



Ako funguje broker pri protokole AMQP?

- Ukladá správy od publishera do zásobníka aby ich mohol prečítať consumer
- Ihneď preposiela správy od publishera consumerovi
- **Ukladá správy od publishera do fronty aby ich mohol prečítať consumer**
- Ukladá správy od consumera do fronty aby ich mohol prečítať publisher

Aké 4 hlavné typy noSQL databáz poznáme? (Vymenujte)

- **Dokumentové databázy (Document databases)**
- **Kľúč-Hodnota databázy (Key-value databases)**
- **Stĺpcovo orientované databázy (Column-oriented databases)**
- **Grafové databázy (Graph databases)**

Aké noSQL DB poznáme?

- Stromová DB
- **Grafová DB**
- Hviezdicová DB
- **Dokumentová DB**
- Kruhová DB

Aké sú rôzne typy mapovania pre ORM? (Vymenujte)

- Mapovanie pomocou XML
- Mapovanie pomocou atribútov
- Mapovanie kódom
- Konvenčné mapovanie (Automapping)

Aký problém rieši návrhový vzor Visitor?

- Single Dispatch
- Double Dispatch
- Triple Dispatch
- Triple Dispatch

Aký typ architektúry používa Docker?

- Peer-to-peer
- Klient-Server
- Event-driven
- REST

Je možné definovať fallback volanie pri použití Feign client-a?

- Áno

Kedy sa odporúča použiť microservice architektúra?

- Pri vytváraní nenáročných aplikácií
- Pri vytváraní aplikácií vedeckého zamerania
- Pri vytváraní aplikácií, ktoré sa často menia a dopĺňajú
- Microservice architektúru sa neodporúča používať

Kedy sa odporúča použiť constructor based DI?

ak sa jedná o objekty, ktoré treba vždy zadať (bez nich vytváraný objekt nevie korektne fungovať)

Napr . povinne pole, uveďte meno a priezvisko

Ina odpoved:

Ak máme dva objekty, ktoré sú na sebe navzájom závislé. Vznikne tzv. kruhová závislosť (circular dependency) a program nebude vedieť tieto dva objekty vytvoriť (nemôže vytvoriť objekt A, pretože vyžaduje inštanciu objektu B, rovnako nemôže vytvoriť inštanciu objektu B, pretože ten vyžaduje inštanciu objektu A). V takomto prípade je nutné použiť pri jednom objekte iný typ DI (napr. setter) a problém sa vyrieši.

Ktorý vyhľadávací nástroj je IBA cloud based?

- ElasticSearch
- **Algolia**
- MeiliSearch
- Lucene
- Solr

Ktorá z možností je určená na Spring Messaging?

- Spring Batch
- **Spring Integration**
- Spring Data
- Spring Cloud

Ktorá z nasledujúcich databáz má grafovú štruktúru?

- Oracle
- Redis
- **Neo4j**
- MongoDB
- CouchDB

- Cassandra

Ktoré metódy nepatria vo frameworku GraphX do partition strategy?

- RandomVertexCut
- **RandomEdgeCut**
- EdgePartition1D
- CanonicalRandomVertexCut
- **EdgePartition3D**

Ktoré sú tri základné entrypointy v GraphQL?

- Query, Addition, Substitution
- **Query, Mutation, Subscription**
- Queue, Mutation, Substitution
- Queue, Addition, Subscription

Ktorý protokol gRPC používa?

- **HTTP 2.0**
- HTTP 1.1
- FTP
- MIME

Ktorý z nasledujúcich servisov nevyžaduje XML konfiguráciu?

- Keycloak a CAS
- **Spring Boot**
- CAS a Spring Boot
- Keycloak a Spring Boot

Ktorý z uvedených typov testovania softvéru nepatrí pod testovanie výkonnosti (performance testing)?

- Load testing
- Stress testing

- Spike testing
- **Functional testing**
- Endurance testing

Majme situáciu, v ktorej do aplikácie prirábame modul na konverziu videí. Chceme použiť komplexnú knižnicu na prácu s videami, pomocou ktorej vieme vytvoriť rýchly a efektívny VideoConverter. Aký štrukturálny návrhový vzor je vhodné v tejto situácii použiť?

- Composite
- Decorator
- Flyweight
- **Facade**

Na čo sa používa programovací jazyk R?

- **analýza dát, štatistika**
- tvorba webstránok a webových aplikácií
- mobilné aplikácie
- **čistenie dát a zobrazenie grafov**

Na čo slúži Consul K/V Store ?

- **Na držanie a poskytovanie konfiguračných properties pre aplikácie**
- Na zobrazovanie aktívneho health statusu aplikácií
- Na zobrazovanie dostupných services registrovaných v Consul

Objekty rodičovskej triedy by sa mali dať nahradiť objektami jej podtried bez toho, aby sa porušila aplikácia. O akom zo SOLID princípov hovorí táto veta?

- Single responsibility principle
- Open Closed principle
- **Liskov Substitution principle**
- Interface Segregation principle
- Interface Segregation principle

Označte pravdivé tvrdenia.

- SparkTorch je jeden z hlavných komponentov obsiahnutých v Apache Spark.
- **Integráciou PyTorch s Apache Spark vieme dosiahnuť rýchlejšie tréovanie neurónovej siete s veľkými dátami.**
- Knižnica TensorFlow je súčasťou Apache Spark.
- **Integrácia Spark s Tensorflow / Pytorch sa používa na distribuované tréovanie.**

Prečo je zakázaný cyklický import závislostí v jazyku Golang?

- Kvôli prehľadnosti kódu
- Aby bol beh programu rýchlejší
- **Aby bola kompilácia programu rýchlejšia**
- Chceli sa odlíšiť od iných jazykov

Pri ktorom z príkladov sa môžu antipatterny prejaviť v kóde?

- Nevhodne pomenované premenné
- **Bloky nepoužitého kódu**
- Rozdelovanie tried
- **Nedostatočná dokumentácia**
- Pravidelné unit testovanie

V akým prípadoch vie exchange v RabbitMQ správne priradiť prijímateľovi správu ktorú poslal odosielateľ?

- **Keď sa routing key aj binding key zhodujú**
- **Exchange je typu "Topic", routing key je foo.bar a binding key je foo.***
- Routing key je foo.bar a binding key je fo.bar
- Vždy keď mám nastavenú exchange typu "Header" (edited)

V návrhovom vzore MVC figurujú 3 vrstvy. Model, view a controller. Čo zabezpečuje vrstva controller?

- Vykresľuje údaje modelu a tiež generuje HTML výstup.
- **Prepája komponenty, spracováva požiadavky používateľov a odovzdáva ich do view na vykreslenie.**
- Zapuzdruje dáta aplikácie a komunikuje s databázovým systémom.
- Stará sa o reprezentáciu a manipuláciu s dátami v aplikácií.

