CPS Code Review Check List

This page is not intended to include a comprehensive list of everything that should be checked during a code review for CPS. Instead it attempt to list to less well known or very often (forgotten) rules that we should apply in CPS to keep the high quality of our production and test code.

Security Related Checks

	Description	Notes
1	Do **not log** any **user data** at any log level	since we do not know what is in the user data there could also be sensitive information inside it. Be awara of logging objects, make sure the toString() implementation doesn't include user data for that object. So instead maybe just log fields that are well defined and do not contain user data.

Simple is Good, Complex is Bad

There is one simple rules that applies to all code and can often be used to decide between several coding solutions.

Description	Bad	Good
blocked URL	`Optional<String> optionalResponseBody = Optional.ofNullable (responseEntity.getBody()) .filter(Predicate.not(String::isBlank)); return (optionalResponseBody.isPresent()) ? convert (optionalResponseBody.get()) : Collections.emptyList();`	`String responseBody = responseEntity. getBody(); if (responseBody == null || responseBody. isBlank()) { return Collections.emptyList(); } return convert(responseBody);`

Common Mishaps

	Description	Bad	Good
1	Don't forget to check the **copyright** 🙂 (add/modify new calendar year to existing copyright if needed)	`* Copyright (C) 2021 Other Company`	`* Copyright (C) 2021 Other Company * Modifications Copyright (C) 2021-2022 Nordix Foundation`
2	Don't forget to check the **commit message structure**		It is following ONAP commit message guidelines: Commit Messages#CommitStructure
3	**Overuse of constant** for string literals (that don't add value) Also duplication is not a good reason, it often 'smells like' a method that should be extracted out instead	`final static String HELLO = "Hello "; String message = HELLO + name;`	`String message = "Hello " + name; String sayHello(final String name) { return "Hello " + name; }`
4	Avoid using **! when else** block is implemented	`if (x!=null) { do something; } else { report error; }`	`if (x==null) { report error; } else { do something; }`
5	No need for **else after return** statement	`if (x==true) { return something; } else { return something-else; }`	`if (x==true) { return something; } return something-else;`
6	No need to check for not **empty before iterating** on collection	`if (!myCollection.isEmpty()) { collection.forEach(some action); }`	`collection.forEach(some action);`
7	No **unnecessary test data** Try to minimize the test data to **just** the data needed for the use case under test (often test data from other test is copied and pasted into new test that simply don't need it)	`def 'Registration with invalid cm handle name'() { given: 'invalid cm handle name' cmHandle.id = 'invalid,name' cmHandle.dmiProperties = [dmiProp1: 'dmiValue1'] when: ...`	`def 'Registration with invalid cm handle name'() { given: 'invalid cm handle name' cmHandle.id = 'invalid,name' when: ...`
8	Do **not use 'var'**. Explicitly define object types instead of just using var	`var dataspaceEntity = dataspaceRepository. getByName(dataspaceName);`	`DataspaceEntity dataspaceEntity = dataspaceRepository.getByName (dataspaceName);`

9	**Maven** dependencies should be defined in **Version Management**
	
	***Note.** CPS(core) project has a separate `cps-dependencies` module!
		`<dependencies>`
		`  <dependency>`
		`  <groupId>net.logstash.logback</groupId>`
		`  <artifactId>logstash-logback-encoder</artifactId>`
		`  <version>7.0.1</version>`
		`  </dependency>`
		`</dependencies>`
			`<dependencyManagement>`
			`  <dependencies>`
			`  <dependency>`
			`  <groupId>net.logstash.logback</groupId>`
			`  <artifactId>logstash-logback-encoder</artifactId>`
			`  <version>7.0.1</version>`
			`  </dependency>`
			`  </dependencies>`
			`</dependencyManagement>`
			
			`<dependencies>`
			`  <dependency>`
			`  <groupId>net.logstash.logback</groupId>`
			`  <artifactId>logstash-logback-encoder</artifactId>`
			`  </dependency>`
			`</dependencies>`

10	Use **ObjectMapper**() instead of Gson() See Spike: Which ObjectMapper	`Gson gson = new Gson();` `JsonElement jsonElement = gson.fromJson(jsonString, JsonElement.class);`	`ObjectMapper mapper = new ObjectMapper();` `JsonNode jsonNode = mapper.readTree(jsonString);`
11	Use **String concatenation** instead of **String.format** (5x slower!) where possible	`String.format("cm-handle:%s",cmHandleId);`	`"cm-handle:" + cmHandleId;`
12	Initialize collections/arrays with **known size** where possible	`void processNames(Collection<String> orginalNames)  {` `  String[] processedNames = new String[0];`	`void processNames(Collection<String> orginalNames)  {` `  String[] processedNames = new String[orginalNames.size()];`

13	Contrary to #12 use **0-size**-arrays when converting collection using `Collection.toArray()`
	See this article for explanation
		`void processNames(Collection<String> names)  {`
		`  String[] namesAsArray =`
		`  collection.toArray(new String[names.size()]);`
			`void processNames(Collection<String> names)  {`
			`  String[] namesAsArray =`
			`  collection.toArray(new String[0]);`

14	Use 'entry' when iterating over a Map.**entrySet()**. And name key and value immediately	`trustLevelPerDmiPlugin.entrySet().forEach(trustLevel -> {`
		`  doSomething(trustLevel.getKey(), trustLevel.getValue());`
		`}`
			`trustLevelPerDmiPlugin.entrySet().forEach(entry -> {`
			`  String dmi = entry.getKey();`
			`  TrustLevel trustLevel = entry.getValue();`
			`  doSomething(dmi, trustLevel);`
			`}`

15	Use **@Service** (org.springframework.stereotype) when a class includes operations.
	Use **@Component** when it is a data object only.
		`@Component`
		`public class TrustLevelManager {`
		`  void handleInitialRegistration(...)`
		`  { ... };`
		`}`
			`@Service`
			`public class TrustLevelManager {`
			`  void handleInitialRegistration(...)`
			`  { ... };`
			`}`

Groovy & Spock Conventions

See Groovy & Spock Test Code Conventions