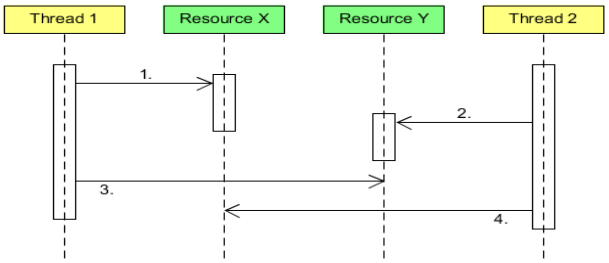
Szálkezelés

1. A szekvencia diagram feladata az objektumok egymás közti üzenetváltásainak ábrázolása egy időtengely mentén elhelyezve. Az objektumok életvonala egy felülről lefelé mutató időtengely. A nyilak az objektumok egymásnak szóló üzenetei.

Az alábbi ábrán a nyilak az erőforrások lock-olását jelölik. A kezdetektől tudjuk, hogy mindkét szálnak szüksége lesz mindkét erőforrásra.



Melyik az a hívás, amelynek megtörténtekor már biztosak lehetünk a deadlock kialakulásában?

2. Mi lesz az eredménye egy olyan objektum wait() metódusának hívásának, amelynek a hívó szál nem birtokolja a monitor lockját?

3. Mi a szálak alapértelmezett prioritása?

4. Melyik output nem lehetséges a következő kód futtatása esetén?

```
public class Tester extends Thread {
    int code = 9;
    public void run() {
        this.code = 7;
    }
    public static void main(String[] args) {
        Tester thread = new Tester();
        thread.start();
        for (int i = 0; i < 5; i++) {
            System.out.print(thread.code);
        }
    }
}
```

A – 97777 B – 77777 C – 79999 D – 99999

5. Mi lesz a következő kód futtatásának eredménye?

```
public class MyThread extends Thread {
    volatile private int x = 3;
    public static void main(String[] args) throws Exception {
        new MyThread().foo();
    }
    public MyThread() {
        x = 10;
        start();
    }
    public void foo() throws Exception {
        join();
        x = x - 1;
        System.out.println(x);
    }
    public void run() { x *= 2;  }
}
```

A – 8 B – 19 C – 18 D – 10

6. Mi lesz a következő kód futtatásának eredménye?

```
public class Test extends Thread {
    public void run() {
        System.out.println(isAlive());
    }
    public static void main(String[] args) {
        Test test = new Test();
        System.out.println(test.isAlive());
        test.run();
    }
}
```

A – false false B – false true C – true false D – true true

7. Melyek Atomi műveletek az alábbiak közül?

A. Referencia változók írása/olvasása
B. Minden primitív típusú változó írása/olvasása
C. Mindkettő
D. Egyik sem

8. Melyik állítás igaz az alábbi kódra?

```
public class Test extends Thread {
    private static int x;
    public synchronized void foo() {
        int current = x;
        current++;
        x = current;
    }
    public void run() {    foo();    }
}
```

A. A run metódus szinkronizálása szálbiztossá tenné az osztályt
B. Az osztály szálbiztos
C. A foo metódus statikussá tétele szálbiztossá tenné az oszályt
D. Futtatáskor exception – t eredményez

9. Melyik állítás nem igaz az immutable osztályokra?

A. Az osztályt final – ként kell deklarálni a leszármaztatás megelőzése végett
B. Az osztály összes adattagja private final – ként kell deklarálni
C. Az objektum típusú adattagokról mindig másolatot kell készíteni amikor azt getter metódusból ki kell adni és konstruktor paraméterből való inicializáció
D. Az osztály nem tartalmazhat olyan metódusokat, amelyek az állapotát a konstruktor lefutása után megváltoztatják.

