Ako funguje broker pri protokole AMQP?

- Ukladá správy od publishera do fronty aby ich mohol prečítať consumer

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?

- Grafová DB
- Dokumentová 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?

- Double Dispatch

Aký typ architektúry používa Docker?

- Klient-Server

Je mozne definovat fallback volanie pri pouziti Feign client-a?

- Áno

Kedy sa odporúča použiť microservice architektúra?

- **Pri vytváraní aplikácií, ktoré sa často menia a dopĺňajú**

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(toto je odpoveď pre kedy nie je možné použiť DI):

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?

- **Algolia**

Ktorá z možností je určená na Spring Messaging?

- **Spring Integration**

Ktorá z nasledujúcich databáz má grafovú štruktúru?

- **Neo4j**

Ktoré metódy nepatria vo frameworku GraphX do partition strategy?

- RandomEdgeCut
- EdgePartition3D

Ktoré sú tri základné entrypointy v GraphQL?

- Query, Mutation, Subscription

Ktorý protokol gRPC používa?

- HTTP 2.0

Ktorý z nasledujúcich servisov nevyžaduje XML konfiguráciu?

- Spring Boot

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

- Functional 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 situácií použiť?

- Facade

Na čo sa používa programovací jazyk R?

- analýza dát, štatistika
- čistenie dát a zobrazenie grafov

Na čo slúži Consul K/V Store ?

- Na držanie a poskytovanie konfiguračných properties pre aplikácie

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?

- Liskov Substitution principle

Označte pravdivé tvrdenia.

- Integráciou PyTorch s Apache Spark vieme dosiahnuť rýchlejšie trénovanie neurónovej siete s veľkými dátami.
- Integrácia Spark s Tensorflow / Pytorch sa používa na distribuované trénovanie.

Prečo je zakázaný cyklicky import závislostí v jazyku Golang?

- Aby bola kompilácia programu rýchlejšia

Pri ktorom z príkladov sa môžu antipatterny prejaviť v kóde?

- Bloky nepoužitého kódu
- Nedostatočná dokumentácia

V akých 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.*

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

- Prepája komponenty, spracováva požiadavky používateľov a odovzdáva ich do view na vykreslenie.

Vyber pojmy tykajuce sa search engines

- index
- dokument

Vyberte pravdivé tvrdenia o Spark ML?

- Spark ML je knižnica kompatibilná s viacerými jazykmi umožňujúca riešiť problémy 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.
- Mediator 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"?

- Client

Čo nepatrí do štruktúry "Iterator"?

- Product

Čo nepatrí k dokumentovým DB?

- Redis

Čo nepodporuje Cassandra Query Language?

- **Cudzie kľúče**

Čo sa považuje za výhody pre Spring Data?

- **Automatické generovanie queries**
- **Repository pattern**

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

- **Kanály by sa mali uzatvárať zo zapisovateľa a to len vtedy ak existuje len 1 zapisovateľ**

Čo znamená skratka gRPC?

- **gRPC Remote Procedure Calls**

Čo znamená skratka SAML?

- **Security Assertion Markup Language**

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

správne	nesprávne
• **coalesce** • **Cogroup** • **groupBy** • **join** • **repartition** • **sortByKey** • **Všetky *ByKey (groupByKey, reduceByKey…) - okrem countByKey**	• Filter • flatMap • flatMapValues • foreach • map • Sample • Union

Ktoré z nasledujúcich operácii spark-api sú transformácie?

správne	nesprávne
• **distinct** • **Filter** • **flatMap** • **flatMapValues** • **groupBy** • **intersection** • **map** • **mapToPair** • **reduceByKey** • **sample** • **sortByKey** • **union**	• 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
• **aggregate** • **Collect** • **count** • **countByValue** • **first** • **fold** • **foreach** • **reduce** • **take(n)** • **top**	• 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 vratit len true alebo false, filter nevie pracovat s 2 a viac parametrami (x,y)*)

správne	nesprávne
• **S -> “hello”.startWith(s)** • **a -> a - 3 > 0** • **s->s.isEmpty()** • **A -> true**	• (a,b) -> a && b • s->s.size() • A -> 1.0 • (s, t) -> s.equals(t)

