**Programozás Módszertana II.**

**Előadás**

1. **Előadás**

* Mi az a Kliens-Szerver architektúra?
  + Másnéven Ügyfél-Kiszolgáló architektúra
  + Régen a hálózatban működő számítógépet jelentette, 80-as évektől használjuk a mai jelentése szerint
  + Kliensekből és szerverekből áll, amik hálózaton, vagy ugyanazon számítógépen kommunikálnak
  + Kliens kapcsolatot kezdeményez, szerver várakozik a kliens kéréseire
  + Lehetőség a terhelés elosztására
* Mi az a P2P architektúra?
  + Peer-to-peer
  + Minden egyes kapcsolódó állomás lehet kiszolgáló és kliens is
  + Mindegyik résztvevő azonos felelőséggel és státusszal rendelkezik
* Mit jelent az, hogy URI és URL?
  + Internetes oldalak a következőkből állnak: HTML, css, JS, képek, videók, hangok stb.
  + Az egyes elemekre hivatkozni kell, URL-el
  + Uniform Resource Identifier, Uniform Resource Locator (egységes erőforrás helymeghatározó), a különböző erőforrások elérésére
* Mit jelent a HTTP kifejezés?
  + webes objektumokat HTTP protokoll segítségével lehet elérni, oldal megjelenítésére pedig webböngésző, azaz kiszolgáló kell
  + Hypertext Transfer Protocol (hiperszöveg átviteli protokoll)
  + oldal megjelenítését a kliens kezdeményezi, megadjuk a megfelelő URL-t, az egy HTTP/HTTPS kérésen keresztül eljut a szerverhez, ami feldoglozza azt és visszaküldi az adatokat a kliensnek
  + kérés-válasz alapú protokoll kliens és szerver között
* Mi az a DMZ?
  + Demilitarizált zóna
  + fizikai, vagy logikai alhálózat
  + plusz biztonsági réteg a LAN számára
  + a rendszer érzékeny részei nem érintkeznek a felhasználóval, DMZ-n keresztül megy
  + DMZ hosztjai csak jól meghatározott szabályok szerint kommunikálhatnak
* Mik a kliens-szerver architektúra előnyei a P2P architektúrával szemben?
  + szerepek és felelősségek szétosztása
  + karbantarthatóság, lehetőség a kiszolgáló cseréjére
  + biztonság: adatok a szerveren tárolódnak
  + adatok frissítése központilag könnyen megoldható
  + számos ügyfélkiszolgáló technológia áll rendelkezésre
* Mik a P2P architektúra előnyei a kliens-szerver architektúrával szemben?
  + csomópontok hozzáadásával nő a sávszélesség, jobb terheléselosztás
  + Hibatűrő, ha egy gép meghibásodik nem áll le a rendszer

1. **Előadás**

* Mi az az Interfész?
  + Érintkezési, kapcsolódási felület
  + számítógépes eszközök (hardver, szoftver), avagy a számítógép és a felhasználó közötti felület
  + Olyan kommunikációs megoldást feltételez, amit minden résztvevő megért (közös nyelv)
  + Típusai:
    - Humán/humanoid
    - HCI (Human Computer Interface) pl perifériák
    - Technikai (hardver-szoftver)
    - Egyéb környezethez illeszkedő-
* Mi az az API?
  + Application Programming Interface
  + Kettő vagy több számítógépes program egymás közötti kommunikációját megvalósító program
  + Egy típusa a szoftveres interfészeknek
* Mit jelent a REST API kifejezés és a wadl?
  + Representational State Transfer / reprezentáción alapuló állapotátvitel
  + tervezési minta
  + Reprezentáció itt erőforrás reprezentációra utal! XML és JSON reprezentációk
  + Ez egy szoftverarchitektúra minta internetes kiszolgáló rendszerekhez
  + protokolloktól független, de legtöbbször HTTP-t használ
  + Megszorítások:
    - kliens-szerver architektúrára alapul
    - állapotmentesség (minden kérés tartalmazza az összes szükséges információt a kiszolgáláshoz)
    - egységes interfész (erőforrások egyértelműen azonosíthatóak URI segítségével, GET, PUT, POST, DELETE)
    - gyorsítótárazhatóság
    - réteges felépítés
  + WADL: Web Application Description Language
    - gépek által értelmezhető xml leírása a HTTP alapú webszolgáltatásoknak
* Mit jelent a SOAP kifejezés és a wsdl?
  + Simple Object Access Protocol
  + egyszerű objektumhozzáférési protokol
  + struktúrált adatok cseréjére szolgáló protokol
  + XML-re alapul
  + WSDL: Web Service Description Language

