

List of Contents

1. [REST API General Tips](#)
2. [REST API in Java with Spring](#)
3. [gRPC](#)
4. [Flask API in Python](#)
5. [Python - general syntax and tips](#)
6. [JUnit](#)
7. [EasyMock](#)
8. [Static and dynamic analysis](#)
9. [Git](#)
10. [DOCKER](#)
11. [ALL RESOURCE LINKS:](#)

REST API General Tips

HTTP Status Messages

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. **Informational responses (100 – 199)**
2. **Successful responses (200 – 299)**
3. **Redirection messages (300 – 399)**
4. **Client error responses (400 – 499)**
5. **Server error responses (500 – 599)**

Often used codes in excercises are:

Code	Name	Purpose	
2xx	Successful		
200	OK	The request is OK (this is the standard response for successful HTTP requests)	
201	Created	The request has been fulfilled, and a new resource is created	
204	No Content	The request has been successfully processed, but is not returning any content (may be returned for successful delete)	
4xx	Client Error		
400	Bad Request	The request cannot be fulfilled due to bad syntax or content	
404	Not Found	The requested page/resource could not be found	

If you don't know an HTTP Code you can use `HttpStatus.<Message Name>.value()` class in Java

For example: `HttpStatus.OK.value()` returns `200`

REST API in Java with Spring

Making REST API Requests

With *WebClient* object:

(e.g Spring WebFlux)

GET

```
public void getAllPersons(CustomerSortingOptions sortingOptions, Consumer<List<Customer>> customerConsumer) {
    webClient.get()
        .uri(uriBuilder -> uriBuilder
            .path("customers")
            .queryParams("sortField", sortingOptions.getSortField())
            .queryParams("sortingOrder", sortingOptions.getSortingOrder())
            .build())
        .retrieve()
        .bodyToMono(new ParameterizedTypeReference<List<Customer>>() {
        })
        .onErrorStop()
        .subscribe(newPersons -> {
            customers.clear();
            customers.addAll(newPersons);
            customerConsumer.accept(customers);
        });
}
```

POST

```
public void addPerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Complete the http post request to the server
    webClient.post()
        .uri("customers")
        .bodyValue(customer)
        .retrieve()
        .bodyToMono(Customer.class)
        .onErrorStop()
        .subscribe(newCustomer -> {
            customers.add(newCustomer);
            customerConsumer.accept(customers);
        });
}
```

PUT

```
public void updatePerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Complete the http put request to the server
    webClient.put()
        .uri("customers/" + customer.getId())
        .bodyValue(customer)
        .retrieve()
        .bodyToMono(Customer.class)
        .onErrorStop()
        .subscribe(newCustomer -> {
            customers.replaceAll(x -> x.getId().equals(customer.getId()) ? newCustomer : x);
            customerConsumer.accept(customers);
        });
}
```

DELETE

```
public void deletePerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Make an http delete request to the server
    webClient.delete()
        .uri("customers/" + customer.getId())
        .retrieve()
        .toBodilessEntity()
        .onErrorStop()
        .subscribe(deletedCustomer -> {
            customers.remove(customer);
            customerConsumer.accept(customers);
        });
}
```

Just in case: webClient Constructor

```
public CustomerController() {
    this.webClient = WebClient.builder()
        .baseUrl("http://localhost:8080/")
        .defaultHeader(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
        .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
        .build();
    this.customers = new ArrayList<>();
}
```

With Request Parameters:

Endpoint:

```
@GetMapping("persons")
public ResponseEntity<List<Person>> getAllPersons(PersonSortingOptions sortingOptions) { // Don't use @RequestParam unless object is
    simple(for example String), for more complex objects Spring parses the object by itself
    List<Person> persons = personService.getAllPersons(sortingOptions);
    return ResponseEntity.ok(persons);
}
```

Request:

```
public void getAllPersons(PersonSortingOptions sortingOptions, Consumer<List<Person>> personsConsumer) {
    this.webClient.get()
        .uri(uriBuilder -> uriBuilder // Use uriBuilder to build the path
            .path("persons")
            .queryParam("sortingOrder", sortingOptions.getSortingOrder()) // The fields are passed separately
            .queryParam("sortField", sortingOptions.getSortField()) // In the format: ("Name",value)
            .build()
        )
        .retrieve()
        .bodyToMono(new ParameterizedTypeReference<List<Person>>() {
        })
        .onErrorStop()
        .subscribe(persons -> {
            this.persons = persons;
            personsConsumer.accept(this.persons);
        });
}
```

With *RestTemplate*:

```
GetMapping("/getTimeLine/{id}")
public List<Tweet> getTimeLine(@PathVariable("id") int userID) {
    headers.setContentType(MediaType.APPLICATION_JSON);

    ResponseEntity<List<Tweet>> response = restTemplate.exchange(
        getActivityURL + userID, // Set the uri HttpMethod.GET,
        null, // requestEntity = null as there is no body,
        new ParameterizedTypeReference<List<Tweet>>(){} // response body type );
    return response.getBody();
}
```