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 ma property urok,
- x je lokálna premenná, y globálna premenná

správne	nesprávne
• s -> new Tuple2(s, 1) • s -> new Tuple2(a, a*a) • s -> s.isEmpty() • s -> "hello".startWith(s) • a -> { int x=a>0?1:0; return x*a; } • (a,b) -> a + b • a -> Integer.MIN_VALUE + a • s -> { if ("hello".startWith(s)) return true; return false; } • a -> { int x=a>0?1:0; return x*a; }	• u -> { Urok x=new Urok(u); x.setUrok(0.1); return x; } • s -> {system.out.println(s); return s.length();} • a -> a * Math.random() • a -> {return a * y;} • a -> a * y • a -> { return Math.random() *a;} • u -> { double x = u.getUrok(); system.out.println("urok="+x); } • u -> {u.setUrok(0.1);} • s -> system.out.println(s) • A -> {Date d = new Date(); return s+d;}

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
• **(a, b) -> Math.min(a, b)** • **(a, b) -> a && b** • **(a, b) -> a + b** • **(a,b) -> a * b** • **(a, b) -> a || b** • **(a, b) -> Math.max(a, b)**	• (a,b) -> a-b • (a,b) -> a • (s, t) -> s.split(t) • (s,t)->s.equals(t) • (a,b) -> a % b • (a,b) -> (a+b)/2

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
• **A -> new Tuple2(a, a*a)** • **A -> true** • **s -> s.isEmpty()** • **s -> { return s.length(); }** • **a -> 1.0** • **s -> s.substring(1, 3)** • **a -> a > 0** • **a -> {int y=a>0?1:0; return a*y;}**	• a -> { int y = a > 0 ? 1 : 0; a * y; } • a -> { a++;} • s -> {s.length() + 2;}

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?

```
static int LIMIT;

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

- **3**

Čo vypíše nasledujúci program?

```
static int LIMIT;


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

- **2**

Čo vypíše nasledujúci program?

```
static  int ZLAVA;

public static void main(String[] args) {

    SparkConf conf = new SparkConf();

    JavaSparkContext sc = new JavaSparkContext(conf);

    ZLAVA = 25;

    List<Integer> dl = Arrays.asList(100, 110, 120, 130, 140, 150);

     JavaRDD<Integer> rdd1 = sc.parallelize(dl);

    JavaRDD<Integer> rdd2 = rdd1.map(x -> x - ZLAVA);

    ZLAVA = 5;

    JavaRDD<Integer> rdd3 = rdd2.map(x -> x - ZLAVA).map(x -> x -
ZLAVA);

    System.out.println("" + rdd3.filter(x -> x < 100).count());

 }
```

- **2**

Čo vypíše nasledujúci program?

```
public static void main(String[] args) {

    SparkConf conf = new SparkConf();

    JavaSparkContext sc = new JavaSparkContext(conf);

    int LIMIT = 20;

    List<Integer> dl = Arrays.asList(1, 10, 100, 1000);

    JavaRDD<Integer> rdd1 = sc.parallelize(dl);

    JavaRDD<Integer> rdd2 = rdd1.filter(x -> x < LIMIT);

    LIMIT = 200;

    System.out.println("" + rdd2.count());

}
```

- **build error**

Čo vypíše nasledujúci program?

```
public static void main(String[] args) {

    SparkConf conf = new SparkConf();

    JavaSparkContext sc = new JavaSparkContext(conf);

    int ZLAVA = 25;

    List<Integer> dl = Arrays.asList(100, 110, 120, 130, 140, 150);

    JavaRDD<Integer> rdd1 = sc.parallelize(dl);

    JavaRDD<Integer> rdd2 = rdd1.map(x -> x - ZLAVA);

    ZLAVA = 5;

    JavaRDD<Integer> rdd3 = rdd2.map(x -> x - ZLAVA).map(x -> x -
ZLAVA);

    System.out.println("" + rdd3.filter(x -> x < 100).count());

}
```

- **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**
- **Prispôsobiť 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 kontainerom.
- 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 funkcionalitu 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 funkcionalitu 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 zdielaný 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 detache()?

