

Java - megoldások

2011.12.20 – 2. Feladat

Töltse ki az alábbi kódrészlet hiányzó részeit a szabványos Java API elemeivel úgy, hogy mind szintaktikailag, mind szemantikailag helyes megoldás szülessen!

```
FileInputStream fis = new FileInputStream("test.txt");  
InputStreamReader foo = new InputStreamReader.(fis);  
char c = foo.read( );
```

Adja meg az egyenlőségjel utáni kódrészlet módosított változatát, hogy olyan fájlból tudjunk beolvasni, aminek a létrehozásakor a `GZIPOutputStream` osztályt használtuk!

```
new InputStreamReader(new GZIPInputStream(fis))
```

Milyen szál állapot-értékkel tér vissza a Javában a `Thread.currentThread().getState()`?

RUNNABLE (esetleg RUNNING)

Jelölje (karikázza be) a Java nyelvvel kapcsolatos állítások igazságtartalmát!

I H Egy változó statikus típusa nem lehet absztrakt osztály.

I H `JScrollPane`-be csak olyan `SWING` komponenst szabad tenni, aminek nincs saját görgetősávja.

2011.12.20 – 8. Feladat

A kulcs felhasználásával jellemezze a Java nyelvvel kapcsolatos állításokat!

[B] Minden `List` interfészt megvalósító objektum értékül adható `Set` típusú változónak, mert a `Set` minden metódusa megtalálható a `List`-ben is.

[D] Egy objektum `wait` metódusát csak az objektum monitorában tartózkodó szál hívhatja meg, mert a `wait` hívása során a szál kilép az objektum monitorából.

[B] Absztrakt osztálynak nem lehet `final` metódusa, mert az osztályból nem lehet példányt létrehozni.

2012.01.03 – 2. Feladat

A kulcs felhasználásával jellemezze a Java nyelvvel kapcsolatos állításokat!

[E] Egy X osztály foo synchronized metódusa nem lehet final, mert az X leszármazottaiban a foo metódusban az őosztály egy objektumának monitorát kellene használni.

[B] A private hozzáférési osztályú attribútumok nem szerializálódnak, mert ezeket csak az adott osztály metódusai érhetik el közvetlenül.

[A] A java.awt.event csomagban a WindowAdapter osztály megvalósítja a WindowListener interfészt, mert a WindowAdapter osztály használatával lehetővé válik a többszörös öröklés.

```
Map<String, Integer> ht = new HashMap<String, Integer>();
for (int i = 0; i < 3; i++) {
    if (!ht.containsKey("alma"))
        ht.put("alma", 1);
    Integer n = ht.get("alma");
    n += i;
}
System.out.println(ht.get("alma"));
```

1						

2012.01.03 – 3. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H Egy objektum referenciáját tartalmazó változón csak olyan metódus hívható meg, amilyen a változó statikus típusában is szerepel.

I H Egy változó dinamikus típusa nem lehet absztrakt osztály.

I H Előfordulhat, hogy két szál (T1 és T2) ugyanazon objektum ugyanazon synchronized metódusát futtatva T1 T2 sorrendben lép be, de T2 T1 sorrendben lép ki.

I H Ha a t tömböt paraméterként átadjuk egy metódusnak, amelyik a tömb egy elemének új értéket ad, akkor az eredeti tömbben is módosul az érték.

2012.01.17 – 2. Feladat

Sorrendben adja meg, hogy a `foo` metódus teljes lefutása során a futtató szál milyen állapotokat vesz fel! (Segítségül megadtuk az induló állapotot.) Tételezze fel, hogy a metódus meghívásakor egy másik szál ugyanazon objektum `bar` metódusát hajtja végre!

```
synchronized void bar() {
    ...
}

synchronized public void foo() {
    try {
        this.wait();
    } catch (Exception e) { }
}
```

1	RUNNABLE (RUNNING)
2	BLOCKED
3	RUNNABLE (RUNNING)
4	WAITING
5	BLOCKED
6	RUNNABLE (RUNNING)
7	

2012.01.17 – 3. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H Egy változó dinamikus típusának nem lehet *abstract* metódusa.