Adatbáziskezelés / JDBC

1. Melyik nem DML utasítás az alábbi SQL utasítások közül?

A. update [table] set ...
B. delete from [table] ...
C. alter table ...
D. insert into [table] ...

2. Mely állítások igazak a JDBC kapcsolat felépítésével kapcsolatban?

1. `DriverManager` osztályon keresztül juthatunk pool-ozott kapcsolathoz.
2. `DataSource` osztályon keresztül juthatunk pool-ozott kapcsolathoz.
3. `DataSource` osztállyal történő kapcsolódás esetén meg kell adnunk az adatbázishoz tartozó connection URL-t.
4. Adatbázis kapcsolatot a `DriverManger.getConnection` hívással kaphatunk.

A – 1, 3 B – 2, 3 C – 2, 4 D – 3, 4

3. Mely információ nem nyerhető ki az SQLException objektumból?

A. SQL státusz kód.
B. Driver/Adatbázis specifikus hibakód.
C. A hibát okozó adatbázis kérés.
D. A felmerült hiba leírása.

4. Melyik helytelen módja a resultSet adatainak elérésének?

```
A. String value0 = rs.getString(0);
B. String value1 = rs.getString(1);
C. int    value2 = rs.getInt(2);
D. int    value3 = rs.getInt(“ADDR_LN1");
```

5. Melyik állítás igaz az alábbiak közül?

1. CallableStatement kiterjeszti a PreparedStatement interface-t. Ez az interface használható SQL tárolt eljárások hívására.
2. Statement kiterjeszti a PreparedStatement interface-t és akkor használatos, amikor az SQL lekérdezést nem szükséges többször futtatnunk.
3. PreparedStatement statikus lekérdezések indítására használatos (pl.: select * from table), ezért PreparedStatement-ek nem paraméterezhetőek.
4. PreparedStatement használatával lehetséges SQL utasítások batch feldolgozása.

6. Hogyan indítható új adatbázis tranzakció?

A. A `Connection`-höz egy `Transaction` object kérésével és azon `begin()` metódus hívással.
B. A `Connection`-höz egy `Transaction` object kérésével és annak `autoCommit` tulajdonságánal false-ra állításával.
C. A `Connection` `beginTransaction` metódusának hívásával.
D. A `Connection` `autoCommit` tulajdonságának false-ra állításával és egy SQL utasítás végrehatásával.

7. Mire való a tranzakció az adatbázisoknál?

A. Tárolt eljárások futtatására
B. Több művelet atomikénti végrehajtására
C. Kapcsolt táblás lekérdezésre
D. Átutalások elnevezésére

8. Melyik kapcsolatot szokás kapcsolótáblával leképezni?

A. 1-n kapcsolatot
B. m-1 kapcsolatot
C. m-n kapcsolatot
D. kapcsolatot

9. Mi nem része az Egyed-Kapcsolat diagramnak?

A. Attribútum
B. Entitás
C. Kulcsok
D. Osztály

10. Melyik nem igaz a JTable és a modell kapcsolatáról?

A. A JTable nem tartalmaz adatot
B. A modell felüldefiniálható
C. A modell nem tudja értesíteni a JTable-t a változásról
D. A modell reprezentációja eltérhet a JTable által lekérdezett adatokétól

Szoftvertechnológia / UML

1. Melyek a szoftverek alapvető minőségi mutatói?

A. Szállítási idő, megvalósítási költség, hardver- és szoftverigény.
B. Karbantarthatóság, megbízhatóság, biztonság, hatékonyság, használhatóság.
C. Módosíthatóság, bővíthetőség, felbonthatóság, újrahasználhatóság, megbízhatóság.
D. Ergonómia, használhatóság, kompatibilitás, hardver- és szoftverigény.

