|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| JAVA |
|  |

|  |
| --- |
| SIMPLE |
|  |

|  |
| --- |
| a) how hashmap/hash set works |
| * Hash to bucket, then equals, set base on map |
|  |

|  |
| --- |
| b) array list vs linked list |
|  |

|  |
| --- |
| -linked - queue, linked better insert (because array new table), access first/last, array – access by index |
|  |

|  |
| --- |
| c)  kind of tree |
| - linked – key order  - tree – sort  - hash – quick access  d) |

|  |
| --- |
| -- what needed to put to tree map, |
| e) |
|  |

|  |
| --- |
| -- hashmap vs hashtable |
| (map one key null, many values, table now ) |
| f) exceptions tree  throwable, error/exception, checked runtime |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ADVANCED |
|  |

|  |
| --- |
| a) what is reflection |
|  |

|  |
| --- |
| b) kinds of synchronization, what bettter, what give us lock additionally |
|  |

|  |
| --- |
| c) kinds of ExecutorService |
|  |

|  |
| --- |
| c) ConcurrentHashMap vs Collections.synchronizedMap |
|  |

|  |
| --- |
| d) lock na static, na obiekt, czy sie blokuja wzejemnie |
| ----------------------------------------------------------------------------------------------------------------  SPRING |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| SIMPLE |
|  |

|  |
| --- |
| a) what it is and why we use it, some modules |
|  |

|  |
| --- |
| b) kinds of configuration |
|  |

|  |
| --- |
| -- xml, java, component scan, how it works |
|  |

|  |
| --- |
| c) kinds of beans, default |
| - singleton, session, request, prototype |
|  |

|  |
| --- |
| d) beans annotations above class, what inside |
| Controller, service, repository, component, all are component |
|  |

|  |
| --- |
| e) initialization, destruction methods |
| @PostConstruct, @PreDestroy |

|  |
| --- |
| ADVANCED |
|  |

|  |
| --- |
| a) what are aspects, how aspects works  - code before, after, around method execution, usually for security, logging, transaction  , based on proxy |
|  |

|  |
| --- |
| b) how request bean is injected to singleton |
| - Firstly proxy invented |
| c) how autowired inject, what if 2 with the same type |
| - by type, use @Qualifier |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
| ----------------------------------------------------------------------------------------------------------------- |
| |  | | --- | | DESIGN PATTERNS | |  |  |  | | --- | | a) SOLID | | Single responsibility, open-close, Liskov | | substitution – subclass is superclass, implement all, interface segregation (small interface), dependency inversion – base on abstraction, details in implementation |  |  | | --- | | b) IMMUTABLE, how to make it | | - No setters, all in constructor  - final instance variables  - collections, dates – returned in getter making new object | |  |  |  | | --- | | c) template vs strategy | | -- template by subclass details, compile time  -- strategy, a runetime we decide by setters/ constructor args | |  |  |  | | --- | | d) how to make singleton ? | | - private constructor, get instance method, final instance or double check | | tutaj to nie zadziala z serializacja  albo pole w statycznej klasie zagniezdzonej, albo enum z jedna wartoscia | |  |  |  | | --- | | e) some web design patterns | | - Controller, front controller, filter, service, DAO, DTO | |  |  |  | | --- | | f) some enterprise design patterns  - channel, filter, transformer, service activator, router, adapter, gateway |   ------------------------------------------------------------------------------------------------------------- |
|  |

|  |
| --- |
| XML |
| What XML libs you know |
|  |

|  |
| --- |
| - xpath, xquery, dom, sax (differences), jax b |
|  |

|  |
| --- |
| ----------------------------------------------------------------------------------------------------------------- |

|  |
| --- |
| TEST  a)  what give us mock |
|  |

|  |
| --- |
| - method separation  - result which we need for test  - called with specific parameters  - verify that called |
|  |

|  |
| --- |
| ----------------------------------------------------------------------------------------------------------- |
|  |