Vyber pojmy tykajúce sa search engines

- **index**
- tabulka
- zaznam
- **dokument**
- kolekcia

Vyberte pravdivé tvrdenia o Spark ML?

- Spark ML je python knižnica umožňujúca riešiť problémy strojového učenia.
- **Spark ML je knižnica kompatibilná s viacerými jazykmi umožňujúca riešiť problémy strojového učenia.**
- Spark ML sa používa na zabezpečenie modelov strojového učenia.
- **Spark ML slúži na spracovanie veľkých objemov dát.**

Vyberte správne tvrdenia ohľadom Mediator a Observer patternu

- **Ak komponent urobí nejakú zmenu, Observer upozorní všetky komponenty, ktoré sú na neho napojené.**
- Ak komponent urobí nejakú zmenu, Mediator upozorní všetky komponenty, ktoré sú na neho napojené.
- **Mediator rozhodne, ktorý komponent má reagovať na zmenu iného komponentu.**
- Observer rozhodne, ktorý komponent má reagovať na zmenu iného komponentu.
- **Mediator eliminuje priame prepojenia a závislosti medzi komponentami.**
- **Observer nastavuje dynamické jednosmerné spojenie medzi komponentami.**

Čo nepatrí do návrhového vzoru "Factory method"?

- Creator
- Product
- **Client**
- Concrete Product
- Concrete Creator

Čo nepatrí do štruktúry "Iterator"?

- Iterator
- Iterator Collection
- **Product**
- Client
- Concrete Collection

Čo nepatrí k dokumentovým DB?

- MongoDB
- **Redis**
- CosmosDB
- DynamoDB

Čo nepodporuje Cassandra Query Language?

- Jedno partičné transakcie
- Kolekcie
- **Cudzíe kľúče**
- Sekondárne indexy

Čo sa považuje za výhody pre Spring Data?

- **Automatické generovanie queries**
- Paralelný mechanizmus
- **Repository pattern**

Čo vraví tzv. "Closing channel principle" , teda princíp uzatvárania kanálov v jazyku Golang?

- Kanály nie je potrebné manuálne uzatvárať, jazyk Go sa o to postará automaticky
- **Kanály by sa mali uzatvárať zo zapisovateľa a to len vtedy ak existuje len 1 zapisovateľ**
- Kanály by sa mali uzatvárať z čitateľa

Čo znamená skratka gRPC?

- **gRPC Remote Procedure Calls**
- Google Remote Procedure Calls
- Geocaching Relative Procedure Calls
- gRPC Remote Pointless Calls

Čo znamená skratka SAML?

- Security Authorisation Markup Language
- Secured Authentication Markup Library
- **Security Assertion Markup Language**
- Single Asset Markup Language

Pri ktorých z nasledujúcich operácii spark-api môže nastať premiešanie (shuffle)?

správne	nesprávne
<ul style="list-style-type: none">• coalesce• Cogroup• groupBy• join• repartition• sortByKey• Všetky *ByKey (groupByKey, reduceByKey...) - okrem countByKey	<ul style="list-style-type: none">• Filter• flatMap• flatMapValues• foreach• map• Sample• Union

Ktoré z nasledujúcich operácii spark-api sú transformácie?

správne	nesprávne
<ul style="list-style-type: none">• distinct• Filter• flatMap• flatMapValues• groupBy• intersection• map• mapToPair• reduceByKey• sample• sortByKey• union	<ul style="list-style-type: none">• collect• count• first• foreach• parallelize• reduce• textFile

Transformácie definícia:

transformujú RDD objekt na iný RDD objekt, vstup a výstup je na workeroch

Narrow transformácia sú: **map, FlatMap, MapPartition, Filter, Sample, Union**

Wide transformácie sú: **Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition, Coalesce**

Ktoré z nasledujúcich operácii spark-api sú akcie?

správne	nesprávne
<ul style="list-style-type: none">• aggregate• Collect• count• countByValue• first• fold• foreach• reduce• take(n)• top	<ul style="list-style-type: none">• distinct• filter• flatMap• groupBy• map• Parallelize• reduceByKey• sample• sortByKey• textFile

Akcie definícia: zobrazujú upravené výstupy, pracujú priamo s daným RDD objektom, výstup je na driveroch

Ktoré z uvedených lambda výrazov spĺňajú podmienky kladené na argumenty operácie filter?

Predpokladajte, že argumenty a, b sú čísla, s, t sú reťazce

hint(Musi vrátiť len true alebo false, filter nevie pracovať s 2 a viac parametrami (x,y))

správne	nesprávne
<ul style="list-style-type: none">• <code>S -> "hello".statWith(s)</code>• <code>a -> a - 3 > 0</code>• <code>s->s.isEmpty()</code>• <code>A -> true</code>	<ul style="list-style-type: none">• <code>(a,b) -> a && b</code>• <code>s->s.size()</code>• <code>A -> 1.0</code>• <code>(s, t) -> s.equals(t)</code>

Ktoré z uvedených lambda výrazov spĺňajú podmienky rýdzej funkcie (pure function)

Môžete predpokladať, že argumenty:

- a,b sú čísla, s reťazec, u je objekt, ktorý má property urok,
- x je lokálna premenná, y globálna premenná

správne	nesprávne
<ul style="list-style-type: none">• <code>s -> new Tuple2(s, 1)</code>• <code>s -> new Tuple2(a, a*a)</code>• <code>s -> s.isEmpty()</code>• <code>s -> "hello".statWith(s)</code>• <code>a -> { int x=a>0?1:0; return x*a; }</code>• <code>(a,b) -> a + b</code>• <code>a -> Integer.MIN_VALUE + a</code>• <code>s -> { if ("hello".statWith(s)) return true; return false; }</code>• <code>a -> { int x=a>0?1:0; return x*a; }</code>	<ul style="list-style-type: none">• <code>u -> { Urok x=new Urok(u); x.setUrok(0.1); return x; }</code>• <code>s -> {system.out.println(s); return s.length();}</code>• <code>a -> a * Math.random()</code>• <code>a -> {return a * y;}</code>• <code>a -> a * y</code>• <code>a -> { return Math.random() *a;}</code>• <code>u -> { double x = u.getUrok(); system.out.println("urok="+x); }</code>• <code>u -> {u.setUrok(0.1);}</code>• <code>s -> system.out.println(s)</code>• <code>A -> {Date d = new Date(); return s+d;}</code>

Rýdza funkcia (pure function) definícia: jej vstupné parametre sa nemenia za behu programu, nesmie ovplyvňovať premenné mimo funkciu, pri rovnakom vstupe dá vždy rovnaký výstup (čiže nesmie obsahovať rng) a nesmie mať žiadny side-effect (napríklad výpis do konzoly)

Uzáver funkcie definícia: enkapsulujú niektoré private premenné, klient k nim vie pristupovať len cez getters a setters

Ktoré z uvedených lambda výrazov spĺňajú podmienky kladené na argumenty operácie reduce?