2. Melyik a használati történet (user story) szerkezete?

A. USER felhasználó IN USE CASE használati eset WITH RELATION kapcsolat
B. AS A szerepkör USE funkció TO cél
C. WHEN tevékenység APPLYING funkció IN ORDER TO cél
D. GIVEN környezet WHEN tevékenység THEN hatás

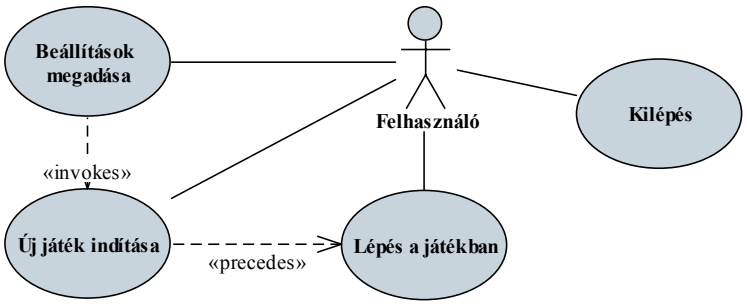
3. Mi a helyes sorrendje a követelményelemzésnek?

A. megvalósíthatósági elemzés, követelmény feltárás, követelmény specifikáció, követelmény validáció
B. követelmény feltárás, követelmény specifikáció, követelmény validáció, megvalósíthatósági elemzés
C. követelmény feltárás, követelmény validáció, követelmény specifikáció, megvalósíthatósági elemzés
D. követelmény feltárás, követelmény specifikáció, megvalósíthatósági elemzés, követelmény validáció

4. Az alábbiak közül melyik nem szoftverfejlesztési modell?

A. waterfall
B. evolution
C. scrum
D. prototyping

5. Milyen funkcionalitás olvasható ki az alábbi használati esetek diagramból?



A. A felhasználónak lehetősége van új játékot kezdeni, de csak miután a beállításokat megadta.
B. A felhasználó a beállítások megadásával automatikusan új játékot indít.
C. A felhasználónak külön nem szükséges beállításokat megadni, vagy új játékot indítania, azonnal léphet a játékban.
D. A felhasználó csak akkor léphet ki a programból, ha elkezdett egy játékot.

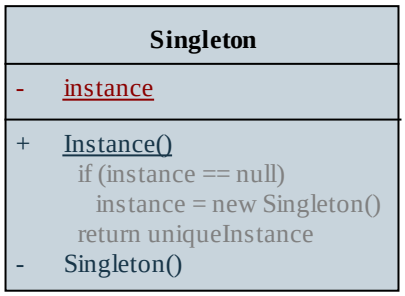
6. Az alábbiak közül melyek az UML használati eset (use case) diagram relációi?

A. függőség (dependency), kompozíció (composition), használat (usage), beágyazás (nesting)
B. előfeltétel (precedes), tartalmazás (include), használat (usage), általánosítás (generalization)
C. használat (usage), beágyazás (nesting), importálás (import), függőség (dependency)
D. felület (interface), megvalósítás (implementation), tartalmazás (include)

7. Mire szolgál az UML telepítési (deployment) diagram?

A. Ábrázolja azt a műveletsorozatot, amely a szoftver adott gépen történő üzembehelyezéséhez szükséges.
B. Ábrázolja az összes hibalehetőséget, amellyel a telepítés során találkozhatunk.
C. Ábrázolja a szoftver összetevőket, annak megfelelően, miként kell őket telepítőcsomagba helyezni.
D. Ábrázolja a szoftver összetevőinek fizikai (különböző gépeken történő) elhelyezését, a szükséges szoftverkörnyezettel.

8. Mely objektumorientált elvet sérti az egyke (singleton) szerkezet, amely korábban egy népszerű tervminta volt? Az egyke szerkezet azt garantálja, hogy az objektumból csak egy példány legyen, amelyet egy statikus művelet segítségével kérhetünk el az osztálytól.