I H Egy változó statikus típusának nem lehet *final* metódusa.

I H Egy *interface* típusnak csak egyetlen közvetlen őse lehet.

I H Az alábbi kódrészlet futtatása végén a *b* változó értéke *true*.

```
String s1 = new String("Hello");
String s2 = s1;
s2 += " world";
boolean b = (s1 == s2);
```

2012.01.17 – 6. Feladat

A kulcs felhasználásával jellemezze az alábbi Java kódrészlettel kapcsolatos állításokat!

```
public class X {
    private long l;
    private static void foo() { System.out.println("hello"); bar(); }
    public void bar() { System.out.println("world"); }
    public void bar(X x) { System.out.println("world"); x.l=10; }
    protected X copy() { return new X(); }
    public void do() { bar(copy()); }
}
```

[A] A foo() metódusban hibás a bar() hívása, mert privát láthatóságú metódusból nem hívhatunk publikusat.

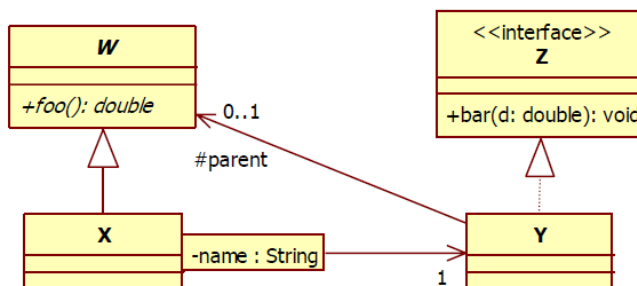
[A] A do metódus fordításkor hibát jelez, mert publikus metódus paraméterének nem lehet protected metódus visszatérési értékét adni.

[E] X osztály nem példányosítható, mert nincs paraméter nélküli konstruktora.

[B] X csak egyetlen másik osztálynak lehet a közvetlen őse, mert a Javában nincs többszörös öröklés.

2012.01.17 – 9. Feladat

Izidor karácsonyra kapta az alábbi szép UML2 osztálydiagramot, azzal, hogy írja meg az ábrán látható osztályok és interfészek Java forráskódját. Izidor még nem készült fel a Szoftch vizsgára, ezért nem megy neki a dolog. Segíts neki, és készítsd el te a kódot! Minden, az ábrából következő metódust és attribútumot vegyél fel! Ha szükséges, használd a szabványos Java API osztályait is! Minden osztály publikus. A metódusok törzse legyen üres!



```
public abstract class W {
    public abstract double foo();
}
```

```
public class X extends W {
    private Map<String, Y> hm;
    public double foo() {}
}
```

```
public class Y implements Z {
    protected W parent;
    public void bar(double d) {}
}
```

```
public interface Z {
    public void bar(double d);
}
```

2012.05.22 – 1. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H *for* (*S x : z*) fejlécű *for* ciklusban a *z* referencia csak tömbre vagy a JDK-val szállított gyári kollektciók példányaira referálhat.

I H egy szál egyszerre csak egy objektum monitorában tartózkodhat.

I H *synchronized* kulcsszó használatával elkerüljük a deadlock kialakulását.

I H szálak nem képesek saját magukat közvetlenül *waiting* állapotból *notify*-jal felébreszteni.

I H előfordulhat, hogy két szál (T1 és T2) ugyanazon objektum ugyanazon *synchronized* metódusát futtatva T1 T2 sorrendben lép be, de T2 T1 sorrendben lép ki.

I H egy változó statikus típusa nem lehet a változó dinamikus típusának leszármazottja.

I H egy metódust el lehet látni egyszerre *abstract* és *final* módosítóval is.