Predpokladajte, že argumenty a, b sú čísla, s, t sú reťazce

hint(Musí to byť komutatívna a asociatívna binárna operácie, vstup viac arg, výstup 1 arg)

správne	nesprávne
<ul style="list-style-type: none"> • <code>(a, b) -> Math.min(a, b)</code> • <code>(a, b) -> a && b</code> • <code>(a, b) -> a + b</code> • <code>(a,b) -> a * b</code> • <code>(a, b) -> a b</code> • <code>(a, b) -> Math.max(a, b)</code> 	<ul style="list-style-type: none"> • <code>(a,b) -> a-b</code> • <code>(a,b) -> a</code> • <code>(s, t) -> s.split(t)</code> • <code>(s,t)->s.equals(t)</code> • <code>(a,b) -> a % b</code> • <code>(a,b) -> (a+b)/2</code>

Ktoré z uvedených lambda výrazov spĺňajú podmienky kladené na argumenty operácie map?

Predpokladajte, že argumenty a, b sú čísla, s, t sú reťazce

správne	nesprávne
<ul style="list-style-type: none"> • <code>A -> new Tuple2(a, a*a)</code> • <code>A -> true</code> • <code>s -> s.isEmpty()</code> • <code>s -> { return s.length(); }</code> • <code>a -> 1.0</code> • <code>s -> s.substring(1, 3)</code> • <code>a -> a > 0</code> • <code>a -> {int y=a>0?1:0; return a*y;}</code> 	<ul style="list-style-type: none"> • <code>a -> { int y = a > 0 ? 1 : 0; a * y; }</code> • <code>a -> { a++; }</code> • <code>s -> {s.length() + 2;}</code>

Ako môže funkcia, ktorá je argumentom operácie filter, pracovať s akumulátorom?

- môže ho len modifikovať

Ako môže funkcia, ktorá je argumentum operácie map, pracovať s broadcast objektom?

- môže ho len čítať

Ako môže funkcia, ktorá je argumentom operácie foreach, pracovať s akumulátorom?

- môže ho len modifikovať

Čo vypíše nasledujúci program?

```
1. static int LIMIT;  
2.  
3. public static void main(String[] args) {  
4.     SparkConf conf = new SparkConf();  
5.     JavaSparkContext sc = new JavaSparkContext(conf);  
6.     LIMIT = 20;  
7.     List<Integer> dl = Arrays.asList(1, 10, 100, 1000);  
8.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);  
9.     JavaRDD<Integer> rdd2 = rdd1.filter(x -> x<LIMIT);  
10.    LIMIT=200;  
11.    rdd2.cache().collect();  
12.    System.out.println("" + rdd2.count());  
13.}
```

- 3

Čo vypíše nasledujúci program?

```
1. static int LIMIT;  
2.  
3. public static void main(String[] args) {  
4.     SparkConf conf = new SparkConf();  
5.     JavaSparkContext sc = new JavaSparkContext(conf);  
6.     LIMIT = 20;  
7.     List<Integer> dl = Arrays.asList(1, 10, 100, 1000);  
8.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);  
9.     JavaRDD<Integer> rdd2 = rdd1.filter(x -> x<LIMIT);  
10.    rdd2.cache().collect();  
11.    LIMIT=200;  
12.    System.out.println("" + rdd2.count());  
13.}
```

- 2

Čo vypíše nasledujúci program?

```
1. static int LIMIT;  
2.  
3. public static void main(String[] args) {  
4.     SparkConf conf = new SparkConf();  
5.     JavaSparkContext sc = new JavaSparkContext(conf);  
6.     LIMIT = 20;  
7.     List<Integer> dl = Arrays.asList(1, 10, 100, 1000);  
8.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);  
9.     JavaRDD<Integer> rdd2 = rdd1.filter(x -> x<LIMIT);  
10.    LIMIT=200;  
11.    rdd2.cache().collect();  
12.    System.out.println("" + rdd2.count());  
13.}
```

- 3

Čo vypíše nasledujúci program?

```

1. static int ZLAVA;
2.
3. public static void main(String[] args) {
4.     SparkConf conf = new SparkConf();
5.     JavaSparkContext sc = new JavaSparkContext(conf);
6.
7.     ZLAVA = 25;
8.     List<Integer> dl = Arrays.asList(100, 110, 120, 130, 140, 150);
9.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);
10.    JavaRDD<Integer> rdd2 = rdd1.map(x -> x - ZLAVA);
11.    ZLAVA = 5;
12.    JavaRDD<Integer> rdd3 = rdd2.map(x -> x - ZLAVA).map(x -> x - ZLAVA);
13.    System.out.println("" + rdd3.filter(x -> x < 100).count());
14.}

```

- 2

Čo vypíše nasledujúci program?

```

1. public static void main(String[] args) {
2.     SparkConf conf = new SparkConf();
3.     JavaSparkContext sc = new JavaSparkContext(conf);
4.     int LIMIT = 20;
5.     List<Integer> dl = Arrays.asList(1, 10, 100, 1000);
6.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);
7.     JavaRDD<Integer> rdd2 = rdd1.filter(x -> x<LIMIT);
8.     LIMIT=200;
9.     System.out.println("" + rdd2.count());
10.}

```

- build error

Čo vypíše nasledujúci program?

```
1. public static void main(String[] args) {  
2.     SparkConf conf = new SparkConf();  
3.     JavaSparkContext sc = new JavaSparkContext(conf);  
4.  
5.     int ZLAVA = 25;  
6.     List<Integer> dl = Arrays.asList(100, 110, 120, 130, 140, 150);  
7.     JavaRDD<Integer> rdd1 = sc.parallelize(dl);  
8.     JavaRDD<Integer> rdd2 = rdd1.map(x -> x - ZLAVA);  
9.     ZLAVA = 5;  
10.    JavaRDD<Integer> rdd3 = rdd2.map(x -> x - ZLAVA).map(x -> x - ZLAVA);  
11.    System.out.println("" + rdd3.filter(x -> x < 100).count());  
12.}
```

- build error

Čo je zámerom návrhového vzoru Abstract Factory?

- Poskytnúť rozhranie na vytváranie objektov viacerých tried, pričom voľbu konkrétnych tried necháte na implementáciu

Čo je zámerom návrhového vzoru Adapter?

- Zabaliť existujúci objekt do nového rozhrania
- Prispôbiť rozhranie existujúceho objektu potrebám klienta

Čo je zámerom návrhového vzoru Bridge?

- Umožniť voľbu implementácie nezávisle od voľby abstrakcie/rozhrania
- Oddeliť abstrakciu od implementácie

Čo je zámerom návrhového vzoru Builder?

- Oddeliť vytváranie komplexných objektov od ich reprezentácie (detailnej špecifikácie)

Čo je zámerom návrhového vzoru Chain of Responsibility?

- Umožniť klientovi odoslať príkaz aj bez toho, aby vedel kto ho vykoná

Čo je zámerom návrhového vzoru Composite?