1. **Előadás**

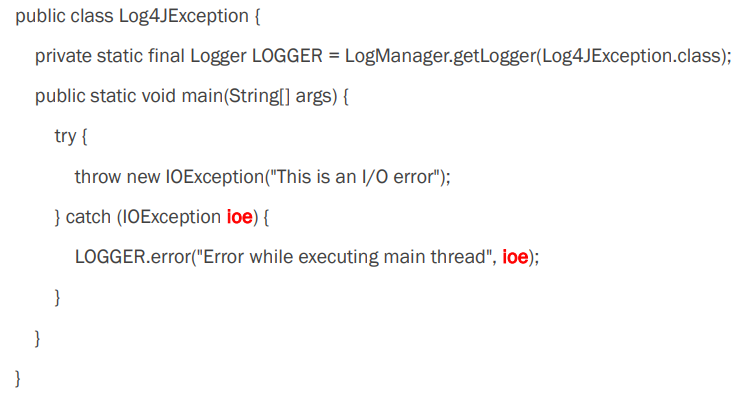
* Mi az az Exception?
  + Magyarul kivétel
  + Egy esemény, ami a program végrehajtása során következik be és megzavarja annak normál, folyamatos futását
  + Kivétel bekövetkeztekor a metódus egy exception object-et hoz létre, és átadja azt a futtató környezetnek
  + Ez információt tartalmaz a kivétel típusáról és a program állapotáról a bekövetkezéskor
* Milyen módszerrel lehet a Java-ban kivételt dobni?
  + Kivétel osztályt létrehozni és átadni azt a futtató környezetnek
  + Throw után: A futtató környezet megpróbálja megtalálni mi kezeli ezt a kivételt a „call stack” segítségével
    - call stack tartalmazza, hogy milyen függvényeket hívtunk meg, mire idejutottunk
  + futtató rendszer megkeresi azt a kódblokkot ami kezeli az adott típusú kivételt
  + amikor megtalálja a kezelőt akkor átadja neki a kivétel objektumot paraméterként
  + throw kulcsszóval saját vagy létező kivételt dobhatunk
* Milyen módon lehet a Java-ban elkapni a kivételeket?
  + Megfelelő kivételkezelő kiválasztása (megfelelő, ha kezeli az adott kivételt)
  + Ha nem talál megfelelő kezelőt a futtató környezet, akkor leáll a futtatási folyamat
  + kivétel létrejön -> futtató környezet megkapja az objektumot -> call stackben megkeresi hogy ki kezeli -> átadja a megfelelő kivételkezelőnek
  + try-catch block
* Mit jelent a Java-ban a „Catch or Specify” követelmény?
  + Kötelező vagy elkapni a kivételt, vagy abban specifikálni
  + tehát kell egy catch ág VAGY specifikálnia kell a metódusnak, hogy eldobhatja a kivételt
* Mi a kivételek három fő csoportja Java-ban?
  + Checked exception – ellenőrzött kivétel, amire előre lehet számítani, hogy be fog következni a program lefutása során. Bekövetkezés és kezelés után fut tovább a program
  + Error – Hiba, amit külső tényező okoz, nem a program lefutása
  + Runtime exception – Futás idejű kivétel a programban, általában bug vagy logikai hiba okozza
* Mit jelent a Try-with-Resources kifejezés? Mire használjuk?
  + Biztosítja, hogy az erőforrások le lesznek zárva a try-catch blokk végén
  + Minden objektummal használható, ami a java.lang.AutoCloseable osztályt implementálja
  + pl fájlolvasó