A. Single Responsibility Principle
B. Open/Closed Principle
C. Liskov Substitution Principle
D. Dependency Inversion Principle

9. Az alábbiak közül melyik technika használható a Dependency Inversion Principle megvalósítására?

A. (figyelő) observer
B. MVC (modell-view-controller)
C. függőség befecskendezés (dependency injection)
D. általánosítás (generalization)

10. Mit jelent a tesztvezérelt fejlesztés (TDD)?

A. Szoftverfejlesztési módszer, amelyben a teszteket a tényleges programkód elkészítése előtt írják meg.
B. Tesztelési módszer, amelynek célja, hogy az tesztesetek minden programegységre kiterjedjenek, és megfelelő sorrendben hajtódjanak végre.
C. Egy általános elv, amely kimondja, hogy a programkód minden utasítását ellenőrizni kell egységtesztek segítségével (100%-os kódlefedettség).
D. Tesztelési módszer, amelyben először egységteszteket készítenek az osztályokra (és metódusaikra), majd integrációs tesztekkel ellenőrzik az osztályok együttes viselkedését, végül rendszertesztekkel a teljes szoftvert viselkedését ellenőrzik.

11. Az alábbiak közül mely funkciót nem tudják biztosítani a teszt keretrendszerek (unit testing frameworks)?

A. Tesztesetek manuális létrehozását külön programegységekben (osztályokban).
B. Mindent lefedő tesztesetek automatikus generálását a programkód elemzésével.
C. A kapott és elvárt eredmények összehasonlítását elvégző assert utasításokat.
D. Tesztjelentés elkészítését, amelyben látható, hogy mely tesztek lettek sikeresek/sikertelenek.

12. Egység (Unit) tesztelés esetén a program részeit el kell különítenünk egymástól és határokat kell felállítanunk közöttük. Erre az egyik lehetséges megoldás, hogy olyan objektumokat használunk, melyek más objektumok működését utánozzák. Hogyan nevezzük ezeket az objektumokat?

A. Mock objektumoknak
B. Moduloknak
C. Egységeknek
D. Atomoknak

13. Miből indul ki az objektum orientált tervezés?

A. Funkciók
B. Tevékenységek
C. Entitások és kapcsolataik
D. Architektúra

14. Mivel fejezzük ki, hogy egy objektum egy osztály több objektumával is kapcsolatban áll?

A. Kompozícióval
B. Multiplicitással
C. Tömb típusú attribútummal
D. Aggregációval

15. Mihez rendelhetünk egy osztálydiagram esetén megszorítást?

A. Reláció
B. Attribútum
C. Metódus paraméterek
D. Mindhez

16. Az alábbi relációk közül, melyik nem értelmezett osztályok objektumai között?

A. Asszociáció
B. Függőség
C. Származtatás
D. Aggregáció

17. Melyik diagram nem része a dinamikus modellnek?

A. Állapot diagram
B. Szekvencia diagram
C. Tevékenyseg diagram
D. Komponens diagram

18. Mije nem lehet az állapotnak?

A. Neve
B. Invariánsa
C. Előfeltétele
D Paramétere

19. Mivel csökkenthető az állapotdiagram komplexitása?

A. Általánosítás
B. Aggregáció
C. Általánosítással és aggregációval
D. Más módszerrel

20. Melyik igaz?

A. Az általánosítás invariánsa az állapotok invariánsainak diszjunkciója
B. Az általánosítás invariánsa az állapotok invariánsainak konjunkciója
C. Az aggregáció invariánsa az állapotok invariánsainak diszjunkciója
D. A két módszerrel kapott állapotoknak nincs invariánsa

21. Melyik opcióban szereplő diagramok részei a statikus modellnek?

A. Osztálydiagram, Objektumdiagram
B. Osztálydiagram, Állapotdiagram
C. Objektumdiagram, Szekvenciadiagram
D. Objektumdiagram, Aktivációs diagram

22. Mivel csökkenthető az állapotdiagram komplexitása?

A. Az állapotok általánosításával.
B. Az állapotok aggregációjával.
C. Az állapotok általánosításával és/vagy aggregációjával.
D. Más módszerrel.

