PERSONAL - EIST Cheat Sheet

Dockerfiles and Paths in Linux

```
FROM baseimage                    # always first command in a dockerfile
WORKDIR </path>                   # directory where the commands in this file
are executed
COPY <src> <dest>                 # copies a file, <src> may contain wildcards
RUN <command>                     # runs a command in a shell, usually linux
syntax
CMD <command> <param1> <param2> # default for executing container, there may
only be one CMD instruction
```

- **absolute vs. relative paths**:
  - **absolute** paths **start** with a `/`: `/home/user/documents/eist/practice/h01/build/libs/*.jar`
  - **relative** paths **never start** with a `/`: `build/libs/*.jar`
  - `.` simply means "here" (*e.g.* `COPY app.jar .` *copies* `app.jar` *to the root of the directory without renaming it*)

Async Messages (RestTemplate, CompleteableFuture)

```
/*
    STEP 1:
    Send POST-request by creating a corresponding ResponseEntity using
postForEntity() of RestTemplate
*/
public void postMethod(Object parameter){
    ResponseEntity<Class> response = rest.postForEntity(URL_STRING,
HTTP_ENTITY, CLASS);
}
```

```
/*
    STEP 2:
    Handle POST-requests accordingly using method with same name in a
@RestController-Class
*/

@RestController
@RequestMapping(value = "/whatever/", consumes = "application/json")
public class InboxResource {
    @PostMapping("whatever")
    public ResponseEntity<String> postMethod(@RequestBody Object parameter){
        return new ResponseEntity<>(parameter, HttpStatus.OK);
    }
}
```

```
/*
    STEP 3:
    Replace synchronous methods with async counterparts using
CompleteableFuture.supplyAsync
*/
// sync version, example given with produce()-method
@PostMapping("whatever")
public ResponseEntity<Object> postMethodOld(@RequestBody Object parameter) {
    return new ResponseEntity<>(factory.produce(parameter), HttpStatus.OK);
}

// async version, example given with produce()-method and string return type
@PostMapping("whatever")
public CompletableFuture<String> postMethodAsync(@RequestBody Object
parameter) {
    return CompletableFuture.supplyAsync(() ->
```

```
factory.produce(parameter).getName());
}
```

```
/*
    STEP 3.5:
    Reroute methods in client class.
*/

// sync version
public Object produceSync() {
    var request = createHttpEntity(STRINGNAME);
    return rest.postForObject(BASE_URL + PARAM, request, Object.class);
}

// async version
public void produceAsync(InboxClient client) {
    CompletableFuture<String> message =
            CompletableFuture.supplyAsync(
                    () -> rest.postForObject(BASE_URL + PARAM,
createHttpEntity(STRINGNAME), String.class)
            );
    try {
        client.ready(message.get());
    } catch (InterruptedException | ExecutionException e) {
        throw new RuntimeException(e);
    }
}
```

REST API w/ Spring Boot

Endpoints (Server-Side)

```
// GET, POST, PUT and DELETE functions (CRUD)
@GetMapping("endpoint_dir")
public ResponseEntity<List<Object>> getObjects(@RequestParam(name = "name",
defaultValue = value)...) {
    return ResponseEntity.ok(objectService.getAllObjects());
}

@PostMapping("endpoint_dir")
public ResponseEntity<Object> createObject(@RequestBody Object object) {
    if (condition) { // most likely object.getId() != null
        return ResponseEntity.badRequest().build();
    }
```

```
    return ResponseEntity.ok(objectService.saveObject(object));
}

@PutMapping("endpoint_dir/{object_id}")
public ResponseEntity<Object> updateObject(
    @RequestBody Object updatedObject,
    @PathVariable("object_id") UUID objectId) {
    if (condition) { // most likely !updatedObject.getId().equals(objectId)
        return ResponseEntity.badRequest().build();
    }
    return ResponseEntity.ok(objectService.saveObject(updatedObject));
}

@DeleteMapping("endpoint_dir/{object_id}")
public ResponseEntity<Void> deleteObject(@PathVariable("object_id") UUID
objectId) {
    objectService.deleteObject(objectId);
    return ResponseEntity.noContent().build();
}
```