REST API Endpoints

Example Java class:

```

@RestController
@RequestMapping("/activity")
public class ActivityController {
    private Map<Integer, List<Tweet>> userActivityMap = new HashMap<>();
    private Map<Integer, List<User>> userFollowedMap = new HashMap<>();

    @GetMapping("/getActivity/{id}")
    public List<Tweet> getActivity(@PathVariable("id") int userID) {
        List<Tweet> userTweets = userActivityMap.get(userID);

        if(userTweets == null) return new ArrayList<>();

        return userTweets;
    }

    @PostMapping("/addActivity")
    public void addActivity(@RequestBody Tweet tweet) {
        userActivityMap.computeIfAbsent(tweet.getUser().getUserID(), x -> new LinkedList<>());
        userActivityMap.get(tweet.getUser().getUserID()).add(tweet);
    }

    @PostMapping("/addFollower")
    public void addFollower(@RequestBody FollowRequest followRequest) {
        User follower = followRequest.getFollower();
        User followed = followRequest.getFollowed();

        userFollowedMap.computeIfAbsent(follower.getUserID(), x -> new LinkedList<>())
            .add(followed);
    }

    @DeleteMapping("/deleteActivity")
    public void deleteActivity(@RequestBody Tweet tweet) {
        List<Tweet> userTweets = userActivityMap.get(tweet.getUser().getUserID());
        if(userTweets == null) return;

        userTweets.remove(tweet);
    }

    @DeleteMapping("/deleteFollower")
    public void deleteFollower(@RequestBody FollowRequest followRequest) {
        User follower = followRequest.getFollower();
        User followed = followRequest.getFollowed();

        List<User> followedUsers = userFollowedMap.get(follower.getUserID());
        if(followedUsers == null) return;

        followedUsers.remove(followed);
    }

    @GetMapping("/getFollowedList/{id}")
    public List<User> getFollowedList(@PathVariable("id") int userID) {
        // TODO: this method is also not complete, so you have to implement it accordingly :)
        List<User> followed = userFollowedMap.get(userID);
        if(followed == null) return new ArrayList<>();
        return followed;
    }
}

```

Exercises:

- L02PE01 TUM Social App
- L02PE02 REST Architectural Style Part 2
- L09PE02 Wizardry Character Creation

gRPC

gRPC reference: <https://grpc.io/docs/>

gRPC Java basics: <https://grpc.io/docs/languages/java/basics/>

Protobuf files

Defining classes:

Important tips:

- Array in protobuf means `repeated` field

```
message Date {
  int32 day = 1; // Number specifies order
  int32 month = 2;
  int32 year = 3;
}

message Location {
  string city = 1;
  string country = 2;
}

message LocationDate {
  Location location = 1;
  Date date = 2;
}
```

Defining services:

```
service WeatherReporter {

  //Returning one object
  rpc GetCityWeatherSingleDay(LocationDate) returns (CityWeatherData) {}

  // Returning a stream of objects
  rpc GetCityWeatherMultipleDays(LocationDatePeriod) returns (stream CityWeatherData) {}
}
```

Example of services and objects with repeated fields

This will be utilized by some of the client side examples underneath

```

service MetadataService {
  rpc getTrackMetadata(EntitiesRequest) returns (Tracks) {}

  rpc getAlbumMetadata(EntitiesRequest) returns (Albums) {}
}

message EntitiesRequest {
  repeated uint32 id = 1;
}

message Track {
  uint64 id = 1;
  string name = 2;
  string album = 3;
  string artist = 4;
  uint32 duration = 5;
}

message Tracks {
  repeated Track track = 1;
}

message Album {
  uint64 id = 1;
  string name = 2;
  string artist = 3;
  repeated uint32 tracks = 4;
}

message Albums {
  repeated Album album = 1;
}

```

Generating classes from *.proto* files

1. Gradle tab in IDE and press "Tasks > other > generateProto")
2. The generated classes from the proto file should be in build/generated/source/proto/main folder

Server side

In practice can be done with streams or with simple iteration

```

public void getCityWeatherSingleDay(LocationDate request, StreamObserver<CityWeatherData> responseObserver) {
    var data = allWeatherData.stream().filter(cityWeatherData ->
cityWeatherData.getLocationDate().equals(request)).findFirst().orElse(null);
    if(data != null) responseObserver.onNext(data);
    responseObserver.onCompleted();
}

```

For returning a *stream* of data just call onNext multiple times

With iteration:

```

public void getCityWeatherMultipleDays(LocationDatePeriod request, StreamObserver<CityWeatherData> responseObserver) {
    for(CityWeatherData data : allWeatherData) {
        if(data.getLocationDate().getLocation().equals(request.getLocation())) {
            if(compareDate(data.getLocationDate().getDate(), request.getStartDate()) >= 0 &&
                compareDate(data.getLocationDate().getDate(), request.getEndDate()) <= 0)
            {
                responseObserver.onNext(data);
            }
        }
    }
    responseObserver.onCompleted();
}