- Poskytnúť jednotné rozhranie pre prácu so samostatnými objektami aj kontajnerom.
- Reprezentovať komplexný objekt ako stromovú štruktúru.

Čo je zámerom návrhového vzoru Command?

- Umožniť narábať s operáciou ako objektom
- Zabaliť príkaz do objektu

Čo je zámerom návrhového vzoru Observer?

- Poskytnúť možnosť reakcie na udalosť/informáciu viacerým objektom

Čo je zámerom návrhového vzoru Prototype?

- Vytvoriť nový objekt kopírovaním.

Čo je zámerom návrhového vzoru Proxy?

- Vytvoriť prostredníka, ktorý bude umožňovať prístup k objektu
- Poskytnúť zástupcu, ktorý rezervuje miesto pre skutočný objekt
- Poskytnúť prostredníka, ktorý bude kontrolovať prístup k objektu

Čo je zámerom návrhového vzoru Singleton?

- Zabezpečiť, že bude vytvorená jediná inštancia triedy

Čo je zámerom návrhového vzoru Visitor?

- Poskytnúť operáciu pracujúcu s objektami rôznych typov (tvoriacich zložitejšiu štruktúru)

Čo je zámerom návrhového vzoru Decorator?

- Pridať funkcionality objektu bez nutnosti vytvoriť podtriedu

Čo je zámerom návrhového vzoru Template Method?

- Vytvoriť kostru algoritmu a detaily prenechať na podtriedy

Čo je zámerom návrhového vzoru Factory Method?

- Poskytnúť rozhranie na tvorbu objektu, pričom rozhodnutie aký objekt sa vytvorí necháte na implementáciu

Potrebujete počítať prístupy k objektu. Aký návrhový vzor by ste použili?

- Proxy

Potrebujete mať možnosť výberu implementácie aj rozhrania komponenty nezávisle na sebe. Aký návrhový vzor by ste použili?

- Bridge

Potrebujete implementovať komunikačnú architektúru PUBLISH-SUBSCRIBE. Aký návrhový vzor by ste použili?

- Observer

Potrebujete rozšíriť funkcionality triedy bez použitia podtried. Aký návrhový vzor by ste použili?

- Decorator

Komponenta, ktorú chcete použiť, nemá rozhranie vyhovujúce vašim dátovým objektom. Aký návrhový vzor by ste použili?

- Adapter

Potrebujete zabezpečiť aby globálny zdieľaný prístupový bod k databáze/mailovému serveru/window- manageru... Aký návrhový vzor by ste použili?

- **Singleton**

Potrebujete zabezpečiť aby existovala len jediná inštancia vašej triedy. Aký návrhový vzor by ste použili?

- **Singleton**

Potrebujete vytvárať objekty pričom ich vytváranie je veľmi náročné na čas a/alebo zdroje. Aký návrhový vzor by ste použili?

- **Prototype**

Potrebujete kontrolovať, kto má prístup k objektu. Aký návrhový vzor by ste použili?

- **Proxy**

Pri implementácii aplikácie / frameworku viete, kedy sa vytvára inštancia istého objektu, nepoznáte však ešte konkrétnu triedu. Aký návrhový vzor by ste použili?

- **Factory Method**

Udalosťami riadená aplikácia potrebuje poskytnúť UNDO podporu pre akcie. Aký návrhový vzor by ste pri tom využili?

- **Command**

Požiadavka nemôže byť spracovaná hneď ako bola vygenerovaná, ale treba čakať na vhodný okamih. Aký návrhový vzor by ste použili?

- **Command**

Implementujete GUI framework, ktorý má podporovať viaceré look-and-feel a témy. Aký návrhový vzor by ste tu využili?

- **Abstract Factory**

Pre ktorý návrhový vzor je charakteristická metóda accept()?

- Visitor

Ktorý návrhový vzor môže pri svojej implementácii využiť wrapper?

- Adapter
- Decorator

Pre ktorý návrhový vzor je charakteristická metóda getInstance()?

- Singleton

Pre ktorý návrhový vzor je charakteristická metóda execute()?

- Command

Pre ktorý návrhový vzor je charakteristická metóda clone()?

- Prototype

Pre ktorý návrhový vzor sú charakteristické metódy attache() a detach()?

- Observer

Pre ktorý návrhový vzor sú charakteristické metódy update() a notify()?

- Observer

Pre ktorý návrhový vzor sú typické komponenty Abstraction a Implementor?

- Bridge

Ktorý návrhový vzor je alternatívou pre objektovo-orientovaný callback?

- Command

Ktorý návrhový vzor obsahuje rekurzívnu štruktúru?

- Composite
- Chain of Responsibility

Pri implementácii aplikácie/frameworku viete, kedy sa vytvára inštancia istého objektu, nepoznáte však ešte konkrétnu triedu. Aký návrhový vzor by ste použili?

- **Factory Method**

Priradte, každému návrhovému vzoru kategóriu do ktorej patrí.

Abstract Factory	> Creational
Adapter	> Structural
Bridge	> Structural
Builder	> Creational
Chain of Responsibility	> Behavioral
Command	> Behavioral
Composite	> Structural
Decorator	> Structural
Factory Method	> Creational
Observer	> Behavioral
Prototype	> Creational
Proxy	> Structural
Singleton	> Creational
Template Method	> Behavioral
Visitor	> Behavioral

Ktorý návrhový vzor je alternatívou pre statický objekt?

- **Singleton**

Stručne (max. 1-2 vetami) porovnajte návrhové vzory Adapter a Decorator.

Adaptér je určený na zmenu rozhrania existujúceho objektu. Dekorátor vylepšuje iný objekt bez zmeny jeho rozhrania.

Iná odpoveď:

Adaptér konvertuje rozhranie triedy na iné rozhranie, s ktorým klienti dokážu pracovať. Decorator sa používa na dynamické rozšírenie funkcionality triedy.

Iná odpoveď:

Adapteri nam dovoluju pridavat' interface-i ku našim classam(lahko ich môžeme meniť) a Decorator nam dynamicky (už po spustení) pridava (dekoruje) naše objekty.

Iná odpoveď:

Adapter: konvertuje interface do iného kompatibilného interfacu
Decorator : dynamicky rozširuje funkcionalitu danej triedy

Stručne (max. 1-2 vetami) porovnajte návrhové vzory Adapter a Bridge.

Adapter umoznuje, aby veci fungovali po tom co boli navrhnuté. Bridge umoznuje, aby veci fungovali predtym. Bridge je navrhnuty dupredu, aby sa abstrakcia a implementacia mohli mohli menit nezávisle. Adapter je zase navrhnuty tak, aby nesúvisiace triedy spolupracovali.

Iná odpoveď:

Bridge: oddeluje abstrakciu od implementácie, aby sa mohli lísiť a riešiť nezávisle na sebe, toto nechá kód klienta nezmenený
Adapter: konvertuje interface do iného kompatibilného interfacu

Stručne (max. 1-2 vetami) vysvetlite, ako súvisia vzory Command a Chain of Responsibility.