- **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

Priraďte, 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
Facade	> Structural
Factory Method	> Creational
Flyweight	> Structural
Interpreter	> Behavioral
Iterator	> Behavioral
Mediator	> Behavioral
Memento	> Behavioral
Observer	> Behavioral
Prototype	> Creational
Proxy	> Structural
Singleton	> Creational
State	> Behavioral
Strategy	> Behavioral
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 pridavať 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 navrhnute. Bridge umoznuje, aby veci fungovali predtym. Bridge je navrhnuty dupredu, aby sa abstrakcia a implementacia mohli mohli menit nezavisle. Adapter je zase navrhnuty tak, aby nesuvisiace triedy spolupracovali.

Iná odpoveď:

Bridge: oddeluje abstrakciu od implementacie, aby sa mohli lisit a riesit nezavisle na sebe, toto necha kod klienta nezmaneny
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 vykonavaju ziadosti bez toho, aby nieco vedeli o pozadovanej operacii alebo o prijimatelovi ziadosti

> **Comments:**
> neuplne
> CoR je casto vykonavatelom operacie 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 te 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 pouziva pre vsetky objekty rovnaky interface (wrapper). Adapter konvertuje existujuci interface na odlisny, aby docielil kompatibilitu s inym rozhranim**

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 spravne? V ASOS_RT_21.pdf to je za 3b, neni to kratke moc? **Potrebujeme ešte vysvetlenie**

Command pattern.

Vymenujte základné pojmy AOP

- **ASPECT - je modul API volani, ktory poskytuje danu funkcionalitu**
- **ADVICE (before, after, after-returning, after-throwing, around) - akcia vykonana aspektom na urcity joinpoint**
- **JOIN POINT - je bod vykovania programuu ako je napr. vykonanie metody alebo spracovanie vynimky.**
- **POINTCUT (najbežnejšie execution pointcut) - je to predikat alebo vyraz, ktory sa zhoduje s jointpoint**
- **INTRODUCTION**
- **TARGET OBJECT**
- **AOP PROXY**
- **WEAVING**

Rozsiahlejsia odpoved 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 potrebnymim 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 funkcionalitu potrebuje iný object)

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

Uveďte rôzne typy proxy objektov

remote, virtual, protection

Uveďte 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>

Odppoved za 2,5 z 3b:

Pouziva sa na definovanie jednej alebo viacerych operacii, tie su definovane ako vstupno / vystupne vzory - isteho 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 definiciu typov sa pouziva XML-schema jazyk.

Predpokladajte, že sme do kolekcie JavaRDD<String> rdd načítali riadky textového súboru. S využitím operacií 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 operacií RDD-api napíšte výraz pre výpočet symetrickej diferencie množín reťazcov t.j. celkového počtu rôznych reťazcov, ktoré sa nachadzajú 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 VEĽKOSŤOU PÍSMEN považujte za totožné.

```
rd1 = rd1.map(s -> s.toUpperCase());
rd2 = rd2.map(s -> s.toUpperCase());

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

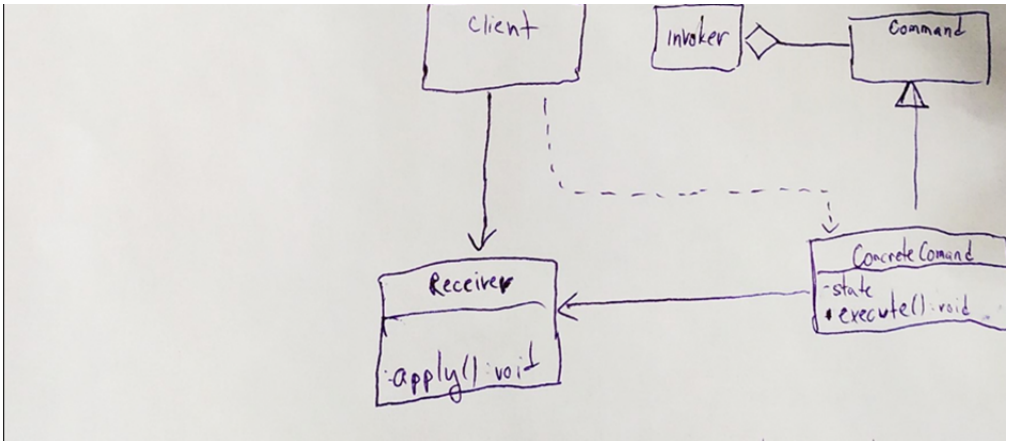
Predpokladajte, že JavaRDD<String> rdd je kolekcia reťazcov. S využitím operacií RDD-api napíšte výraz, ktorého výstupom je boolovská hodnota hovoriaca či sú v kolekcii duplicitné reťazce, t.j. ak sú má výraz hodnotu true inak false.

