

# VIZSGA FELADATSOR SZOFTVERTECHNOLÓGIA

c. tárgyból  
2013. január 15.

**Az első lapon található feladatok megoldására 30 perc áll rendelkezésére. Az elérhető 24 pontból minimum 14 pontot kell kapnia ahhoz, hogy a második lapon szereplő feladatokra adott megoldásait értékeljük.**

*A tesztkérdésekre adott rossz válasz esetében pontot veszít, de feladatonként a total pontszám >= 0*

1. Subversion-ben a commit (check in) végrehajtása attól függ, hogy az utolsó letöltés (check out/update) óta módosult-e a letöltött és/vagy a repositoryban tárolt változat.

Az alábbi táblázatba írja be, hogy a változásoktól függően mi történik commit esetén (4 pont)

		repository példány	
		nem változott	változott
munkap éldány	változott	<b>publish</b>	<b>out of date error</b>
	változatlan	-	-

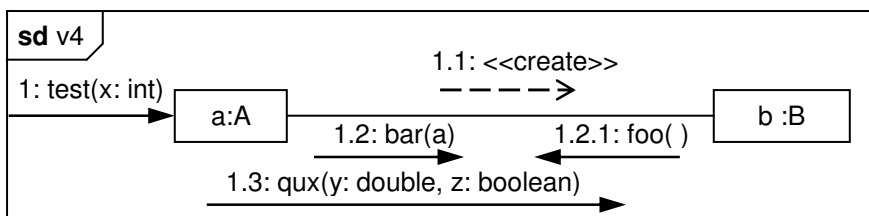
2. Legyen egy *X* osztályunk, *aaa()* és *bbb()* metódusokkal jellemezve. Egy kliens meghívja az *aaa()* metódust. Az *aaa()* futása közben egy másik kliens meghívja a *bbb()* metódust. Az alábbi táblázatba írja be, hogy a különböző UML2 szemantikák esetében mi a követett eljárás (policy)! (5 pont)

szemantika neve	eljárás (policy)
<b>sorrendi (Sequ)</b>	<b>kizárt – nem fordulhat elő</b>
<b>őrzött (Guarded)</b>	<b>befejezi aaa-t, majd elkezd bbb-t</b>
<b>konkurrens (Conc)</b>	<b>félbeszakítja aaa-t és elkezd bbb-t</b>

Miben különbözik, ha a másik kliens is az *aaa()* metódust hívja? (1 pont)

**semmiben** .....

3. Adja meg, hogy az alábbi UML2 kommunikációs diagramon leírt viselkedés megvalósításához minimálisan milyen példány attribútumokat szükséges definiálni az osztályokban! (3 pont)



osztály	nem kell attribútum	attribútum típusa
A	<input type="checkbox"/>	
B	<input type="checkbox"/>	

4. Milyen általános kiterjesztő technikákat (general extension mechanisms) alkalmaz az UML2? (3 pont)

**constraint, stereotype, tagged value** .....

## 5. Adott az alábbi (hibás) Java kódrészlet.

<pre>public class ThreadSafe {     private long x;     private long y;     private Object o = new Object();     public void setX(long v) {         x = v;         y = 1-x;     }     public long getXY() {         return x+y;     }     public synchronized void setY(long w) {         o.notify();         y = w;         x = 1-y;     }     public void finalize() {         System.out.println(             "destructor called");     } }</pre>	<pre>public class MyThread implements Runnable {     private boolean done = false;     private ThreadSafe ts;     public MyThread(ThreadSafe ts)         { this.ts = ts; }     public void exit() { done = true; }     public void run() {         long i = 0;         while (!done) {             ts.setX(i++);         }         ts.finalize();         ts.getXY();     } }  public class Main {     public static void main(String[] args) {         MyThread mt = new MyThread             (new ThreadSafe());          mt.start();     } }</pre>
---	---

Jellemezze az alábbi állításokat a kulcs felhasználásával! (8 pont)

- A** - csak az első tagmondat igaz (+ -)
- B** - csak a második tagmondat igaz (- +)
- C** - mindkét tagmondat igaz, de a következtetés hamis (+ + -)
- D** - mindkét tagmondat igaz és a következtetés is helyes (+ + +)
- E** - egyik tagmondat sem igaz (- -)

- [B]** Egy **ThreadSafe** objektum belső állapota mindig konzisztens marad a **setX** metódus több szálból egyszerre történő hívása során is, mert a metódus törzse nem tartalmaz feltételes elágazást.
- [C]** A **ThreadSafe** típusú objektumok használata esetén a **getXY** metódus nem mindig 1-gyel tér vissza, mert egyszerre több szál is be tud lépni a metódusba.
- [E]** Az **o.notify()**-t **synchronized** metódusból kell hívni, ezért az **o.notify()** hívás a **setY** metódusban nem dob **IllegalMonitorStateException** kivételt.
- [A]** A **main** metódusban az **mt.start()** hívás hibás, mert a szálat az **mt.run()** hívással kell elindítani.
- [A]** A **MyThread** szál megállítására használt **done** attribútum hiányosan van definiálva, mert a **transient** kulcsszóval jelezni kell, hogy az attribútum értékét más szál is módosíthatja.
- [E]** A **MyThread.run** metódusában a **ts.finalize()** hívás meghívja a garbage collectort és felszabadítja a **ts** objektumot, ezért az ezt követő **ts.getXY()** hívás **NullPointerException** kivételt fog dobni.
- [B]** A **ThreadSafe** típusú objektumok többszálú használata esetén az **x+y==1** invariáns mindig érvényes marad, mert egy **MyThread** szál csak egyetlen **ThreadSafe** objektumot használ.
- [E]** A **MyThread** konstruktorában a **ts** paraméter neve nem egyezhet meg a **ts** attribútum nevével, mert így a **this.ts = ts** utasításnak semmi hatása nincs.