```

With streams:

```

public void getCityWeatherMultipleDays(LocationDatePeriod request, StreamObserver<CityWeatherData> responseObserver) {
    var startDate = request.getStartDate();
    var endDate = request.getEndDate();
    allWeatherData.stream()
        .filter(cityWeatherData -> cityWeatherData.getLocationDate().getLocation().equals(request.getLocation()))
        .filter(cityWeatherData -> compareDate(cityWeatherData.getLocationDate().getDate(), startDate) >= 0)
        .filter(cityWeatherData -> compareDate(cityWeatherData.getLocationDate().getDate(), endDate) <= 0)
        .forEach(responseObserver::onNext);
    responseObserver.onCompleted();
}

```

For returning an object which contains array of other objects (stupid???)

```

public void getAlbumMetadata(EntitiesRequest request, StreamObserver<Albums> responseObserver) {
    Albums.Builder albumsBuilder = Albums.newBuilder();

    for(int i = 0; i < request.getIdCount(); i++) {
        // Find the album by its id
        Album album = database.findAlbumById(request.getId(i));

        if(album != null) {
            albumsBuilder.addAlbum(album);
        }
    }

    // In case that an album isn't found builder builds empty list
    Albums albums = albumsBuilder.build();
    responseObserver.onNext(albums);
    responseObserver.onCompleted();
}

```

If you want to return an error

```

public void getTrafficData(RouteRequest request, StreamObserver<TrafficResponse> responseObserver) {
    var trafficModel = db.getTrafficModel(request);
    var response = trafficModel.query(request);
    if(response.isPresent()) {
        responseObserver.onNext(response.get());
        responseObserver.onCompleted();
    } else {
        // Just pass an exception with the onError method
        responseObserver.onError(new Exception("No data available."));
    }
}

```

Client Side

Setup

The **blocking stub** is a placeholder on which functions are called. It needs to be received from the gRPC object in some way.

```
public WeatherClient(Channel channel) {  
    // Get blocking stub from the gRPC object  
    blockingStub = WeatherReporterGrpc.newBlockingStub(channel);  
}
```

or

```
public SpotifyClient(Channel channel) {  
    // spotifyConnection is a wrapper defined in the exercise  
    // this is just an example that there can be other ways to get the stub  
    this.spotifyConnection = new SpotifyConnection(channel);  
    this.blockingStub = spotifyConnection.getStub();  
}
```

Calling method with *single object* return value

```
public void getCityWeatherSingleDay(int day, int month, int year, String city, String country) {  
    // Need to use builders to create the objects for the request  
    Date date = Date.newBuilder().setDay(day).setMonth(month).setYear(year).build();  
    Location location = Location.newBuilder().setCity(city).setCountry(country).build();  
  
    // Building the locationDate which will be passed as a parameter  
    LocationDate locationDate = LocationDate.newBuilder().setDate(date).setLocation(location).build();  
  
    // Getting the returned object from calling the method on the stub  
    CityWeatherData cityWeatherData = blockingStub.getCityWeatherSingleDay(locationDate);  
  
    System.out.println(cityWeatherData.toString());  
}
```

Calling method which returns *stream*

```
public void getCityWeatherMultipleDays(int startDay, int startMonth, int startYear, int endDay, int endMonth, int endYear, String city, String country) {
    // Building objects for the request
    Date startDate = Date.newBuilder()
        .setDay(startDay)
        .setMonth(startMonth)
        .setYear(startYear)
        .build();

    Date endDate = Date.newBuilder()
        .setDay(endDay)
        .setMonth(endMonth)
        .setYear(endYear)
        .build();

    Location location = Location.newBuilder().setCity(city).setCountry(country).build();

    // Building LocationDatePeriod object to be passed as parameter
    LocationDatePeriod locationDatePeriod = LocationDatePeriod.newBuilder()
        .setStartDate(startDate)
        .setEndDate(endDate)
        .setLocation(location)
        .build();

    // Method call returns Iterator<CityWeatherData> for the stream of CityWeatherData
    Iterator<CityWeatherData> iterator = blockingStub.getCityWeatherMultipleDays(locationDatePeriod);

    // Iterating over the objects
    while (iterator.hasNext()) {
        CityWeatherData cityWeatherData = iterator.next();
        System.out.println(cityWeatherData.toString());
    }
}
```