2012.05.22 – 4. Feladat

Töltse ki az alábbi kódrészlet hiányzó részeit a szabványos Java API elemeivel úgy, hogy mind szintaktikailag, mind szemantikailag helyes megoldás szülessen!

```
FileOutputStream fis = new FileOutputStream("test.txt");
OutputStreamWriter foo = new OutputStreamWriter (fis);
foo.write('A'); foo.write('B');
```

Adja meg az egyenlőségjel utáni kódrészlet módosított változatát, hogy olyan fájlba tudjunk írni, aminek a beolvasásához a *GZIPInputStream* osztályt kell használnunk!

```
new OutputStreamWriter(new GZIPOutputStream(fis));
```

2012.05.22 – 9. Feladat

Adja meg, hogy az alábbi UML2 kollektció-jellemzők definiálása esetén melyik *java.util*-beli kollektcióinterfészt használhatjuk! Adjon meg mindegyikhez egy tipikus *java.util*-beli megvalósítást is!

UML	Java util interfész	Java util megvalósítás
ordered	List	ArrayList, Vector
unique	Set	HashSet
unique, ordered	SortedSet	TreeSet
qualified	Map	HashMap, Hashtable
qualified, ordered	SortedMap	Treemap

2012.06.05 – 1. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H Statikus attribútumot csak statikus metódusból lehet elérni.

I H Példánymetódust kivétel nélkül csak példánymetódusból hívhatunk.

I H Egy osztály lehet akkor is absztrakt, ha nincs absztrakt metódusa.

I H A *java.lang.String* osztálynak vannak a string értékét (tartalmát) megváltoztató metódusai.

I H A primitív típusokhoz tartozó csomagoló osztályok (wrapper classes) nem változtathatók (immutable)

2012.06.05 – 2. Feladat

Adja meg, hogy az alábbi állítások melyik szabványos Java kivételosztályra igazak! Ha többre is igaz, akkor az öröklési hierarchiában legmagasabban levő (ős)osztályt jelölje meg!

A Throwable

B Exception

C InterruptedException

D RuntimeException

E NullPointerException

F Error

[A] Minden kivételként eldobható osztály ősoztálya.

[F] JVM szintű rendszerhiba

[F] Elkapása nem kötelező, nehéz rá felkészülni, nehéz lekezelni.

[B] Programhiba

[D] Elkapása nem kötelező, "zajt" vinne a kódba.

[A] Az *initCause* metódusával be lehet állítani az okozó kivételt

2012.06.05 – 4. Feladat

A *Java.util.Stack<E>* osztály metódusai közötti kohézió milyen típusú?

Kommunikációs (ugyanazon adatszerkezeten operálnak).

2012.06.05 – 5. Feladat

Mi a szerepe a JUnit-ban az alábbi annotációknak?

@BeforeClass - **Tesztosztály közös inicializálása, erőforrások lefoglalása.**

@After - **Egyedi tesztet lefutása után erőforrások elengedése.**

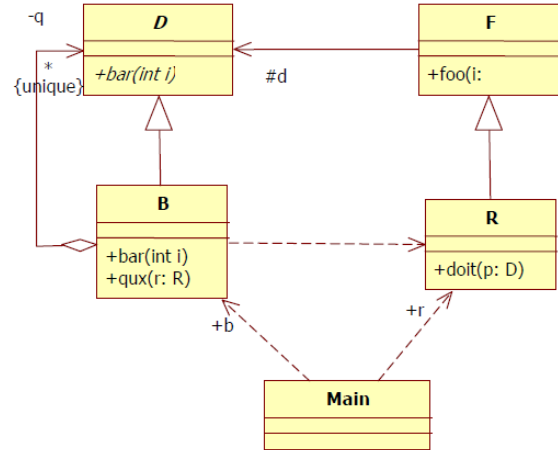
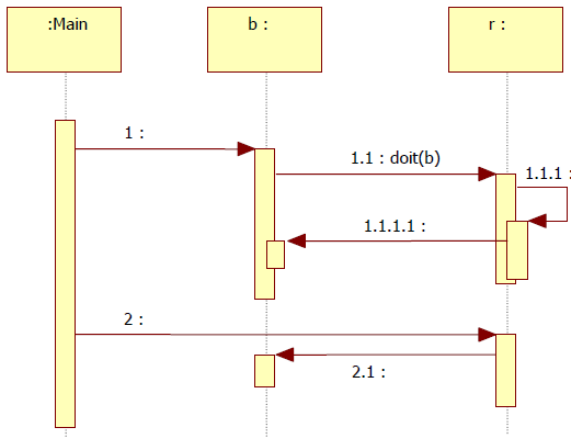
Egy JUnit teszt eredménye *pass* vagy *fail* vagy *error* lehet. A *pass* a tesztben foglalt állítás beteljesülését jelenti. Mi a jelentése a másik két eredménynek?

