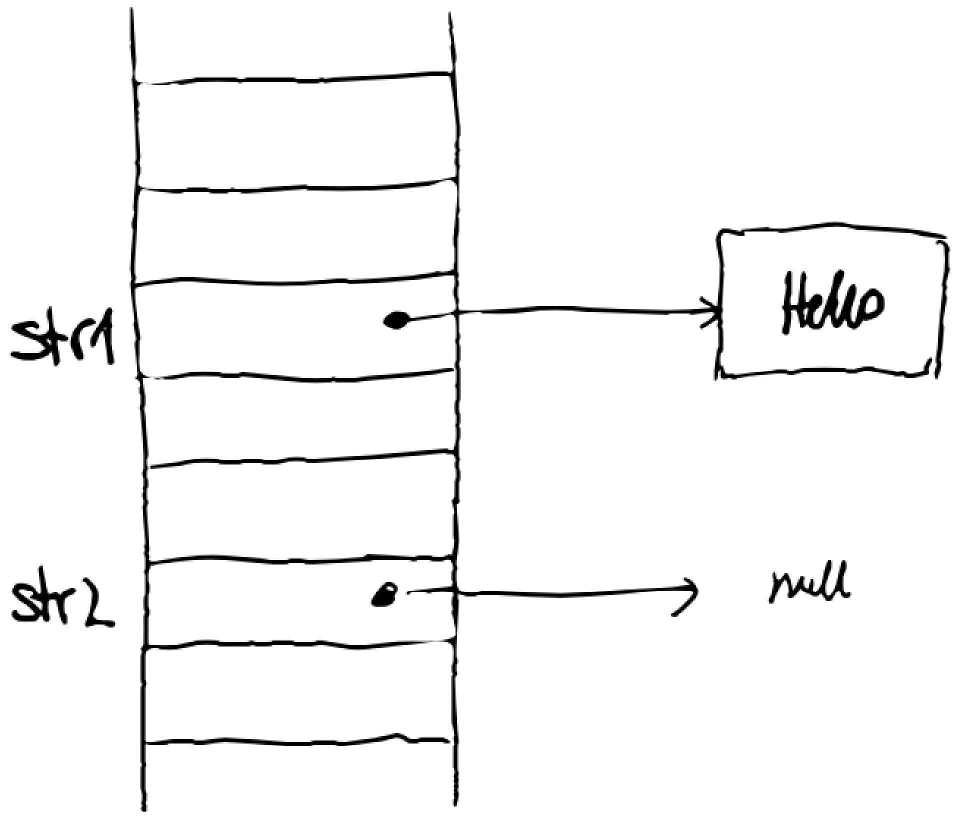
```java
public class Human {
    String name;

    public Human() {
        this("John Doe");
    }

    public Human(String name) {
        this.name = name;
    }
}
```

→ constructor chaining

Object References


str
str
Hello


str1
str2
Hello


str1
str2
Hello
null

this Reference

- Reference to the current object

```
public void setName(String name) {
    this.name = name;
}
```

Object Interaction

```java
public class Human {
    String name;

    public Human(String name) {
        this.name = name;
    }

    public void switchNames(Human otherHuman) {
        String tempName = this.name;
        this.name = otherHuman.name;
        otherHuman.name = tempName;
    }
}
```

```java
Human john = new Human("John");
Human jane = new Human("Jane");

john.switchNames(jane);
```

Object Destruction

```java
Human human = new Human("John", "Doe");

human = new Human("Jane", "Doe");

human = null;
```

Garbage Collector

- Runs in background
- Periodically counts references
- Deletes objects which are unreachable (meaning there are no more references to the object)

Accessing Members

```
variable.memberField

variable.memberMethod()
```

Class-level Members

- Class-level members are shared by all instances of a class
- Declared with the `static` modifier
    - Static variables (aka class variables)
    - Static methods

```java
public class Human {
    static final String HOME_PLANET = "Earth";

    static String latestNameUsed;

    String name;

    public Human(String name) {
        this.name = name;
        latestNameUsed = name;
    }

    public static getLatestNameUsed() {
        return latestNameUsed;
    }
}
```

Accessing Class-level Members

```
String name = Human.latestNameUsed;

name = Human.getLatestNameUsed();
```

Do avoid

```
Human scrooge = new Human("Scrooge McDuck");

String name = scrooge.latestNameUsed;

name = scrooge.getLatestNameUsed();
```

Class-level Members and Objects

- Class-level members cannot access instance fields or methods directly.
- There is no `this` reference in class-level members.

```java
public class Message {
    String message;

    String sayHello() {
        return "Hello!";
    }

    public static void main(String[] args) {
        System.out.println(message); // won't compile
        System.out.println(this.sayHello()) // won't compile
    }
}
```

Contact

Moodle Discussion Board

claudia.maderthaner@fh-hagenberg.at