Calling method which has *parameter with repeated field*

```
public String displayPlaylist(int[] playlist) throws SpotifyClientException {
    StringBuilder stringBuilder = new StringBuilder();

    EntitiesRequest.Builder builder = EntitiesRequest.newBuilder();

    // IMPORTANT for REPEATED fields
    // Add all id's to request
    for(int id : playlist) {
        builder.addId(id);
    }

    EntitiesRequest request = builder.build();

    // Unlike in earlier examples returns OBJECT which contains multiple Tracks
    Tracks tracks;

    tracks = blockingStub.getTrackMetadata(request);
    if(tracks.getTrackCount() != playlist.length) throw new SpotifyClientException("Failed to get track metadata");

    for(int i = 0; i < tracks.getTrackCount(); i++) {
        Track track = tracks.getTrack(i);

        int durationInSeconds = track.getDuration();
        int durationInMinutes = durationInSeconds / 60;
        durationInSeconds = durationInSeconds % 60;

        stringBuilder.append(String.format("%d. %s - %s (%d:%02d)%n", i+1, track.getName(), track.getArtist(), durationInMinutes, durationInSeconds));
    }

    return stringBuilder.toString();
}
```

Excercises:

- L02PB01 Weather Reporter
- L02P01 Spotify gRPC
- L03PE03 Gobblemaps

Flask API in Python

Server side endpoints

GET

```
@api.route("/<key>", methods=["GET"])
def get(key: str) -> Response:
    """
    Retrieve the file-content associated with the given key owned by the user.
    If there is no user-id `uid` given in the request body or the user-id does not match, return error message and status.
    Otherwise, return the file-content and success status.
    """
    request_body: dict = request.json
    ff: FileFolder = current_app.ff # type: ignore

    user_id = request_body.get("uid")

    if user_id is None:
        return Response(NO_UID_IN_REQUEST_BODY, BAD_REQUEST)

    file = ff.get(key)

    if file is None:
        return Response(response=KEY_DOES_NOT_EXIST % key, status=NOT_FOUND)

    if file.uid != user_id:
        return Response(response=NO_ACCESS_TO_ENTRY % key, status=BAD_REQUEST)

    return Response(response=file.content, status=OK)
pass
```

PUT

```
@api.route("/<key>", methods=["PUT"])
def put(key: str) -> Response:
    """
    Upsert a file-content associated with the given key owned by the user.
    If there is no user-id `uid` or file-content `content` given in the request body, return error message and status.
    Otherwise, return the success message and success status
    """
    request_body: dict = request.json
    ff: FileFolder = current_app.ff

    if "uid" not in request_body:
        return Response(NO_UID_IN_REQUEST_BODY, BAD_REQUEST)
    if "content" not in request_body:
        return Response(NO_CONTENT_IN_REQUEST_BODY, BAD_REQUEST)

    uid = request_body["uid"]
    file_content = request_body["content"]

    old_entry = ff.get(key)

    if old_entry is None:
        entry = Entry(uid, file_content)
        success = ff.put(key, entry)
        if success:
            return Response(PUT_SUCCESS, OK)
        else:
            return Response(NO_SPACE_IN_FILE_FOLDER, BAD_REQUEST)

    if uid != old_entry.uid:
        return Response(NO_ACCESS_TO_ENTRY % key, status=BAD_REQUEST)

    entry = Entry(uid, file_content)
    ff.put(key, entry)

    return Response(PUT_SUCCESS, OK)
pass
```

DELETE

```
@api.route("/<key>", methods=["DELETE"])
def remove(key: str) -> Response:
    """
    Remove the file-content associated with the given key owned by the user.
    If there is no user-id `uid` given in the request body or the user-id does not match, return error message and status.
    Otherwise, return the removed file-content and success status
    """
    request_body: dict = request.json
    ff: FileFolder = current_app.ff

    if "uid" not in request_body:
        return Response(NO_UID_IN_REQUEST_BODY, BAD_REQUEST)

    uid = request_body["uid"]

    entry = ff.get(key)

    if entry is None:
        return Response(KEY_DOES_NOT_EXIST % key, NOT_FOUND)
    if entry.uid != uid:
        return Response(NO_ACCESS_TO_ENTRY % key, BAD_REQUEST)

    ff.remove(key)

    return Response(entry.content, OK)
pass
```

GET - all

```
@api.route("/", methods=["GET"])
def list() -> Response:
    """
    List all (key, file-content) tuples owned by the user.
    If there is no user-id `uid` given in the request body, return error message and status.
    Otherwise, return the list of file-content and success status.
    """
    request_body: dict = request.json
    ff: FileFolder = current_app.ff

    if "uid" not in request_body:
        return Response(NO_UID_IN_REQUEST_BODY, BAD_REQUEST)

    uid = request_body["uid"]

    items = ff.items()

    output = []

    for key, entry in items:
        output.append(f"{key}:{entry.content}")
    output_string = ".".join(output)

    return Response(output_string, OK)
pass
```

Making requests on the *client side*

Helper method

```
# make json request to server
def json_request(self, method: str, path: str, body: dict):
    self.conn.request(method, path, json.dumps(body), headers={"Content-Type": "application/json"})
    flask_response = self.conn.getresponse()
    return flask_response.read().decode()
```