23. Az alábbi megvalósítás alapján melyik reláció áll fenn pontosan az autó (Car) és a motor (Engine) között?

```
public class Car {
    private Engine engine;

    public Car(Engine engine){
        this.engine = engine;
    }

    public Engine getEngine(){
        return engine;
    }
}
```

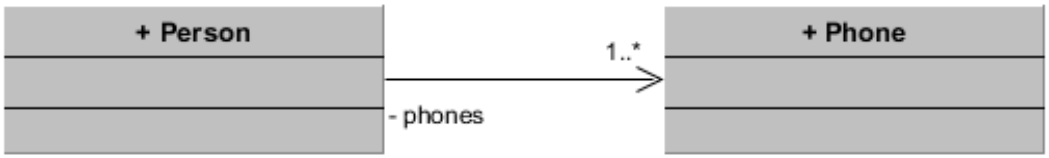
A. Asszociáció (Association)
B. Aggregáció (Aggregation)
C. Kompozíció (Composition)
D. Általánosítás (Generalization)

24. Az alábbi megvalósítás alapján melyik reláció áll fenn pontosan az autó (Car) és a motor (Engine) között?

```
public class Car {
    private final Engine engine;

    public Car(int engineCapacity, int engineSerialNumber) {
        this.engine = new Engine(engineCapacity, engineSerialNumber);
    }

    public int getEngineCapacity() {
        return engine.getEngineCapacity();
    }

    public int getEngineSerialNumber(){
        return engine.getEngineSerialNumber();
    }
}
```

A. Asszociáció (Association)
B. Aggregáció (Aggregation)
C. Kompozíció (Composition)
D. Általánosítás (Generalization)

25. Milyen kapcsolat áll fenn az alábbi osztálydiagramon a Person és a Phone osztály között?



A. Asszociáció (Association)
B. Aggregáció (Aggregation)
C. Kompozíció (Composition)
D. Általánosítás (Generalization)

Implementációs kérdések

1. Mit kell implementálni saját típus eseten a HashMap használatához?

A. == operátor
B. hashCode(…) metódus
C. equals(…) metódus
D. Mindkét metódust

2. Melyik megvalósítást válasszuk az alábbi opciók közül abban az esetben, ha főleg index alapú keresést szeretnénk alkalmazni egy dinamikusan változó méretű adathalmazon, ahol többször is előfordulhat ugyan az az elem? (Új elemet csak a gyűjtemény végére szeretnénk helyezni, törölni a gyűjteményből nem szeretnénk gyakran.)

A. ArrayList
B. LinkedList
C. Tömb (Array)
D. HashSet

3. **Melyik megvalósítást válasszuk az alábbi opciók közül, ha olyan gyűjteményt szeretnénk a feladat során használni, mely nem duplikált elemeket tartalmaz és nincs szükségünk arra, hogy az elemeket a beszúrás sorrendjében vagy az értékek szerint növekvő sorrendben tároljuk?**

A. TreeSet
B. LinkedHashSet
C. HashSet
D. ArrayList

4. Az alábbi interfészek közül melyik implementációit használjuk kulcs-érték párok tárolására?

A. List
B. Set
C. Map
D. Collection

5. Mely állítás igaz?

A. final abstract osztálynak legalább egy metódusa abstract
B. abstract osztálynak legalább egy metódusa abstract
C. final osztály minden attribútuma final
D. abstract osztály leszármazottja is lehet abstract

6. Mely állítas nem igaz?

A. final osztály nem származtatható
B. interface-ek nem származtathatók egymásból
C. Egy osztály több interface-t is megvalósíthat
D. Az interface minden metódusát meg kell valósítani