Handler-i v Chain of Responsibility môžu byť implementované ako Command-y.

Oba vykonávajú príkazy, pričom nevedia nič o prijímateľovi a nepoznajú detaily operácie

Inak (1 z 2b.):

Obidva vzory vykonávajú žiadosti bez toho, aby niečo vedeli o požadovanej operácii alebo o prijímateľovi žiadosti

Comments:

neuplne

CoR je často vykonavateľom operácie reprezentovanej commandom

Viac:

command is basically just a command encapsulated in an object. Chain of responsibility is more an object trying to handle something & if not, pass it onto the next one in the 'chain'. In chain of responsibility pattern you do not have the chance to undo, save or queue the actions. If you need to do that you have to use command pattern. If you want to execute the operation in a different time than use command pattern. If more than one object can handle a request use chain of responsibility.

Stručne (max. 1-2 vetami) porovnajte návrhové vzory Proxy a Adapter.

Proxy používa pre všetky objekty rovnaký interface (wrapper). Adapter konvertuje existujúci interface na odlišný, aby docielil kompatibilitu s iným rozhraním

Stručne vysvetlite rozdiely medzi návrhovými vzormi Template method a Strategy

Pri strategy sa typicky vytvorí rozhranie(interface) a vykonanie akcie sa deleguje na implementáciu daného rozhrania. Pri template sa využíva dedičnosť a podtrieda override-ne požadované metódy a vykoná v nich potrebné kroky.

Stručne vysvetlite rozdiely medzi návrhovými vzormi Adapter a Facade

Adapter sa používa keď wrapper musí použiť určené rozhranie a podporovať polymorfizmus. Používa sa na prepojenie dvoch nekompatibilných rozhraní. Facade sa používa na uľahčenie práce s rozhraním alebo jeho zjednodušenie, napríklad spojenie viacerých malých úkonov do jedného volania.

S akým návrhovým vzorom súvisí uzáver (closure) funkcie? Stručne vysvetlite.

Je správne? V ASOS_RT_21.pdf to je za 3b, není to krátke moc? **Potrebujeme ešte vysvetlenie**

Command pattern.

Vymenujte základné pojmy AOP

- ASPECT - je modul API volaní, ktorý poskytuje danú funkčnosť
- ADVICE (before, after, after-returning, after-throwing, around) - akcia vykonaná aspektom na určité jointpoint
- JOIN POINT - je bod vykonania programu ako je napr. vykonanie metódy alebo spracovanie výnimky.
- POINTCUT (najbežnejšie execution pointcut) - je to predikát alebo výraz, ktorý sa zhoduje s jointpoint
- INTRODUCTION
- TARGET OBJECT
- AOP PROXY
- WEAVING

Rozsiahlejšia odpoveď za 3/3b:

- Aspekt – Prostriedok modulárneho vyjadrenia pretínajúcich aktivít
- Join point – dobre definované miesto v kóde programu komponentu, v ktorom je možné pripojenie kódu rady aspektu
- Advice – definuje kód pripájaný v označených bodoch spojenia
- Pointcut – vyberá (označuje) body spojenia, ku ktorým je pripojený kód rady
- Weaving – program vykonávajúci spojenie (linkovanie) komponentov a aspektov do výsledného kódu
- Component – zdrojový kód, do ktorého sú nalinkovane aspekty

Stručne (max. 1-2 vetami) vysvetlite, čím sa líšia a čo majú spoločné návrhové vzory Builder a Abstract Factory.

Builder sa zameriava na konštrukciu zložitejších objektov krok po kroku. Abstract Factory sa špecializuje na vytváranie skupín príbuzných objektov. AF vráti inštanciu okamžite, zatiaľ čo Builder umožní pred načítaním inštancie vykonať niekoľko ďalších konštrukčných krokov.

Iné vysvetlenie:

Oba sa používajú na vytváranie objektov. Builder sa používa na vytváranie potencionálne zložitejších objektov po častiach (pomocou niekoľkých príkazov). Builder teda nevráti inštanciu okamžite, ale až na konci, napríklad zavolaním funkcie na to určenej (napr. build()). Abstract Factory sa používa na vytváranie podobných objektov a inštanciu vráti typicky okamžite.

Stručne (max. 1-2 vetami) vysvetlite, čím sa líšia a čo majú spoločné návrhové vzory Prototype a Factory Method.

prototype je používaní ak je "cena" vytvorenie nového objektu vysoká a je v pohode, že novú inštanciu urobíme kopírovaním starej (vytvorenie klonu) a Factory method nám zase spraví na novo celú inštanciu aj s potrebnými metodami

Inak: (2 z 2b.)

Oba slúžia na vytváranie objektov. Factory Method definuje interface na vytváranie objektov ale o tom, ktorá trieda sa vytvorí nechá rozhodovať podtriedy. Prototype vytvára nové objekty kopírovaním prototypu.

Inak: (2 z 2b.)

Factory method je rozhranie ktore umoznuje vytvaranie objektov, vytvarame pomocou neho objekty generickym sposobom. Napriklad ked vytvaranie objektu ma navyse nejaku zavislost ktoru nechceme zahrnut do konkretnej triedy

Prototype zas vytvara objekty pomocou klonovania inych objektov, vsetky objekty maju podobnu strukturu avsak prototype nie je zalozeni na dedeni, pricom factory method je zalozeny na dedeni.

Stručne (max. 1-2 vetami) vysvetlite, čím sa líšia a čo majú spoločné návrhové vzory Proxy a Adapter.

Adapter poskytuje rozdielny interface pre dany objekt a Proxy poskytuje ten isty interface pre dany objekt. Adapter je urceny pre zmenu interfacu existujuceho objektu. Oba navrhove vzory implementuju objekt pre ovladanie ineho objektu.

Vymenujte základné princípy, na ktorých stojí architektúra frameworku Spring.

Inversion of Control - Každý modul sa "sústredí" iba na to, na čo je určený

Dependency Injection - Vytváranie objektov, na ktoré sa spoliehajú iné objekty počas compile-time (automatické vytvorenie a poskytnutie objektu, ktorý pre svoju funkcionálnosť potrebuje iný objekt)

Aspect-Oriented Programming (AOP) - Na oddelenie Cross-Cutting logiky od Business logiky aplikácie

Vymenujte typy Dependency Injection, ktoré podporuje Spring

Setter injection, Constructor injection, field injection, lookup method injection

Uved'te rôzne typy proxy objektov

remote, virtual, protection

Uved'te dva rôzne spôsoby implementácie adaptéra

object, class

Vysvetlite stručne čo popisuje WSDL element <binding>

Spôsob serializácia vstupov a výstupov do XML

The <binding> element provides specific details on how a *portType* operation will actually be transmitted over the wire.