PUT

```
def put(self, key, content):
    """
    Upsert (key, content) tuple into the file folder.
    Return the server response
    """
    request_body = {"uid": self.user_id, "content": content}
    return self.json_request("PUT", f"/{key}", request_body)
pass
```

GET

```
def read(self, key):
    """
    Retrieve the file-content associated with the given key owned by the user.
    Return the server response
    """
    request_body = {"uid": self.user_id}
    return self.json_request("GET", f"/{key}", request_body)
pass
```

DELETE

```
def remove(self, key):
    """
    Remove the file-content associated with the given key owned by the user.
    Return the server response
    """

    request_body = {"uid": self.user_id}
    return self.json_request("DELETE", f"/{key}", request_body)
    pass
```

GET - ALL

```
def list(self):
    """
    List all file-content owned by the user.
    Return the server response
    """

    request_body = {"uid": self.user_id}
    return self.json_request("GET", "/", request_body)
    pass
```

Excercises:

- L03PE01 Online FileFolder

Python - general syntax and tips

Dictionaries

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict["brand"])
```

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

You can check if a key exists with the keyword *in*:

```
if "model" in thisdict:
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

You can loop over keys and values in dictionary:

```
for x in thisdict: # Loop over keys
    print(x)
for x in thisdict.keys(): #Another way to loop over keys
    print(x)

for x in thisdict: # Loop over values
    print(thisdict[x])
for x in thisdict.values(): # Another way to loop over values
    print(x)

for x, y in thisdict.items(): # Looping over key-value pairs as tuples
    print(x, y)
```

Dictionary methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

JUnit

assertThrows()

```
@Test
public void testSetOnlineCourseUrlWithInvalidUrl() {
    OnlineCourse course = new OnlineCourse("test");
    // You need to pass the CLASS of the exception and a runnable which calls the function
    // then the test will pass if EXACTLY this exception is thrown
    // If you want to test for any exception pass Exception.class however, that is usually discouraged and not needed
    assertThrows(MalformedURLException.class, () -> course.setUrl("hjdahjsd"));
}
```

assertDoesNotThrow()

```
@Test
public void testSetOnlineCourseUrlWithValidUrl(){
    OnlineCourse course = new OnlineCourse("test");
    // Simply assertDoesNotThrow with a runnable
    assertDoesNotThrow(() -> course.setUrl("https://test"));

    // We need to handle constructor of URL throwing an exception despite knowing that it won't as we are passing it a correct URL
    try {
        assertEquals(new URL("https://test"),course.getUrl());
    } catch (MalformedURLException e) {
    }
}
```

Exercises:

- L06P01 University App Unit Testing Part 2
- L06PB02 Chris Neapolitan Restaurant
- L06PB01 University App Unit Testing
- L06PE01 The Iron Bank of Braavos

EasyMock

`@TestSubject` annotations automatically calls the constructor with the mockObjects. However it is not applicable for constructors with other parameters that are not mocked(for example primitive types). In such cases you can call the constructor of the **test subject** as usual inside the test or in a `@Before` method or wherever you want.

Second example contains how to expect *void* method calls

Example with TestSubject

```
// Needs to extend the tests with EasyMock
// If not extended you would need to manually initialize Mock objects
@ExtendWith(EasyMockExtension.class)
class DiscussionTest {

    // TestSubject annotations automatically calls the constructor with the mockObjects
    // However it is not applicable for constructors with other parameters

    @TestSubject
    private Discussion discussion;

    // Mock objects need to be annotated with @Mock
    @Mock
    private Course courseMock;

    @Mock
    private Comment commentMock;

    // TODO implement the tests

    @Test
    public void testComment() {
        // First we need to record
        // Expect a function call and set a predetermined return value
        // This expects the method to be called exactly once
        expect(commentMock.save()).andReturn(true);

        // After setting all expectations we need to replay the mock - e.g set it so it behaves the way we recorded
        replay(commentMock);

        // Test your objects as usual
        discussion = new Discussion();
        assertEquals(0, discussion.getNumberOfComments());
        assertTrue(discussion.addComment(commentMock));
        assertEquals(1, discussion.getNumberOfComments());

        // At the end you need to verify the mock - this checks whether the expected methods were actually called
        verify(commentMock);
    }

    // The methods below show the same usage
    @Test
    public void testCommentIfSavingFails() {
        expect(commentMock.save()).andReturn(false);
        replay(commentMock);

        discussion = new Discussion();
        assertEquals(0, discussion.getNumberOfComments());
        assertFalse(discussion.addComment(commentMock));
        assertEquals(0, discussion.getNumberOfComments());

        verify(commentMock);
    }

    @Test
    public void testStartCourseDiscussion() {
        Person person = new Lecturer("", "", LocalDate.now());
        String testTopic = "testTopic";

        expect(courseMock.isDiscussionAllowed(person)).andReturn(true);

        replay(courseMock);
        assertTrue(discussion.startCourseDiscussion(courseMock, person, testTopic));
        assertEquals(courseMock, discussion.getCourse());
        assertEquals(testTopic, discussion.getTopic());

        verify(courseMock);
    }
}
```