```
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 - metodou execute vykonava request**
- **Command - interface, ktory cez ConcreteCommand plni request**
- **Invoker - vola jednotlive requesty**

Predpokladajte, že JavaRDD<String> rdd je kolekcia slov. S využitím operacií 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 líšiace 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 operacií 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())
        ).distinct().collect();
```

Predpokladajte, že sme do kolekcie JavaRDD<String> rdd načítali riadky textového súboru. S využitím operacií 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 operacií 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())).distinct().collect()`

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

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 operácii 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 prázdnymi 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 operacií RDD-api napíšte výraz, ktorého výstupom je boolovská hodnota hovoriaca či sú v kolekcii duplicity (t.j. ak sa v kolekcii vyskytuje reťazec viac krát výraz hodnotu true inak false). Reťazce líšiace sa len veľkosťou písmen považujte pri tom za rovnaké.

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

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

```
pdd.flatMapValues(x->Arrays.asList(x.split("\\s")).iterator()).distinct().countByKey();
```

Predpokladajte, že sme do kolekcie JavaRDD<String> rdd načítali riadky textového súboru. S využitím operacií RDD-api napíšte výraz, vráti počet rôznych slov v súbore, pričom slová líšiace 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<String, List<String>> pdd je kolekcia dvojíc, kde prvá zložka je meno študenta druhá názvov predmetu, ktorý má zapísaný. S využitím operacií 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)));
```

Predpokladajte, že JavaPairRDD<String, String> pdd je kolekcia dvojíc, kde prvá zložka je meno študenta druhá názvov predmetu, ktorý má zapísaný. S využitím operacií 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.flatMapValues(x->Arrays.asList(x.split("
    ")).iterator()).groupByKey().collectAsMap().forEach((k, v) ->
    System.out.println(k + " " + String.join(",", v)));
```

Predpokladajte, že sme do kolekcie JavaRDD<String> rdd načítali riadky textového súboru. S využitím operacií RDD-api napíšte výraz, vráti počet výskytov slova ASOS v súbore, pričom nezáleží na veľkosti písmen (Pozn. riadky treba rozdeliť na slová)

```
rdd.flatMap(s -> Arrays.asList(s.split(" ")).iterator()).map(s
-> s.toLowerCase()).filter(s -> s.equals("asos")).count()
```

Predpokladajte, že sme do kolekcie JavaRDD rdd načítali riadky textového súboru. S využitím operacií RDD-api napíšte výraz, vráti dĺžku najdhšieho slova v súbore (Pozn. riadky treba rozdeliť na slová)