7. Mi lehet generikus paraméter?

A. Alaptípus
B. Osztály
C. interface
D. Osztály, ami megvalósítja a generikusban használt műveleteket

8. Mely gyűjtemény indexelhető?

A. HashSet
B. HashMap
C. Vector
D. TreeMap

9. Mi lehet statikus?

A. Adattag
B. Metódus
C. Osztály / interface
D. Felsorolási típus

10. Mit támogat a Java a többszörös specializáció és többszörös általánosítás közül?

A. Általanosítás
B. Specializáció
C. Mindkettő
D. Egyik sem

Szoftverfejlesztési módszertanok és modellek

1. **Melyik nem agilis elv a következők közül?**

A. a módszertan érvényesítése, szemben az eszközökkel
B. a működő szoftver, szemben az átfogó dokumentációval
C. együttműködés az ügyféllel, szemben a szerződéses tárgyalásokkal
D. a változásra való reagálás, szemben a terv követésével.

2. **Melyik állítás nem igaz a futam kapcsán?**

A. A terméknek mind a tervezése, kódolása és tesztelése is a futamon belül történik.
B. A futam eredménye üzleti értéket képviselő működő kód.
C. A feladatok és az idők meghatározása után csak a termékgazda szól bele a csapat munkájába.
D. A Scrum csapat önszerveződő módon dolgozik a futam során.

3. **Melyik állítás igaz a Scrum mesterre?**

A. A Scrum mester a Scrum csapat menedzsere
B. A Scrum mester vezeti a napi Scrumot
C. A Scrum mester nem felel azért, hogy külső hatásoktól védje a Scrum csapat munkáját
D. A Scrum mester a folyamatokért felel

4. **Melyik állítás igaz a napi Scrum-ra?**

A. A napi Scrum-ot a Scrum mester vezeti.
B. A napi Scrum során a csapattagok beszámolnak a Scrum mesternek a haladásukról.
C. A napi Scrum során az a cél, hogy felszámoljuk a csapatot érintő akadályokat.
D. A napi Scrum maximum 15 percet tarthat.

5. **A három Scrum termék a következő:**

A. termék kívánságlista, futam feladatlista, Scrum tábla
B. termékvízió, termék kívánságlista, felhasználói történet
C. termék kívánságlista, Scrum tábla, haladási diagram
D. termék kívánságlista, futam feladatlista, inkrementum

6. **A Test-driven development (TDD) egy szoftverfejlesztési módszertan, mely szerint …**

A. a teszteket a tényleges programkód implementálása előtt kell elkészíteni.
B. a teszteket a tényleges programkód implementálása után kell elkészíteni minden egységre.
C. a tesztelő kolléga jóváhagyása után lehet új programkódot implementálni.
D. a tesztjegyzőkönyvet egy nappal az új programkód implementálása előtt kell a dokumentációhoz rendelni.

Verziókezelés

1. Mi a célja a folyamatos integrációs (continuous integration, CI) gyakorlati módszernek?

A. A lehetséges hibák, integrációs problémák azonnali, automatizált kiszűrése, visszajelzés a fejlesztőnek. (Önellenőrző build)
B. Az elbukott integrációstesztek automatikus újra futtatása, ameddig meg nem javulnak.
C. Objektum orientált programozási nyelvre való átállást segíti elő.
D. A manuális tesztelés teljes kiváltása.

2. Mik a centralizált verziókövető rendszerek (Például: SVN, Perforce, CVS) hátrányai?

A. A szerver kitüntetett szerepe. (Meghibásodás esetén használhatatlanná válik a rendszer a szerver javításáig.), Továbbá a verziókezeléshez hálózati kapcsolat szükséges.
B. Fájl alapú műveletvégzés (1 verzió 1 fájl változásai)
C. Lokális tároló
D. Konkurenciakezelés kizárólagos zárak által történik.