1. **Előadás**

* Mik a kezelt és nem kezelt kivételek Java-ban?
  + Checked / kezelt: olyan kivételek, amiket a compiler maga is ellenőriz. Amikor eldobod, akkor vagy helyben kell kezelned, vagy meg kell jelölnöd a metódus throws deklarációjában, különben nem fut le a kód. Object->Throwable->Exception-ből örökítettek
  + Unchecked / nem kezelt: fordítási időben nem kerülnek ellenőrzésre, try-catch ill. throws nélkül is „lefut” velük a kódunk. A Runtime Exception-ből származtatva tudunk ilyeneket magunktól írni
* Soroljon fel néhány (min 3.) naplózással kapcsolatos elvet és röviden fejtse ki ezeket!
  + naplózzunk minél többet, de ne túl sokat
    - derüljön ki, hogy működik a program pl. megfelelő függvények meghívásra kerülnek
    - túl sok adat, pl futó sql scripteket ne logoljuk már, mert túl sok lesz
  + használjunk standard naplózó eljárást
    - standard naplózó keretrendszerek: SLF4J, java.util.logging, stb
  + használjunk kifejező üzeneteket
    - standard naplózó keretrendszerek, konkrét üzenetek
  + naplózzuk a kivételeket és a hibákat
    - minden esetben meg kell jeleníteni! ha kezeli és lefut ha nem!!
  + naplózzuk a stack trace-t
  + naplózzunk fájlba
    - utólag visszaolvasható, pl xml, json, text stb
  + használjunk megfelelő naplózási szintet
  + ne naplózzunk szenzitív adatokat
    - személyes adatok, jelszavak, egyéb elérés adatbázisokhoz stb
    - üzemeltető is megnézheti, ezért nem szabad logolni
  + ha megtehetjük, használjunk napló management eszközt és központilag ellenőrizzük
* Mutasson be egy hibakezelést naplózással Java nyelven!

Graphical user interface, text, application

Description automatically generated



1. **Előadás**

* Mi az az Aspektus Orientált Programozás?
  + olyan programozási paradigma, amely növeli a modularitást, azáltal, hogy lehetővé teszi a támogató funkciók elkülönítését a főprogram üzleti logikától
  + Aspektus – szemlélet, nézőpont
  + AspectJ-ben az aspektus egy olyan osztály, ami egy másik meglévő osztály tulajdonságait egészíti ki
  + kiemelhetővé tesz programrészeket, pl logolás amik sok helyen lefutnak másképp
  + program horizontális bővítése
* Mit nevezünk Join-Pointnak az Aspektus Orientált programozásban?
  + Belépési pont
  + AOP központi eleme, kisebb megszorításokkal a program bármely azonosítható futási pontja lehet, pl metódus hívása, változónak értékadás stb.
  + ez egy hely, ahol AOP-vel tudsz kódot futtatni
* Mit nevezünk Pointcut-nak az Aspektus Orientált programozásban?
  + több joinpoint együttese
  + ezek valós osztályokban vannak
  + kb egy filter ami megszabja, hogy az adott feltétel mely join-pointoknál teljesüljön
* Mit nevezünk Advice-nak az Aspektus Orientált programozásban?
  + Meghatározza, hogy mi történjen az adott join-point(ok) esetében
  + before advice, after advice és around advice is létezik attól függően, hogy a joinpointhoz képest hol fusson le a kód
* Mit nevezünk inter-type declaration-nek az Aspektus Orientált programozásban?
  + Az osztályokba való statikus beavatkozás.
  + Új adattagokat és metódusokat vezethetünk be

1. **Előadás**

* Mit értünk interfész alatt a Java-ban?
  + A Java-ban az interfész alatt egy absztrakt típust értünk, amely konstansok és kívülről elérhető metódusok gyűjteménye
* Hogyan írjuk le az osztály definícióban Java-ban, hogy megvalósít egy interfészt?
  + public class [className] implements [implementation] {}
  + implements kulcsszó
  + minden metódust kötelező implementálni!
* Java nyelven hogyan oldjuk meg a többszörös öröklődést interfészek segítségével?
  + egy osztály nem tud több osztályból öröklődni, ellenben tud több interfészt implementálni
  + mindkét interfészt implementálja, így azok funkcióit megkapja
* Mi az a többalakúság, hogyan valósíthatjuk meg interfészek segítségével Java-ban?
  + Többalakúság / polimorfizmus: annak a lehetősége, hogy egy objektum több különböző alakot öltsön futásidőben (pl metódusoknak ugyanaz a neve, de argumentumok alapján eldönti, hogy melyik metódust hívjuk meg)
  + pontosabban: egy specifikus objektumtípushoz köthető felüldefiniált metódus végrehajtása
  + van egy interface ami tartalmaz egy konkrét metódust, azt implementáljuk több helyen (pl interface Shape, class circle) Shape circleShape = new Circle())
* Mit nevezünk absztrakt függvénynek és absztrakt osztálynak Java-ban?
  + Absztrakt osztályt használunk, ha az osztályban néhány függvényt implementálni szeretnénk, néhányat pedig csak a leszármaztatott osztályban szeretnénk implementálni
  + Nem példányosítható, de le kell származtatni és a leszármaztatottban implementálni a metódusokat
  + Akkor használjuk, ha néhány fő funkciót szeretnénk egységbe zárni (Kód újrahasznosítás miatt)
  + Absztrakt metódus az a metódus az absztrakt osztályban, amit nem implementáltunk