```
rdd.flatMap(s -> Arrays.asList(s.split("
")).iterator()).map(String::length).reduce(Math::max)
```

Spark rozdeluje funkcie na pracu s distribuovanymi datami na akcie a transformacie. Vysvetlite v com je hlavny rozdiel medzi nimi.

Transf- transformuju RDD objekt na iny RDD. Vstup a vystup je na workeroch.

Akcie- zobrazuju upravene vystupy, vystup 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 kontainerom. - composite
Visitor riesi poskytnutie operacie pracujucej s objektami roznych typov a composite poskytuje jednotne rozhranie pre pracu so samotnymi 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 napriklad aj Facade sa zvykne implementovat ako singleton lebo jeden facade objekt je vo vacsine pripadov postacujuci.

Vysvetlite stručne úlohu IoC kontajnera.

-moznost na upravu a spravu objektov pomocou reflection
-zodpoveda za iniciaciu,configuraciu a zostatovanie beanov
-je zodpovedný za správu životného cyklu objektov

IoC kontajner je velmi výhodný pri developovaní applikácii/web app. ,pretože sa v nom lahko "testuju/vymienaju" veci. Tzn, že kludne môžme jeden kontajner zmeniť za druhý z uplne iným vyznamom a kludne sa môžme k nemu naspäť vrátiť, ak by bol nový zlý atd.. Tak isto, vo vnutri jedneho kontajnera, je práca ulahčená, pri výmenách/testoch kodu

IoC kontajner je velmi výhodný pri developovaní applikácii/web app. ,pretože sa v nom lahko "testuju/vymienaju" veci. Tzn,
že kludne môžme jeden kontajner zmeniť za druhý z uplne iným vyznamom a kludne sa môžme k nemu naspäť vrátiť, ak by bol
nový zlý atd.. Tak isto, vo vnutri jedneho kontajnera, je práca ulahčená, pri výmenách/testoch kodu

Komentár:

ale konkretene> sluzi na vytvaranie inicializaciu a manazovanie komponent tvoriacich aplikaciu

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 menežovanie životného cyklu objektov (beanov) a závislostí objektov pomocou dependency injection.
Komentár:
a vytvaranie

Inak (2 z 3b.):
objekt ktory tvori kostru aplikacie su manazovane IOC kontajterom - beans.
Komentár:
a vytvara beany

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