RESTFUL Requests (Client-Side, Controller Class)

```
private final WebClient webClient; // web client to handle REST requests
private final List<Object> objects; // private, local list of objects to be
stored

public ObjectController(){
    this.webClient = WebClient.builder()
            .baseUrl("http://localhost:8080/") // insert URL here
            .defaultHeader(HttpHeaders.ACCEPT,
MediaType.APPLICATION_JSON_VALUE)
            .defaultHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
            .build();
    this.objects = new ArrayList<>();
}

// POST
public void addObject(Object object, Consumer<List<Object>> objectConsumer)
{
    webClient.post()
            .uri("endpoint_dir")
            .bodyValue(object)
            .retrieve()
```

```
            .bodyToMono(Object.class)
            .onErrorStop()
            .subscribe(newObject -> {
                objects.add(newObject);
                objectConsumer.accept(objects);
            });
}

// PUT
public void updateObject(Object object, Consumer<List<Object>>
objectConsumer) {
    webClient.put()
            .uri("endpoint_dir/" + object.getId())
            .bodyValue(object)
            .retrieve()
            .bodyToMono(Object.class)
            .onErrorStop()
            .subscribe(newObject -> {
                objects.replaceAll(oldObject ->
oldObject.getId().equals(newObject.getId()) ? newObject : oldObject);
                objectConsumer.accept(objects);
            });
}

// DELETE
public void deleteObject(Object object, Consumer<List<Object>>
objectConsumer) {
    webClient.delete()
            .uri("endpoint_dir/" + object.getId())
            .retrieve()
            .toBodilessEntity()
            .onErrorStop()
            .subscribe(x -> {
                objects.remove(object);
                objectConsumer.accept(objects);
            });
}

// GET
public void getAllObjects(Consumer<List<Object>> objectConsumer,
QueryParam...) {
    webClient.get()
            .uri(uriBuilder -> uriBuilder
```

```
                    .path("endpoint_dir")
                    .queryParam("paramName", paramValue) // can be repeated
however many times needed
                    .build())
            .retrieve()
            .bodyToMono(new ParameterizedTypeReference<List<Object>>() {})
            .onErrorStop()
            .subscribe(newObjects -> {
                objects.clear();
                objects.addAll(newObjects);
                objectsConsumer.accept(objects);
            });
}
```

Model-View-Controller

- **model**: application's dynamic data structure; manages data, logic and rules
- **view**: representation of information; what the user sees, e.g. window
- **controller**: accepts input and converts it to commands for the model or view

Multithreading

```
// via Runnable
public class MyRunnable implements Runnable{
    public int integer = 0;
    public void run(){
        System.out.println("Integer: " + integer++);
    }
}

public class Main{
    public static void main(String[] args){
        // with custom, explicit runnable
        Runnable r = new MyRunnable();
        Thread t = new Thread(r);

        t.start();
        t.join(); // surround with try...catch...

        // def run() method of runnable in lambda
        Thread t1 = new Thread(() -> {
            System.out.println("Lambda!");
        });
```

```
        t1.start();
    }
}
```

TEMPLATE: Using Locks

```
lockObject.lock();
try {
   // statements
}
finally {
   lockObject.unlock();
}
```

Semaphores

```
// buffer object
class Buffer {
    Semaphore free;
    Semaphore occupied;
}

// producer thread
produce(){
    free.acquire();
    synchronized(Buffer){
        ...
    }
    occupied.release();
}

// consumer thread
consume(){
    occupied.acquire();
    synchronized(Buffer){
        ...
    }
    free.release();
}
```