1. **Előadás**

* Mi az a programtervezési minta?
  + A tervezési minták egymással együttműködő objektumok és osztályok leírásai, amelyek testreszabott formában valamilyen általános tervezési problémát oldanak meg egy bizonyos összefüggésben.
* Milyen elemei vannak egy tervezési mintának?
  + Minta neve
  + A probléma, amire alkalmazni szokták a mintát
  + A megoldás – nem konkrét megvalósítás – elvont leírás az osztályok és objektumok általános elrendezése, ahogyan oldja meg a problémát
  + Következmények – az alkalazás előnyei és hátrányai
* Mi az az absztrakt gyár, vagy AbstractFactory? Mire használjuk?
  + Elvont gyár / abstract létrehozási minta
  + Abban nyújt segítséget, hogy objektumok egy csoportját anélkül hozhassuk létre, hogy az üzleti logikában törődnénk annak konkrét implementációjával
  + A gyakran változó, vagy még nem végleges típusú objektumainkat (nevezzük ezeket termékeknek) egy önálló osztályban, a Gyárban hozzuk létre.
  + Gyárnak van egy metódusa, ami visszaad egy új Terméket.
  + Az a logika, hogy pontosan hogyan, milyen erőforrások és egyéb osztályok használatával történik egy Termék létrehozása, a Gyár magánügye.
  + A kliens kód csak kap egy Gyárat és elkéri tőle a Terméket
* Soroljon fel három létrehozási mintát!
  + Abstract Factory
  + Építő programtervezési minta (Builder)
  + Gyártó metódus programtervezési minta (Factory method)

1. **Előadás**

* Mire szolgálnak a viselkedési tervezési minták?
  + A szoftverfejlesztésben a viselkedési tervezési minták olyan tervezési minták, amelyek azonosítják az objektumok közötti közös kommunikációs mintákat. Ezáltal ezek a minták növelik az objektumok közötti kommunikáció rugalmasságát.
  + A viselkedési tervezési minták az objektumok interakciójával és felelősségével kapcsolatosak. Ezekben a tervezési mintákban az objektumokt közötti interakciónak olyannak kell lennie, hogy könnyen beszélhessenek egymással és mégis lazán összekapcsolódjanak. Ez azt jelenti, hogy az implementációt és a klienst lazán kell összekapcsolni, a hardcode-olt függősék elkerülése érdekében
* Soroljon fel legalább három viselkedési mintát!
  + Chain of Responsbility Pattern
  + Command Pattern
  + Iterator Pattern
* Mi az a Chain of Responsibility minta, mondjon példát arra, hogy hol használják!
  + Felelősségi lánc minta
  + Olyan tervezési minta, amely parancs objektumok forrásából és feldolgozó objektumok sorozatából áll
  + A láncban minden feldolgozó objektum egy bizonyos parancsért felel, ha a feldolgozás megtörtént, akkor pedig továbbítja a parancsot a lánc következő feldolgozójának
  + Pl.: Spring keretrendszer – Security Filter Chain
* Mi az a Parancs (Command) viselkedési minta? Fejtse ki röviden!
  + Olyan viselkedési tervezési minta, amely a kérést önálló objektummá alakítja, amely tartalmazza a kéréssel kapcsolatos összes információt.
  + Ez lehetővé teszi a kéréseg függvény argumentumként való átadását, a kérés végrehajtásának késleltetését, vagy sorba állítását, valamint támogatja a végrehajtatlan műveleteket is