fail – **Az állítás nem teljesül.**

error – **A tesztmetódus általa nem kezelt kivételt dob.**

2012.05.06 – 9. Feladat

Java nyelven implementálja az alábbi osztály és szekvenciadiagramon megtervezett osztályokat és metódusokat! A Main osztály működése a main metódusban valósul meg, ami a futása elején létrehoz egy B és egy R objektumot (ez az ábrán helyhiány miatt nincs jelölve).



```

abstract class D {
    public abstract void bar(int i);
}
    
```

```

class B extends D {
    private Set<D> q;
    public void bar(int i) {...}
    public void qux(R r) {
        r.doit(this);
    }
}
    
```

```

class F {
    protected D d;
    public void foo(int i) {
        d.bar(i);
    }
}
    
```

```

class R extends F {
    public void doit(D p) {
        d = p;
        foo(3);
    }
}
    
```

```

public class Main {
    public static void main(String[] args) {
        B b = new B();
        R r = new R();
        b.qux(r);
        r.foo(7);
    }
}
    
```

2012.06.12 – 2. Feladat

A dobozban adott egy Test osztály. Mit ír ki az alatta álló kódrészlet, ha az őt befoglaló kódról feltételezzük, hogy hibátlan?

```
FileOutputStream os = new FileOutputStream("test.sr");
ObjectOutputStream oos = new ObjectOutputStream(os);
Test t = new Test();
t.foo(1,2,3,4);
oos.writeObject(t);
t.foo(6,7,8,9);
oos.writeObject(t);
oos.close();
```

```
FileInputStream is = new FileInputStream("test.sr");
ObjectInputStream ois = new ObjectInputStream(is);
Test t2 = (Test)ois.readObject();
Test t3 = (Test)ois.readObject();
t2.bar();
t3.bar();
```

```
class Test implements java.io.Serializable {
    public int a;
    public static int b;
    public transient int c;
    private int d;
    void foo (int i1, int i2, int i3, int i4) {
        a=i1; b=i2; c=i3; d=i4;
    }
    void bar() {
        System.out.println(a+" "+b+" "+c+" "+d);
    }
}
```

1	7	0	4						
1	7	0	4						

2012.06.12 – 3. Feladat

JUnittal tesztelni akarjuk a nem kezelt kivételt. Írjon olyan tesztmetódust, amely akkor fut le sikeresen, ha nem kezelt ArithmeticException (pl. nullával osztáskor) kivételt dob?

```
@Test(expected=ArithmeticException.class)
public void test() {program, ami kivételt dob}
```

2012.06.12 – 6. Feladat

Az alábbi Java kódrészlet alapján töltsse ki a táblázatot. A rubrikákba azt kell beírni, hogy a két osztály közé milyen jelölést kellene rajzolni UML 2 osztálydiagram esetén. Minden rubrikába egy számot és egy betűt kell írni a kulcs alapján: a szám jelzi a vonal stílusát, a betű a **fejlécben** levő osztály oldalára kerülő végződést.