Example with custom Constructor (also contains expecting *void* call)

```
@Mock
Platform platformMock;

...

@Test
public void addNoteGetsURIFromPlatform() throws URISyntaxException {
    URI uri = URI.create("https://notes.tum.de/examplePath"); // An example uri your mock can return
    String expectedWriteDataPath = uri.getPath(); // The path that the NotesServer should write to

    // Set the mock to expect the call and return the set URI
    expect(platformMock.nextURI()).andReturn(uri);

    // To expect a void method call you can just call the method like usual before **replaying** the mock
    platformMock.writeData(expectedWriteDataPath, "Some text");

    // Replay the mock
    replay(platformMock);

    // Instead of annotating with @TestSubject we instantiate the testServer(System Under Test) inside the Test
    NotesServer testServer = new NotesServer(platformMock, false, false);

    // Test like usual
    URI testUri = testServer.addNote("alice", "Some text");
    assertEquals(expectedWriteDataPath, testUri.getPath());

    // At the end you need to verify the mock - this checks whether the expected methods were actually called
    verify(platformMock);
}
```

If you want multiple method invocations

Method	Behaviour
<code>.times(3)</code>	Expects the method to be called exactly 3 times.
<code>.times(2,5)</code>	Expects the method to be called between 2 and 5 times (inclusive).
<code>.atLeastOnce()</code>	Expects the method to be called 1 or more times.
<code>.anyTimes()</code>	The method can be called 0 or more times (it will never cause <code>verify()</code> to fail).

For *void* methods:

```
commentMock.update();           // Call the void method
expectLastCall().times(3);      // Set the invocation count
```

If you want to check that a method is NOT called just replay without expect()

```
@Test
void testSubmitAbsenceDoesNotAlertInstructor() {
    /**
     * TODO: Mock mailService.sendMail(...) to ensure that the instructor is not alerted
     * Tip: By not specifying the behaviour of the mock you can afterwards verify if a method was NOT called
     * Don't forget to use replay!
     */
    Person person = personBuilder
        .withFirstName("Max")
        .withLastName("Mustermann")
        .withIsInstructor(false)
        .build();
    personService.save(person);

    Person instructor = personBuilder
        .withFirstName("Ivailo")
        .withLastName("Traikov")
        .withIsInstructor(true)
        .build();
    personService.save(instructor);
    List<Person> instructors = personService.getInstructors();

    Absence absence = new Absence();
    absence.setPerson(person);
    absence.setComment("test");
    LocalDate now = LocalDate.now();
    absence.setDate(now);
    absence.setId(Long.valueOf(1));

    replay(mailServiceMock);

    absenceService.submitAbsence(absence, false);

    verify(mailServiceMock);
}
```

Excercises:

- L06PB02 Chris Neapolitan Restaurant
- L06P02 Discussion Forum Mock Testing

Static and dynamic analysis

Static analysis

Gcc

With `-Wall`, `-Wextra` compiler flags

Cmake

With `clang --analyze`

Dynamic Analysis

Undefined Behavior Sanitizer

Deals with undefined behaviour in the program such as **Integer overflow**

To use Undefined Behaviour Sanitizer **compile** with `-fsanitize=undefined`

It is available both on `gcc` and `clang`

```
gcc -fsanitize=undefined <source> -o <output>
```

Address Sanitizer

C Address Safety errors that can be caught by address sanitizer:

- **Buffer Overflow:** This occurs when a program writes data beyond the bounds of allocated memory. It can lead to data corruption, crashes, and even allow attackers to exploit the program.
- **Use-Stack-After-Return:** This involves accessing memory after it has been freed. It can lead to undefined behavior as the freed memory might be allocated again for a different purpose.
- **Double Free:** Calling `free()` twice on the same value can lead to undefined behavior. When a program calls `free()` twice with the same argument, the program's memory management data structures become corrupted and could allow a malicious user to write values in arbitrary memory spaces.

Address Sanitizer is available on `gcc` and `clang`

To use Address Sanitizer compile with `-fsanitize=address`

```
clang -fsanitize=address -w <source> -lpthread -o <output_file>
```

Memory Sanitizer

Memory sanitizer specifically targets uninitialized memory reads

It is only available on `clang` with the flag `-fsanitize=memory`

```
clang -fsanitize=memory <source> -o <output>
```

Thread Sanitizer

Thread Sanitizer deals specifically with data races

It is available on both `gcc` and `clang` with the flag `-fsanitize=thread`

```
gcc -fsanitize=thread <source> -o <output>
```

In general to use multi threading in C you need to use the `-pthread` flag. So if you don't see it you may need to add that as well.