3. Melyik állítás nem igaz az elosztott verziókövető rendszerekre (Például: Git, Mercurial)?

A. Decentralizált, elosztott hálózati modellt használnak, ahol a konkurenciakezelés jellemzően a beküldés utáni egyesítéssel történik.
B. Minden kliens rendelkezik a teljes tárolóval és verziótörténettel. A revíziókezelő eszköz műveletei lokálisan, a kliens tárolóján történnek.
C. A kommunikáció peer-to-peer elven történik, de kitüntetett szerverek felállítására is van lehetőség.
D. Fájlhalmaz alapú műveletvégzés jellemző rá, ahol a konkurenciakezelés jellemzően a beküldés előtti egyesítéssel történik.

4. Igaz-e, hogy Git merge esetén nem lehet konfliktus?

A. Igaz, mivel csak rebase esetén alakulhat ki konfliktus.
B. Hamis, mivel minden merge esetén van konfliktus a kollégák között.
C. Igaz, mivel minden merge egyben egy újabb commit is.
D. Hamis, mivel előfordulhat, hogy a git nem tudja megoldani a változások automatikus integrálását. (Például: Két különböző commit egy fájl ugyanazon sorára vonatkozóan tárol módosítást.)

5. Az alábbiak közül, melyek az ismertebb build eszközök?

A. Ant, Maven, Gradle
B. Ant, Git, Subversion (SVN)
C. Ant, Maven, Subversion (SVN)
D. Maven, Gradle, Git

Tervezési elvek, Tervezési minták

1. Mit mond ki a DRY elv?

A. Ne implementáljunk előre olyan kódot, ami „majd a jövőben kelleni fog”, mert szinte biztos, hogy sose lesz rá szükségünk.
B. A tudás minden darabkájának egyetlen, egyértelmű és megbízható reprezentációval kell rendelkeznie egy rendszeren belül.
C. A tökéletességet nem akkor lehet a legjobban megközelíteni, ha egy rendszerhez nem tudunk már semmit hozzáadni, hanem akkor, ha nem tudunk mit elvenni belőle.
D. Az biztosan elmondható, hogy javulni fog a kódbázisunk minősége, ha mindig úgy hagyjuk ott az aktuális kódunkat, hogy az egy kicsit „jobb”, egy kicsit tisztább annál, mint ahogy megtaláltuk.

2. Melyik objektumorientált elvet szegtük meg az alábbi kódrészletben?

```
public enum Status {
    UNRESOLVED, IN_PROGRESS, RESOLVED, CLOSED
}

public class Task {
    protected Status status;

    public void close() {
        this.status = Status.CLOSED;
    }
}

public class ProjectTask extends Task {

    @Override
    public void close() {
        if(status == Status.IN_PROGRESS) {
            throw new RuntimeException("You cannot close …");
        }
        super.close();
    }
}
```

A. Liskov Substitution Principle (Liskov-féle helyettesítési elv)
B. Dependency Inversion Principle (Függőség megfordítása elv)
C. KISS
D. DRY

3. Melyik nem SOLID alapelv az alábbiak közül?

A. Liskov Substitution Principle (Liskov-féle helyettesítési elv)
B. Open/Closed Principle (Nyílt/Zárt elv)
C. Single Responsibility Principle (Egy felelősség elve)
D. Separation of Concerns Principle (A vonatkozások szétválasztásának elve)

4. Melyik objektumorientált elvet szegtük meg az alábbi kódrészletben?

```
public class DamageCalculator {
    private int damageFactor;

    public DamageCalculator(int damageFactor) {
        this.damageFactor = damageFactor;
    }

    public int totalDamage(int baseDamage, int damageAlgorithm){
        switch(damageAlgoritm) {
            case 0: return baseDamage * damageFactor;
            case 1: return baseDamage * 4;
            default: return baseDamage;
        }
    }
}
```

A. Dependency Inversion Principle (Függőségek megfordításának elve)
B. Open/Closed Principle (Nyílt/Zárt elv)
C. Interface segregation Principle (Interface szétválasztási elv)
D. Liskov Substitution Principle (Liskov-féle helyettesítési elv)