|  |
| --- |
| TOOLS. |
|  |

|  |
| --- |
| 1. Common maven sections 2. Git flow |
|  |

|  |
| --- |
| - I have in master commit with hash 000, 111, 222 , want remove 111, how to solve it |
|  |

|  |
| --- |
| -- new branch from master |
|  |

|  |
| --- |
| -- back to 000 on master |
|  |

|  |
| --- |
| -- cherry pick |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
| -------------------------------------------------------------------------------------------------------  SQL |

|  |
| --- |
|  |
| a) |

|  |
| --- |
| - 2 tables author and book |
|  |

|  |
| --- |
| - all books (title) of author ABC |
|  |

|  |
| --- |
| - kartesian, join |
|  |

|  |
| --- |
| b) |
|  |

|  |
| --- |
| incomes |
|  |

|  |
| --- |
| usr\_id | income |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| - all with income greater than 10000; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| SELECT usr\_id, sum(income) |
|  |

|  |
| --- |
| FROM incomes |
|  |

|  |
| --- |
| GROUP BY usr\_id |
|  |

|  |
| --- |
| HAVING sum(income) > 10000 |
|  |

|  |
| --- |
| -------------------------------------------------------------------------------------------------------------- |

|  |
| --- |
|  |
|  |

|  |
| --- |
| LAMBDA |
|  |

|  |
| --- |
| a)  list of strings, only start from "a", as uppercase, as one string with , as selector delimiter |
|  |

|  |
| --- |
| -- list.stream().filter(x -> x.startWIth("a"))map(x -> x.toUpperCase()).collect(Collectors.joining(", ")); |
|  |

|  |
| --- |
| b)  findAny, findFirst, wat return, when difference, when we use one, when other |
|  |

|  |
| --- |
| -- Optional, difference with parallel, if first needed - use first, but slower than any |
|  |

|  |
| --- |
| c)  advanced : |
|  |

|  |
| --- |
| Map<String, List<String>> - surname, name |
|  |

|  |
| --- |
| - only with surname == "abc", |
|  |

|  |
| --- |
| result Map<Integer, List<String>> - length, list with length |
|  |

|  |
| --- |
| Map<Integer, String> map = map.entrySet().stream().filter(entry -> entry.key().equals("abc")) |
|  |

|  |
| --- |
| .flatMap(entry -> entry.value().stream()).collect(Collectors.groupingBy(String::length)); |
|  |

|  |
| --- |
| -- if set - collect(Collectors.groupingBy(String::length,Collectors.toSet())); |
|  |

|  |
| --- |
| -- set, tree map - ohMy.collect(Collectors.groupingBy(String::length, TreeMap::new, Collectors.toSet())); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| -------------------------------------------------------------------------------------------------------------  1.ALG SIMPLE |
|  |

|  |
| --- |
| - how to make string start from capital letter |
|  |

|  |
| --- |
| 2.ALG ADVANCED |
|  |

- 2 words anagrams - the best O(n), edge cases

**public boolean** checkAnagram(String first, String second) {  
 **if** (first == **null** && second == **null**) {  
 **return true**;  
 }  
  
 **if** (first == **null** || second == **null**) {  
 **return false**;  
 }  
  
 **if** (first.length() != second.length()) {  
 **return false**;  
 }  
  
 Map<Character, Integer> characterCount = **new** HashMap<>();  
  
 countCharacters(characterCount, first, 1);  
 countCharacters(characterCount, second, -1);  
  
 **return** characterCount.isEmpty();  
}  
  
**private void** countCharacters(Map<Character, Integer> characterCount, String word, **int** change) {  
 **for** (Character character : word.toCharArray()) {  
 **int** amount = countAmount(characterCount, character, change);  
 **if** (amount == 0) {  
 characterCount.remove(character);  
 } **else** {  
 characterCount.put(character, amount);  
 }  
 }  
}  
  
**private int** countAmount(Map<Character, Integer> characterCount, Character character, **int** change) {  
 **if** (characterCount.get(character) == **null**) {  
 **return** change;  
 }  
  
 **return** characterCount.get(character) + change;  
}