Ina odpoved za 3/3b:

Poskytuje konkrétne podrobnosti o tom, ako sa operácia typu portType prenesie.

Čo popisuje WSDL element <portType>

Odpoved za 2,5 z 3b:

Používa sa na definovanie jednej alebo viacerých operácií, tie sú definované ako vstupno / výstupné vzory - istého typu spravy.

Viac:

The <portType> element combines multiple message elements to form a complete one-way or round-trip operation.

For example, a <portType> can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations.

Čo popisuje WSDL element <message>

The messages used by the web service

Čo popisuje WSDL element <types>

The data types used by the web service

Na definíciu typov sa používa XML-schema jazyk.

Predpokladajte, že sme do kolekcie `JavaRDD<String> rdd` načítali riadky textového súboru. S využitím operácií RDD-api napíšte výraz, ktorý vráti počet rôznych slov dlhších ako 2 (Pozn. riadky treba rozdeliť na slová)

```
rdd.flatMap(line -> Arrays.asList(line.split(" ")).iterator()).distinct().filter(word -> word.length() > 2).count();
```

Predpokladajte, že máte dve kolekcie `JavaRDD<String> rd1` a `JavaRDD<String> rd2` obsahujúce reťazce. S využitím operácií RDD-api napíšte výraz pre výpočet symetrickej diferencie množín reťazcov t.j. celkového počtu rôznych reťazcov, ktoré sa nachádzajú práve v jednej z kolekcií (ale nie v oboch)

```
rd1.union(rd2).distinct().subtract(rd1.intersection(rd2)).count();
```

Predpokladajte, že máte dve kolekcie `JavaRDD<String> rd1` a `JavaRDD<String> rd2` obsahujúce reťazce. S využitím operácií RDD-api napíšte výraz pre výpočet symetrickej diferencie množín reťazcov t.j. celkového počtu reťazcov, ktoré sa nachádzajú len v jednej z kolekcií ale nie v oboch. Reťazce líšiace sa len **VEL'KOSTOU PÍSMEN** považujte za totožné.

```
rd1 = rd1.map(s -> s.toUpperCase());  
rd2 = rd2.map(s -> s.toUpperCase());  
  
return rd1.union(rd2).distinct().subtract(rd1.intersection(rd2)).count();
```

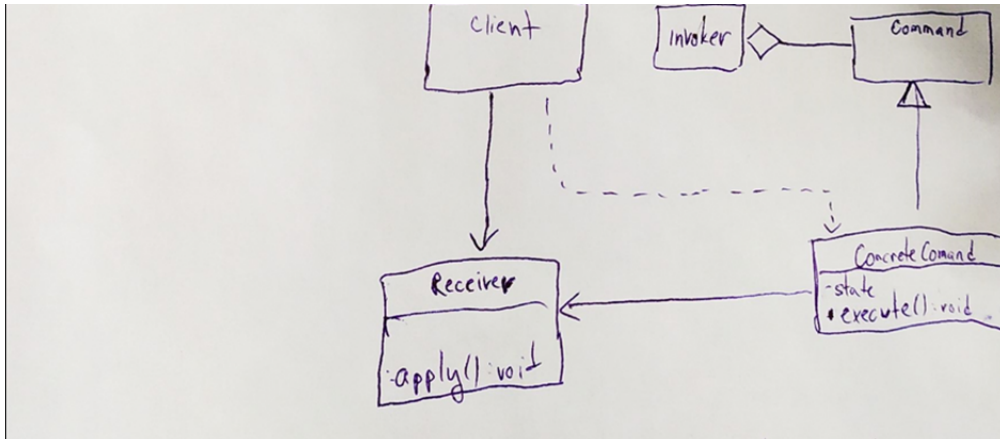
Predpokladajte, že `JavaRDD<String> rdd` je kolekcia reťazcov. S využitím operácií RDD-api napíšte výraz, ktorého výstupom je boolovská hodnota hovoriaca či sú v kolekcii duplicitné reťazce, t.j. ak sú má výraz hodnotu `true` inak `false`.

```
rdd.distinct().count() != rdd.count();
```

Charakterizujte návrhový vzor Command

- Nakreslite UML class-diagram a sequence diagram
- Vymenujte triedy/objekty tvoriace navrhovy vzor a charakterizujte strucne ich ulohu v nom

-ukladá requesty do fronty, requesty sa môžu vykonávať v odlišnom čase



- Client - nastavi na zaciatku receivera a vytvori Concrete Command
- Receiver - aplikuje request
- ConcreteCommand - metodu execute vykonava request
- Command - interface, ktory cez ConcreteCommand plni request
- Invoker - vola jednotlivé requesty

Predpokladajte, že JavaRDD<String> rdd je kolekcia slov. S využitím operácií RDD-api napíšte výraz, ktorého výstupom je mapa (java.util.map) početností jednotlivých slov v kolekcii (t.j. kľúč je slovo a hodnota je počet výskytov). Slová líšiacie sa len veľkosťou písmen považujte za totožné.

```
rdd.map(s -> s.toLowerCase()).countByValue();
```

Predpokladajte, že máte dve kolekcie JavaRDD<String> rdd1 a JavaRDD<String> rdd2 obsahujú riadky dvoch textových súborov. S využitím operácií RDD-api napíšte výraz, ktorý vráti zoznam (java.util.List) obsahujúci všetky rôzne slová, ktoré sa nachádzajú v prvom súbore ale nenachádzajú v druhom. (Pozn. riadky treba rozdeliť na slová)

```

rdd1.flatMap(s -> Arrays.asList(s.split(" ")).iterator())

    .subtract(

        rdd2.flatMap(s -> Arrays.asList(s.split(" ")).iterator())

    ).collect();

```

Predpokladajte, že sme do kolekcie `JavaRDD<String> rdd` načítali riadky textového súboru. S využitím operácií RDD-api napíšte výraz, ktorého výstupom je mapa (`java.util.map`) početností jednotlivých slov v súbore (t.j. kľúč je slovo a hodnota je počet výskytov). (Pozn. riadky treba rozdeliť na slová)

- `rdd.flatMap(s -> Arrays.asList(s.split(" ")).iterator()).countByValue();`

Predpokladajte, že kolekcie `JavaRDD<String> rdd1` a `JavaRDD<String> rdd2` obsahujú riadky dvoch textových súborov. S využitím operácií RDD-api napíšte výraz, ktorý vráti zoznam (`java.util.List`) obsahujúci všetky rôzne slová, ktoré sa nachádzajú v prvom súbore ale nenachádzajú v druhom. (Pozn. riadky treba rozdeliť na slová)

- `rdd1.flatMap(s -> Arrays.asList(s.split(" ").iterator()).subtract(rdd2.flatMap(s -> Arrays.asList(s.split(" ").iterator()))).collect()`

**Aký návrhový vzor sa uplatní pri parsovaní XML-dokumentov?
Stručne vysvetlite.**

Napr. Factory a Builder
ParserFactory na vytváranie parsovacích generatorov ako je napr SaxParserFactory alebo DocumentBuilderFactory,
kdežto pre samotný Document by sme mohli použiť Builder pattern na vytváranie takýchto objektov.