Exercises:

- L09P02 Garching Best Software
- L08PE04 Spaceship
- L07P02 Dynamic Code analysis for Priority Scheduling
- L07PE02 Playlist Manager -- Memory Debugging with MemorySanitizer
- L07PE03 Gym Threading

Git

git checkout

Git checkout has two functions:

1. To change the current branch
2. To restore a version of a specified file

Switching and creating branches

If you want to switch to an existing branch you can use:

```
git checkout <branch-name>
```

If you want to create a new branch and switch to it you can do:

```
git checkout -b <branch-name>
```

This starts the branch at the current commit.

Restoring or taking a specific version of a file

To restore the file version from a specific commit:

```
git checkout <commit-hash> <file-path>
```

To restore the file version from a specific branch:

```
git checkout <branch-name> <file-path>
```

You can also checkout a whole commit.

```
git checkout <commit-hash>
```

This however creates a detached head state in which you cannot make new commits. It isn't the same as replacing all files in the current directory with the states from the commit. Think of it more as changing to the selected commit like you would change to a branch.

If you however want to replace all files in the current directory with the states from the commit

```
git checkout <commit-hash> .
```

git merge

This merges the selected branch into the current branch. You can change the current branch with `git checkout`

Merging is usually done automatically but sometimes that cannot be done because the two branches have changes to the same code which cannot be merged automatically. This is called a merge conflict. When this is the case `git` puts conflict markers in the files which look like this:

```
void calculate_bonus(struct employeeInfo *entry)
{
    //calculation of bonus factor
    float bonus_factor = 1000 / ( entry->salary);

    //calculation of grade factor
    float grade_factor = entry->performanceGrade / 10.0;

    //calculate_bonus
    <<<<<< HEAD // This ist the change from the current branch
    entry->bonus = entry->salary * bonus_factor * grade_factor * 0.5;
    =====
    entry->bonus = entry->salary * bonus_factor * grade_factor * 2;
    >>>>>> feature/new_feature_E // This is the incoming change
}
```

To resolve the merge conflict you need to delete the conflict markers and keep the code you desire. You can even chose to resolve the conflict through writing code that isn't included in both of the changes. After that you need to commit the resolved code.

If you want to **squash** all commits on another branch and apply them as a **single** commit on the current branch you can use:

```
git merge --squash <branch-name>
```

git rebase

Git **rebase** is usually used to take a series of commits and **replay** them on top of another commit.

This will take all commits on the current branch and **replay** them on main. Unlike merge it goes in the direction *current commit => main*

```
git rebase <onto-branch>
```

or if you want to select the branch to **replay**

```
git rebase <branch-name> <onto-branch>
```

Git rebase however can also be used to **squash** or **reorder** commits by running it in interactive mode on the last N commits.

```
git rebase -i HEAD~N
```

This opens this editor:

```
pick f7f3f6d Change my name a bit
pick 310154e Update README formatting and add blame
pick a5f4a0d Add cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Commits are listed from first to last chronologically.

If you want to **squash** all commits you still have to **pick** the **first** one.

```
pick f7f3f6d Change my name a bit
squash 310154e Update README formatting and add blame
squash a5f4a0d Add cat-file
```

After this another editor opens for the message of the commit which remains.

If you want to then copy squashed commits from another branch onto main you can `git merge` which will automatically fast-forward and just paste the squashed commit on main.

However this whole operation can also be done with `git merge --squash <branch-name>`

git cherry-pick

This applies changes **ONLY** from the selected commit(s) to the current branch. By default this creates a new commit for each of them.

```
git cherry-pick <commit-hash> <commit-hash1> ...
```

If you don't want to create any commits automatically and manage it yourself you can do:

```
git cherry-pick --no-commit <commit-hash> <commit-hash1> ...
```

git apply

`git apply <patch-file>` applies the changes in a .patch file (a file which describes these changes) to the repository. This however does not commit them so you will still need to `git commit`

```
git apply patch_file.patch
```

Java BUILD.bazel

```
# You need to first load the java_binary and java_library rules
# How this is done depends on your MODULE.bazel file
load("@rules_java//java:defs.bzl", "java_binary", "java_library")

java_binary(
    name = "client", # Name of the target
    srcs = ["src/eist/Main.java"], # Sources are the .java file(s)
    main_class = "eist.Main", # The main class is specified as such
    deps = [":event-facade", ":model"] # Depends on Event Facade and the model
)

# TODO: Add a java_library target with the name 'event-facade'
java_library(
    name = "event-facade", # Name of the library
    srcs = ["src/eist/EventFacade.java"], # Sources. IMPORTANT - library has no main
    deps = [":event-client", ":token-generator-client", ":model"] # Dependencies
)