Mechanizmus Sparku na opätovnú distribúciu údajov tak aby boli rôzne zoskupené medzi particiami - toto typicky zahŕňa kopírovanie dát medzi exekútormi alebo strojmi čím sa shuffle stáva komplexnou a náročnou operáciou. Nastáva keď nejaká operácia potrebuje prečítať údaje zo všetkých partícií na nájdenie všetkých hodnôt pre všetky kľúče a potom dať dokopy všetky tieto hodnoty na získanie finálneho výsledku pre každý kľúč. Príklady: reduceByKey, groupByKey, cogroup, join

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 obaluju instanciu existujuceho rozhrania (vnutornej instancie), ktore implementuje to iste rozhranie a deleguje
volania svojich funkcii na rovnake funkcie vo svojej vnutornej 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)_/¯(neoverene) Vzor Composite vytvára stromovú štruktúru objektov, kde objekt môže mať viacero potomkov a každý potomok môže byť objekt alebo kompozit. Vzor Decorator dynamicky pripája k objektu ďalšie povinnosti tým, že pôvodný objekt obalí objektom decorator, ktorý poskytuje ďalšiu funkcionalitu.

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
doiplmementovať 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 13

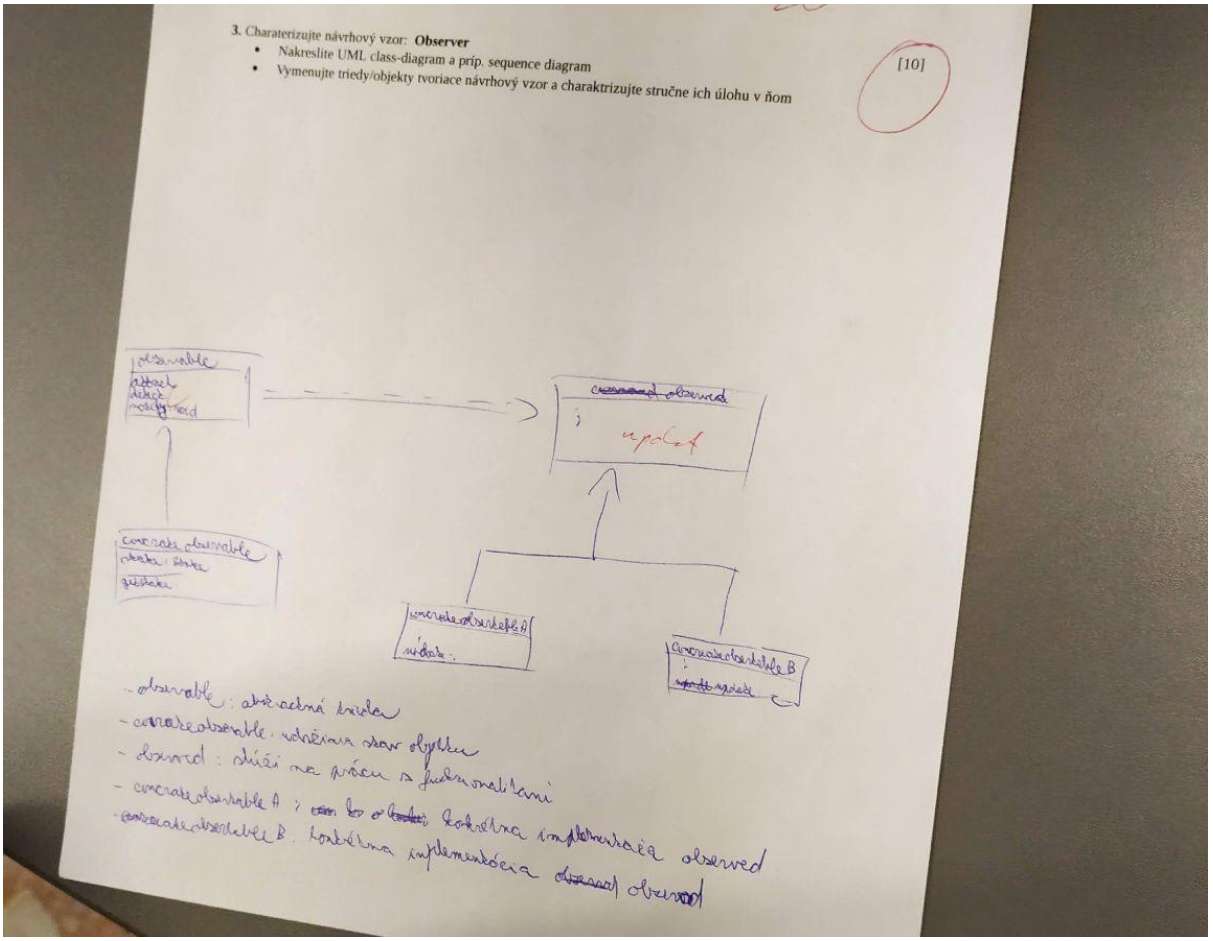
Predpokladajte, že JavaPairRDD pdd je kolekcia dvojíc, kde prvá zložka je meno študenta druhá názvov predmetu, ktorý má zapísaný. S využitím operacií RDD-api napíšte výraz, ktorého návratovou hodnotou je mapa (java.util.map) udávajúca pre každý predmet, koľko študentov ho má zapísaný (t.j. kľúč je názov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

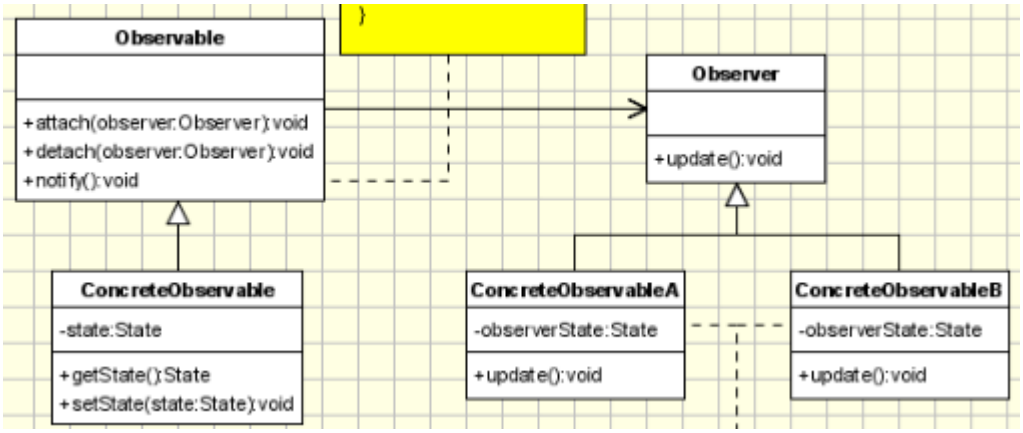
(neoverene)

```
pdd.flatMapValues(s -> Arrays.asList(s.split(",")))
        .mapToPair(pair -> new Tuple2<>(pair._2, 1))
        .reduceByKey((a, b) -> a + b)
        .collectAsMap();
```

Charakterizujte návrhový vzor Observer

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





Observable - abstraktna trieda definujuca funkcie na pridanie alebo oddelenie Observers klientovi.

Concrete observable - udrziava stav objektu a ak nastane zmena, upozorni prislusnych Observers

Observer - sluzi na pracu s funkcionalitami

concrateObservable A - konkretna implementacia observeru

concrateObservable B - konkretna implementacia observeru

Dalšie kody z ktorych môže byť otazka

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