Comments:

1b za snahu

správna odpoveď je Visitor

Visitor – podobné operácie sa vykonávajú na objektoch rôznych typov, ktoré spolu tvoria určitú štruktúru. Môžeme vytvoriť samostatnú triedu konkrétnych visitorov, ktorí budú zodpovední za spracovanie rôznych XML elementov.

Predpokladajte, že máte dve kolekcie JavaRDD<String> rd1 a JavaRDD<String> rd2 obsahujúce reťazce. S využitím operácií RDD-api napíšte výraz pre výpočet symetrickej diferencie množín reťazcov t.j. Celkového počtu reťazcov, ktoré sa nachádzajú len v jednej z kolekcií (ale nie v oboch). Reťazce líšiacie sa len prázdnyimi znakmi na začiatku a konci reťazca považujte za totožné.

```
rd1 = rd1.map(s -> s.trim());  
rd2 = rd2.map(s -> s.trim());  
  
rd1.union(rd2).subtract(rd1.intersection(rd2)).count();
```

Predpokladajte, že JavaRDD<String> rdd je kolekcia reťazcov. S využitím operácií RDD-api napíšte výraz, ktorého výstupom je boolovská hodnota hovoriaca či sú v kolekcií duplicity (t.j. ak sa v kolekcií vyskytuje reťazec viac krát výraz hodnotu true inak false). Reťazce líšiacie sa len veľkosťou písmen považujte pri tom za rovnaké.

```
rdd.map(s -> s.toLowerCase()).distinct().count() !=  
rdd.count();
```

// TODO - Exercise 11

Predpokladajte, že sme do kolekcie JavaRDD<String> rdd načítali riadky textového súboru. S využitím operácií RDD-api napíšte výraz, vráti počet rôznych slov v súbore, pričom slová líšiacie sa len veľkosťou písmen považujte za totožné. (Pozn. riadky treba rozdeliť na slová)

```
rdd.flatMap(s -> Arrays.asList(s.split(" ")).iterator()).map(s ->  
s.toLowerCase()).distinct().count();
```

// TODO - Exercise 13

Predpokladajte, že JavaPairRDD pdd je kolekcia dvojíc, kde prvá zložka je meno študenta druhá názov predmetu, ktorý má zapísaný. S využitím operácií RDD-api napíšte výraz, ktorý pre každého študenta vypíše na štandardny výstup riadok obsahujúci meno študenta a reťazec zložený z názvov jeho predmetov oddelených čiarkou. (napr. Fero ASOS,VSA,RZZ)

```
pdd.collectAsMap().foreach((k,v) -> System.out.println(k + " " +  
    String.join(", ", v)));
```

Spark rozdeluje funkcie na prácu s distribuovanými dátami na akcie a transformácie. Vysvetlite v čom je hlavný rozdiel medzi nimi.

Transf- transformuju RDD objekt na iný RDD. Vstup a výstup je na workeroch.

Akcie- zobrazujú upravené výstupy, výstup je na driveroch

Stručne (max. 1-2 vetami) vysvetlite, ako súvisia vzory Visitor a Composit.

Poskytnúť operáciu pracujúcu s objektami rôznych typov (tvoriacich zložitejšiu štruktúru) - visitor

Poskytnúť jednotné rozhranie pre prácu so samostatnými objektami aj kontajnerom. - composite

Visitor rieši poskytnutie operácie pracujúcej s objektami rôznych typov a composite poskytuje jednotné rozhranie pre prácu so samotnými objektami aj kontajnerom

S akými návrhovými vzormi súvisí AOP

proxy, singleton, template method, decorator, observer, command, CoR

Uveďte, ktoré návrhové vzory bývajú zvyčajne implementované ako Singleton.

Napr. Abstract factory, Prototype, Builder. Ale napríklad aj Facade sa zvykne implementovať ako singleton lebo jeden facade objekt je vo väčšine prípadov postacujúci.

Vysvetlite stručne úlohu IoC kontajnera.

- moznost na upravu a spravu objektov pomocou reflection**
- zodpoveda za iniciáciu, konfiguráciu a zostatovanie beanov**
- je zodpovedný za správu životného cyklu objektov**

IoC kontajner je veľmi výhodný pri developovaní aplikácii/web app. ,pretože sa v nom lahko "testuju/vymienaju" veci. Tzn, že kludne môžeme jeden kontajner zmeniť za druhý z úplne iným významom a kludne sa môžeme k nemu naspäť vrátiť, ak by bol nový zlý atď.. Tak isto, vo vnútri jedného kontajnera, je práca uľahčená, pri výmenách/testoch kodu

IoC kontajner je veľmi výhodný pri developovaní aplikácii/web app. ,pretože sa v nom lahko "testuju/vymienaju" veci. Tzn, že kludne môžeme jeden kontajner zmeniť za druhý z úplne iným významom a kludne sa môžeme k nemu naspäť vrátiť, ak by bol nový zlý atď.. Tak isto, vo vnútri jedného kontajnera, je práca uľahčená, pri výmenách/testoch kodu

Komentár:

ale konkrétne> slúži na vytváranie inicializáciu a manažovanie komponent tvoriacich aplikáciu

Inak: (2 z 2b.)

slúži na implementáciu automatického DI. Vytvára požadované objekty, spravuje ich životnosť a automaticky ich injectuje do potrebných objektov.

Inak:

IoC container slúži na manažovanie životného cyklu objektov (beanov) a závislostí objektov pomocou dependency injection.

Komentár:

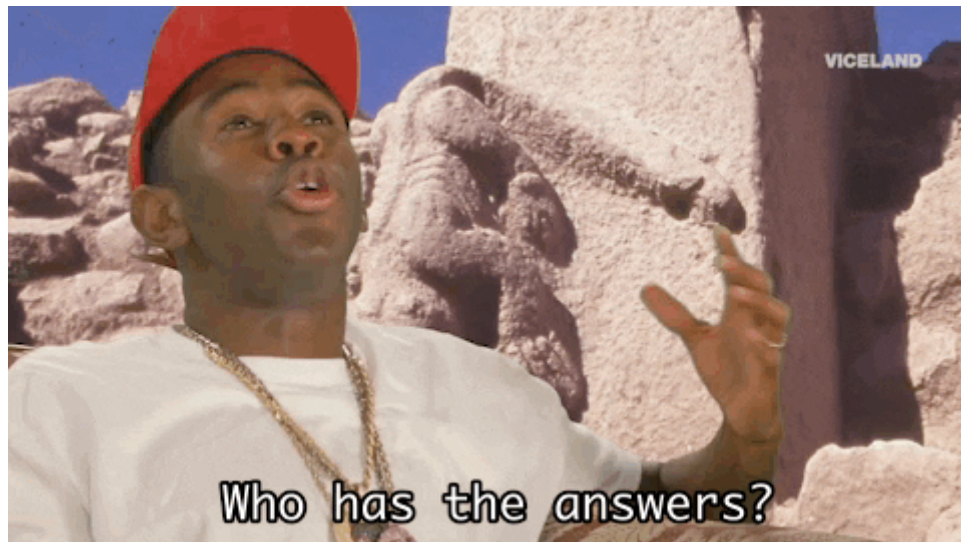
a vytváranie

Inak (2 z 3b.):

objekt ktorý tvorí kostru aplikácie sú manažované IOC kontajnerom - beans.