JUnit 5 and Mock Testing with EasyMock

JUnit Annotations and Assertions

- **annotations**:
  - `@Test public void exampleTest()`: identifies that `exampleTest()` is a test method
  - `@Timeout(value = x, unit = TimeUnit.y)`: test automatically fails after x units of time
  - `@BeforeEach void setup()`: method gets executed before **every** test (*i.e. multiple times for each test*)
  - `@AfterEach void teardown()`: method gets executed after **every** test
  - `@BeforeAll static void setup()`: code only gets executed **once** at instantiation of class (*before all tests*)
  - `@AfterAll static void teardown()`: code only gets executed **once** after removal of class (*after all tests*)
  - `@Disable`: disables test
- **assertions**:
  - `assertTrue(condition, message)`: checks if `condition` is **true**
  - `assertFalse(condition, message`: checks if `condition` is **false**
  - `assertEquals(expected, actual, message)`: checks if values / objects `expected` and `actual` are **equal**
  - `assertEquals(expected, actual, delta, message)`: for floats and doubles, with error margin delta
  - `fail(message)`: fails the test
  - `assertNull(object, message)`: checks if **object is null**
  - `assertNotNull(object, message)`: checks if **object is not null**
  - `assertSame(expected, actual, message)`: check is `expected` and `actual` refer **to the same object**
  - `assertNotSame(expected, actual, message)`: check is `expected` and `actual` **do not refer to the same object**
  - `assertThrows(expectedType, executable, message)`: check that execution of the supplied executable (*e.g. lambda expression*) throws an exception of the `expectedType` and returns the exception

TEMPLATE: EasyMock

```
@Mock // mock object
Object mockObject;

@TestSubject // SUT
Object testSubject = new Object();
```

```
@Test
public void test() {
    expect(mockObject.function()).andReturn(returnValue);
    replay(mockObject);
    // insert test as usual here
    verify(mockObject);
}
```

SpotBugs, Infer and AddressSanitizer

- **SpotBugs with IntelliJ**: Right Click Project File → SpotBugs → Analyze Project Files Not Including Test Sources
- **Infer**: `infer run -- javac src/de/tum/in/ase/eist/*.java`
- **AddressSanitizer**: if not already included in `Makefile`, add `-fsanitize=address` to `gcc` instruction
  - using heap memory after `free()` (*heap-use-after-free*)
  - using heap memory out of bounds of specified space (*heap-buffer-overflow, e.g.*)
  - using stack memory out of bounds of specified space (*stack-buffer-overflow*)
  - using global memory out of bounds of specified space (*global-buffer-overflow*)

Continuous Integration with Gradle

```
// add plugins
plugins {
    id 'sth.etc.pluginname.morethings' version 'version_number'
}

// add dependencies
// easy way: intellij bottom bar -> dependencies -> search and add -> reload
// manual labor:
dependencies {
    implementation 'ins.ert.coolname.here'
    testimplementation 'sth.to.do.withjunitprobably'
}

// directories
sourceSets {
    main {
        java {
            srcDir 'src'
        }
```

```
    }
    test {
        java {
            srcDir 'test'
        }
    }
}

// see project repositories for full gradle build files if necessary
```

Refactoring

- **Step 1**: create new abstract class, then extend class to subclasses with duplicate code that needs restructuring
- **Step 2**: move all duplicate code from subclasses to new abstract superclass
- **Step 3**: for code fragments that slightly differ from one class to another, create abstract methods in abstract class which will be implemented separately in each subclass

```
// example: L10H01
// subclasses: PhysicalTutorGroupMeeting and VirtualTutorGroupMeeting
public abstract class TutorGroupMeeting {
    // common attributes go here...

    public void practice(){                              // MOVED FUNCTION OF
SUBCLASSES THAT GETS CALLED
        Student tutor = getTutorGroup().getTutor();
        greet(tutor);                                    // SUBCLASS-DEFINED
        session(tutor);                                  // IMPLEMENTED HERE
        end(tutor);                                      // SUBCLASS-DEFINED
    }

    // common getters and setters go here...

    public abstract void greet(Student tutor);           // TO BE IMPLEMENTED
IN SUBCLASS
    public void session(Student tutor){                  // IMPLEMENTED HERE
        // identical code in both subclasses goes here...
    }
    public abstract void end(Student tutor);             // TO BE IMPLEMENTED
IN SUBCLASS
}
```

Merge Conflicts without Rebase

- `git pull --rebase=false` to use **merging**
- `git pull --rebase=true` to use **rebasing** (*not recommended for lecture scope*)