5. Adott egy lámpa (Lamp) osztály. A lámpának van színe, illetve ki/be lehet kapcsolni. A lakásunkban a falon található egy kapcsoló (Switch) mely az alábbi módon lett implementálva. Mi lehet a probléma ezzel a megvalósítással?:

```
public class Switch {
    private Lamp lamp;

    public Switch(Lamp lamp) {
        this.lamp = lamp;
    }

    public void useSwitch(){
        if(lamp.isOn()) {
            lamp.turnOff();
        } else {
            lamp.turnOn();
        }
    }
}
```

A. A kapcsoló megsérti a Liskov Substitution Principle-t (Liskov-féle helyettesítési elv)
B. A kapcsoló megsérti az Open/Closed Principle-t (Nyílt/Zárt elv)
C. A kapcsoló magasabb absztrakciós szinten helyezkedik el, mint a lámpa, így megsérül a Dependency Inversion Principle (Függőségek megfordításának elve)
D. A kapcsoló megsérti a Single Responsibility Principle-t (Egy felelősség elve)

6. Melyik tervezési minta nyújt megoldást arra a problémára, ha több objektumot szeretnénk értesíteni, amikor egy másik objektumnak megváltozik az állapota.

A. Singleton (Egyke)
B. Observer (Megfigyelő)
C. Adapter (Illesztő)
D. Factory (Gyártó)

7. Melyik tervezési minta alkalmazható a hosszú paraméterlistájú konstruktorok elkerülésére?

A. Observer (Megfigyelő)
B. Factory (Gyártó)
C. Builder (Építő)
D. Command (Parancs)

8. Melyik tervezési minta megvalósításának része lehet az alábbi kódrészlet?

```
public MyPattern withName(String name) {
    this.name = name;
    return this;
}

public MyPattern withNumber(int number) {
    this.number = number;
    return this;
}
```

A. Singleton (Egyke)
B. Builder (Építő)
C. Command (Parancs)
D. Adapter (Illesztő)

9. Melyik tervezési mintát alkalmazhatjuk abban az esetben, ha konkrét osztály megadása nélkül szeretnénk kapcsolódó vagy egymástól függő objektumok családjának létrehozására felületet biztosítani?

A. Factory method (Gyártó függvény)
B. Adapter (Illesztő)
C. Builder (Építő)
D. Abstract Factory (Absztrakt gyár)

10. Melyik tervezési mintát alkalmazhatjuk abban az esetben, ha egy adott osztály példányosítását szeretnénk a hozzátartozó alosztályokra átruházni?

A. Factory method (Gyártó függvény)
B. Builder (Építő)
C. Command (Parancs)
D. Observer (Megfigyelő)

11. Melyik tervezési mintát valósítja meg az alábbi kódrészlet?

```
public class MyPattern {
    private MyPattern(){}

    private static class MyPatternInstance {
        private static final MyPattern INSTANCE = new MyPattern();
    }
    public static MyPattern getInstance() {
        return MyPatternInstance.INSTANCE;
    }
}
```

A. Singleton (Egyke)
B. Factory (Gyártó)
C. Builder (Építő)
D. Adapter (Illesztő)

12. Melyik tervezési mintát valósítja meg az alábbi kódrészlet?

```
public abstract class Maze {
    private final List<Room> rooms = new ArrayList<>();

    public Maze() {
        Room room1 = makeRoom();
        Room room2 = makeRoom();
        room1.connect(room2);
        rooms.add(room1);
        rooms.add(room2);

    }

    protected abstract Room makeRoom();
}

public class MagicMaze extends Maze {


    @Override
    protected Room makeRoom() {
        return new MagicRoom();
    }
}
```

A. Factory method (Gyártó függvény)
B. Command (Parancs)
C. Adapter (Illesztő)
D. Abstract Factory (Absztrakt gyár)