Komentár:

a vytvára beany



Stručne (max. 1-2 vetami) vysvetlite, čím sa líšia a čo majú spoločné návrhové vzory Proxy a Decorator.

Odpoved za 1,5 z 2b.:

Oba vzory obalujú instanciu existujúceho rozhrania (vnútornej instancie), ktoré implementuje to isté rozhranie a deleguje volania svojich funkcií na rovnaké funkcie vo svojej vnútornej instancii.

Komentár:

ale v com su rozne

Stručne (max. 1-2 vetami) vysvetlite, čím sa líšia a čo majú spoločné návrhové vzory Composite a Decorator.

¬_(ツ)_/

S akým návrhovým vzorom súvisí IoC kotainer frameworku Spring? Stručne vysvetlite.

strategy, template method

Odpoved za 1 z 2b. :

Pri IoC chceme dosiahnuť to, aby sme nemuseli pri rozširovaní aplikácie zasahovať do existujúceho kódu.

Návrhové vzory:

- Template Method - časť algoritmu je definovaná v nadtriede, podtriedy si vedia doimplementovať chýbajúce časti bez zmeny pôvodného kódu.
- Strategy - je definovaná množina algoritmov, ktoré sa dajú navzájom zamieňať - vieme zmeniť algoritmus bez toho aby bol klient zasiahnutý.

Komentár:

ano, ale hlavne IoC kontainer je Composit a vytvara ho builder

// Exercise 11

Predpokladajte, že kolekciu JavaPairRDD pdd sme vytvorili načítaním textových súborov funkciou wholeTextFiles S využitím operácií RDD-api napíšte výraz, ktorého výstupom je mapa (java.util.map) udávajúca počet rôznych slov v každom súbore (t.j. kľúč je meno súboru a hodnota počet)

╰_(ツ)_╯

Skontrolovať ešte

```
JavaPairRDD<String, String> rdd = sc.wholeTextFiles(path: "src/main/resources");
JavaPairRDD<String, String> rwdd = rdd.flatMapValues(s -> Arrays.asList(s.split(regex: "\\s+")));
System.out.println("\nmemo suboru : pocet roznych slov");
rwdd.distinct().countByKey().foreach((k, v) -> System.out.println(k + ":" + v));
```

Vysvetlite, čo je premiešanie (shuffle) a kedy nastáva, uveďte príklad spark funkcie, pri ktorej môže nastať.

Certain operations within Spark trigger an event known as the shuffle. The shuffle is Spark's mechanism for re-distributing data so that it's grouped differently across partitions. This typically involves copying data across executors and machines, making the shuffle a complex and costly operation. repartition, sortByKey, join// **Exercise 13**

Predpokladajte, že JavaPairRDD pdd je kolekcia dvojíc, kde prvá zložka je meno študenta druhá názov predmetu, ktorý má zapísaný. S využitím operácií RDD-api napíšte výraz, ktorého návratovou hodnotou je mapa (java.util.map) udávajúca pre každý predmet, koľko študentov ho má zapísaný (t.j. kľúč je názov predmetu a hodnota počet)

~_(_)_/_

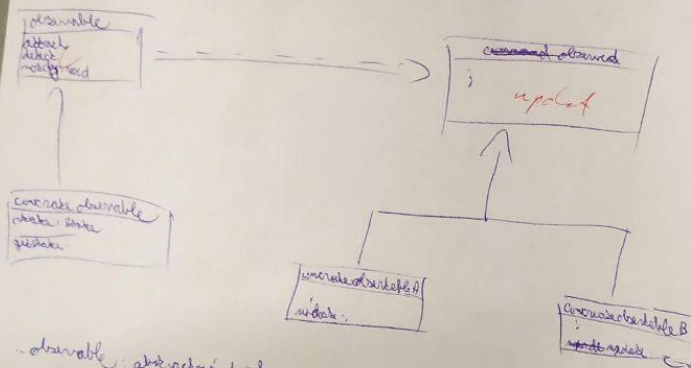
Charakterizujte návrhový vzor Observer

- **Nakreslite UML class-diagram a sequence diagram**
- **Vymenujte triedy/objekty tvoriace návrhový vzor a charakterizujte stručne ich úlohu v nom**

3. Charakterizujte návrhový vzor: **Observer**

- Nakreslite UML class-diagram a príp. sequence diagram
- Vymenujte triedy/objekty tvoriace návrhový vzor a charakterizujte stručne ich úlohu v ňom

[10]



- observable: abstraktná trieda
- concreteObservable: udržiava stav objektu
- observer: slúži na prácu s funkcionalitami
- concreteObservable A: konkrétna implementácia observed
- concreteObservable B: konkrétna implementácia observed

Skusim to prelozit

Observable - abstraktna trieda

Concrete observable - udrziava stav objektu

Observed - sluzi na pracu s funkcionalitami

concrateObservable A - konkretna implementacia observed

concrateObservable B - konkretna implementacia observed

Dalšie kody z ktorých môže byť otázka

```
// wholeTextFiles collectAsMap
System.out.println("\nmno suboru : cely obsah");
JavaPairRDD<String, String> pdd = sc.wholeTextFiles(path: "src/main/resources");
pdd.collectAsMap().foreach((k, v) -> System.out.println(k + ":" + v));

// mapValues
System.out.println("\nmno suboru : velkost suboru");
JavaPairRDD<String, Integer> pdd2 = pdd.mapValues(s->s.length());
pdd2.collectAsMap().foreach((k, v) -> System.out.println(k + ":" + v));

// flatMapValues
System.out.println("\nmno suboru : slova");
JavaPairRDD<String, String> fwdd = pdd.flatMapValues(s -> Arrays.asList(s.split(regex: "\\s+")).iterator());
List<Tuple2<String, String>> tl = fwdd.collect();
tl.foreach(t -> System.out.println(t._1 + ":" + t._2));

// reduceByKey
System.out.println("\nmno suboru : dlzka najkratsieho slova v subore");
fwdd.mapValues(s->s.length()).reduceByKey(Math::min).collectAsMap()
    .foreach((k, v) -> System.out.println(k + " : " + v));

// countByKey
System.out.println("\nmno suboru : pocet roznych slov");
fwdd.distinct().countByKey().foreach((k, v) -> System.out.println(k + ":" + v));
```