6. Felsoroltunk szoftverrel kapcsolatos "hibákat". A hibák mellett jelölje be, hogy az melyik kategóriába tartozik! (4 pont)

	failure	error	bug	fault
A ciklusfeltétel hibás	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Alábecsültük a példányosítás erőforrás-igényét	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Hiányzik a "synchronized" kulcsszó	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sok objektumnál nagyon lassan kapjuk az eredményt	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Elmaradt a kritikus kódok felülvizsgálata	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Az első futásnál hibás eredményt kapunk	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

7. Adott az alábbi dekorált XML leírás.

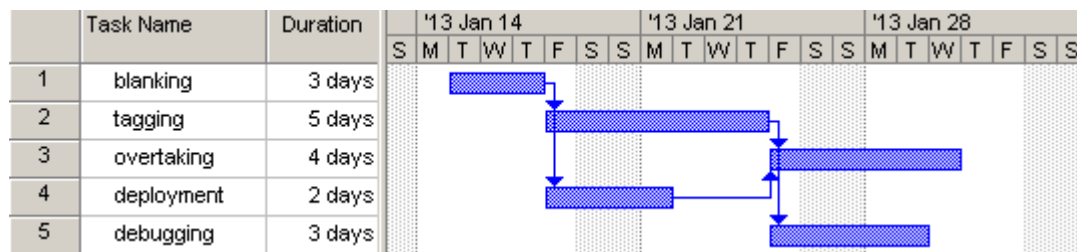
```
<!DOCTYPE y [
  <!ELEMENT y (a+, b)>
  <!ELEMENT a ANY>
  <!ELEMENT b (#PCDATA)>
  <!ELEMENT c (#PCDATA |y)*>
]>
<y>
  <a><y><a>
    ②
    <b></b></y>
  a/</a><b>/b</b>
</y>
```

Mi állhat ② helyében, hogy az XML érvényes legyen? (6 pont)

- <b><c>/b</c></a>b</a>
- <a><c>b</c></a>b</a>
- <b>b/b</b></a>b</a>
- <c><b></a>b</c></y>
- <a><b>a</b>/b</a></a>
- <y><c></b></a>b</c></y>
- <b><c></b></a>b</c></a>
- <y><c>b</c></y>

8. Egy ma (január 15., kedd) induló projekt feladatait, azok hosszát és függéseit a mellékelt táblázat foglalja össze. Rajzoljon Gantt-diagramot ! (szombaton és vasárnap nem dolgozunk !) (3 pont)

szám	feladat	nap	függés
1	blinking	3	
2	tagging	5	1
3	overtaking	4	2, 4
4	deployment	2	1
5	debugging	3	2

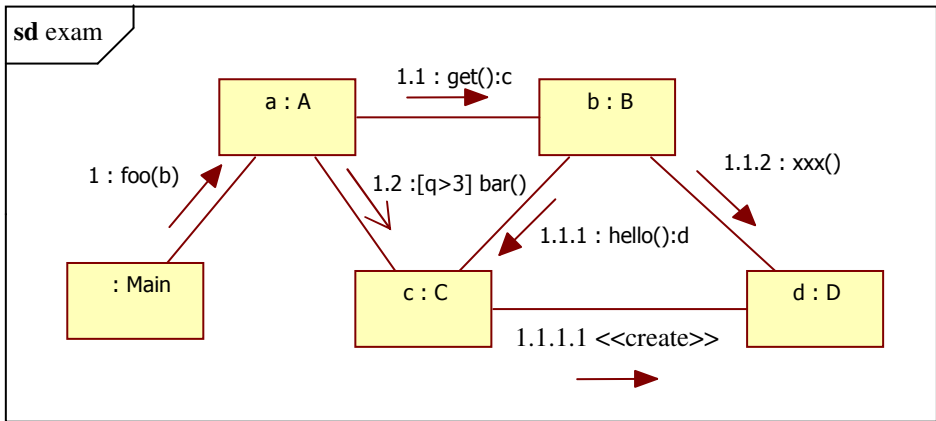


Hányadikán kezdetjük legkésőbb a „deployment”-et, ha a véghatáridőt tartani akarjuk (2 pont)? **jan 23.-án**

9. Az alábbi Java kódrészletek alapján készítsen UML 2 kommunikációs diagramot! A diagramot a START-tal jelölt megjegyzéstől kezdje! Tételezze fel, hogy a C.bar() metódus aszinkron! (7 pont)

```
public class A {
    double q;
    static protected double qux(double d){
        q = d/2;
        return 2*d;
    }
    public void foo(B b) {
        C c = b.get();
        if (q > 3) c.bar();
    }
}
public class B {
    C c;
    public void set(C x) {
        c = x;
        c.bar();
    }
    public C get() {
        c.hello().xxx();
        return c;
    }
}
}
```

```
public class C {
    public void bar() {
    }
    public X hello() {
        return new D();
    }
}
public class D implements X {
    public void xxx() {}
}
public class Main {
    public static void main(String args[]) {
        A a = new A();
        B b = new B();
        C c = new C();
        b.set(c);
        // START
        a.foo(b);
    }
}
public interface X {
    void xxx();
}
}
```



10. Az UML2-ben definiált **Sequence** gyűjteménynek (kollekciónak) adja meg a tulajdonságait! (4 pont)

Eredmények értékelése:

Pontszám	Osztályzat
21 -	2
28 -	3
35 -	4
42 -	5

igen	nem	nem jellemző	tulajdonság
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	egyedi (unique)
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	minősített (qualified)
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	rendezett (ordered)
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	delegált (delegated)