# TODO: Add a java_library target with the name 'event-client'
java_library(
    name = "event-client",
    srcs = ["src/eist/EventClient.java"],
    deps = [
        "@maven//:org_springframework_boot_spring_boot_starter_web",
        "@maven//:org_springframework_boot_spring_boot_starter_webflux",
        "@maven//:org_springframework_spring_web",
        ":model"
    ]
)

# TODO: Add a java_library target with the name 'token-generator-client'
java_library(
    name = "token-generator-client",
    srcs = ["src/eist/TokenGeneratorClient.java"],
    deps = [
        "@maven//:org_springframework_boot_spring_boot_starter_web",
        "@maven//:org_springframework_boot_spring_boot_starter_webflux",
        "@maven//:org_springframework_spring_web",
        ":model"
    ]
)

java_library( # Library for the model as it is used in all other classes
    name = "model",
    srcs = ["src/eist/Employee.java", "src/eist/Event.java"]
)
```

``

Python BUILD.bazel

```
load("@rules_python//python:defs.bzl", "py_binary")

# TODO: Add a py_binary target with the name 'event_microservice'
py_binary(
    name = "event_microservice", # name of the binary for inside bazel
    srcs = ["src/event_microservice.py"], # the source files paths relative to the BUILD.bazel
    main = "src/event_microservice.py", # The main for python is just the source file
    deps = ["@pip_deps_server//flask"], # Dependencies - in this case we load from dependencies hub defined in the MODULE.bazel
)
```

MODULE.bazel

```
#####
# Bazel now uses Bzlmod by default to manage external dependencies.
# Please consider migrating your external dependencies from WORKSPACE to MODULE.bazel.
#
# For more details, please check https://github.com/bazelbuild/bazel/issues/18958
#####

bazel_dep(name = "rules_jvm_external", version = "6.7")
bazel_dep(name = "rules_java", version = "8.3.2")
maven = use_extension("@rules_jvm_external//extensions.bzl", "maven")

maven.install(
    artifacts = [
        'org.springframework.boot:spring-boot-starter-web:2.1.6.RELEASE',
        'org.springframework.boot:spring-boot-starter-webflux:2.1.6.RELEASE',
        'org.springframework:spring-web:5.1.5.RELEASE',
    ],
    repositories = [
        'https://repo1.maven.org/maven2',
    ],
    fetch_sources = True,
)

use_repo(maven, "maven")

# Helpful documentation for Python rules can be found here:
# https://rules-python.readthedocs.io/en/latest/index.html

bazel_dep(name = "rules_python", version = "1.4.1")
python = use_extension("@rules_python//python/extensions:python.bzl", "python")

# Setup Python 3.8 environment
python.defaults(
    # The environment variable takes precedence if set.
    python_version = "3.8",
)
python.toolchain(
    python_version = "3.8",
)

pip = use_extension("@rules_python//python/extensions:pip.bzl", "pip")
pip.parse( # Create a rule for the requirements
    python_version = "3.8",
    hub_name = "pip_deps_server",
    requirements_lock = "//server-side:requirements_lock.txt"
)

# Imports the pip toolchain generated by the given module extension into the scope of the current module.
use_repo(pip, "pip_deps_server")
```

DOCKER

Docker reference: <https://docs.docker.com/>

Dockerfiles

<https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/>

<https://docs.docker.com/reference/dockerfile/>

```
FROM python:3.13
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy in the source code
COPY src ./src
EXPOSE 8080

# Setup an app user so the container doesn't run as the root user
RUN useradd app
USER app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Building docker

```
docker build -t <tag>
```

Docker processes

```
docker ps
```

Exercises

- [L09P01 - Registry Office Service Deployment](#)
- [L09P02 - Garching Best Software](#)
- [L08PE04 - Spaceship](#)

ALL RESOURCE LINKS:

- Lecture slides (PDF)
- Your own homework solutions (cloned)
- Cheat-sheets/self-written notes (printed/on paper/PDF)
- C reference: <https://en.cppreference.com/w/c/language>
- Python reference: <https://docs.python.org/3/reference/index.html>
- Java reference: <https://www.baeldung.com/get-started-with-java-series>
- Java reference: <https://docs.oracle.com/en/java/javase/22/>
- gRPC reference: <https://grpc.io/docs/>
- Spotbugs: <https://spotbugs.readthedocs.io>
- AddressSanitizer: <https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html>
- Spring Boot: <https://www.baeldung.com/spring-boot>
- Spring Boot: <https://spring.io/projects/spring-boot>
- Docker: <https://docs.docker.com/>
- JavaFX: <https://openjfx.io/openjfx-docs>
- Gradle: <https://docs.gradle.org/current/userguide/userguide.html>
- Git: <https://git-scm.com/docs>
- JUnit: <https://junit.org/junit5/docs/current/user-guide/>
- Mockito: <https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html>
- EasyMock: <https://easymock.org/user-guide.html>
- man pages (command line, not online)