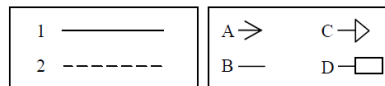
```
class A {
    String name;
    public E e;
}

class B extends E {
    public void foo(C c) {}
    public D foo() { return new D(); }
}

class C {
    private Map<String, A> list;
}

class D {
    public void bar(A a) { a.e.qux(); }
}

class E {
    public void qux() {}
}
```



	A	B	C	D	E
A	-	-	1D	2B	1A
B	-	-	2A	2A	1C
C	1A	2B	-	-	-
D	2A	2B	-	-	2A
E	1B	1B	-	2B	-

2012.06.12 – 7. Feladat

Egy programban két Java szálunk van, *t1* és *t2*. Mindkettő ugyanazt a Qux objektumot ismeri (*q*). A *t1* meghívja a *q.foo()*, majd 1 mp múlva *t2* a *q.bar()* metódusát. A metódusok lefutása után a szálak megállnak (végetérnek). Milyen sorrendben zajlanak le a számozott sorok, és az egyes sorok hatására milyen állapotba kerülnek a szálak?

```
public class Qux {
    synchronized void foo() throws InterruptedException { //1
        Thread.sleep(10000); // 10 mp-ig alszik //2
        wait(); //3
    } //4

    synchronized void bar() { //5
        notifyAll(); //6
    } //7
}
```

Esemény (sor)	t1	t2
0	Runnable	Runnable
1	Runnable	Runnable
2	Timed waiting (sleep)	Runnable
5	Timed waiting (sleep)	Blocked
3	Waiting	Runnable
6	Blocked	Runnable
7	Runnable	Terminated
4	Terminated	Terminated

2012.12.18 – 1. Feladat

Adott az alábbi (hibás) Java kódrészlet.

```
public class Valami extends Exception implements Serializable {
    private transient int foo;
    public boolean bar;
    protected static String baz;
    public Valami(String s) { baz = s; }
}

public class Main {
    public void serialize(String file, Set<Valami> sv) {
        try {
            FileOutputStream os = new FileOutputStream(file);
            ObjectOutputStream oos = new ObjectOutputStream(os);
            oos.writeObject(sv);
            os.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Jellemezze az alábbi állításokat

- [B] A **Main** osztály nem tud sorosítani, mert a **Main** osztály nem implementálja a **Serializable** interfészt.
- [E] A **serialize** metódus **sv** paramétere nem sorosítható, mert interfész típusú változót nem lehet sorosítani.
- [E] A **Valami** konstruktorában a **baz** attribútumnak egyenlőségjellel nem adható értékül az **s** paraméter, mert csak a következő forma lenne helyes: **baz = new String(s)**
- [A] Egy **Valami** típusú objektum **foo** attribútuma nem kerül sorosításra, mert **private** láthatóságú attribútum nem sorosítható.
- [D] A **Main** osztály **serialize** metódusa tartalmaz hibát, mert hiányzik belőle egy **oos.close()** hívás.
- [B] A **Valami** osztály egy példánya nem sorosítható a **writeObject** metódussal, mert **Valami** az **Exception** leszármazottja.
- [B] **protected** módosítóval ellátott attribútum nem sorosítható, ezért a **Valami** osztályban definiált **baz** attribútum nem kerül sorosításra.
- [B] A **FileOutputStream** konstruktorhívása hibás, mert nem **File** típusú paramétert kap.

2012.12.18 – 4. Feladat

Legyen az alábbi A osztályunk.

```
class A {
    protected int j;
    public int foo(int i) {return(j);}
}
```

Legyen egy B osztály, amely A-ból származik, és metódusai az alábbi táblázatban találhatóak. Jelölje meg az(oka)t a metódus(oka)t, amely(ek) megsérti(k) a Meyer-féle nyit-zár (open-close) elvet!

```
public int foo1(int i) {return(j);}
public double foo(double d, int i) {return(d*j);}
public int foo(int i) {return(j*i);}
public int foo(double d) {return((int)(j*d));}
public double foo2(int i) {return(3.0*i);}
```

2013.01.08 – 1. Feladat

Jelölje, hogy az oszlopokban szereplő Java nyelvi elemek milyen módosítókkal rendelkezhetnek.

	konstruktor	statikus metódus	példány metódus	példány attribútum
abstract			X	
final		X	X	X
protected	X	X	X	X
static		X		
synchronized		X	X	
transient				X
volatile				X

2013.01.08 – 6. Feladat

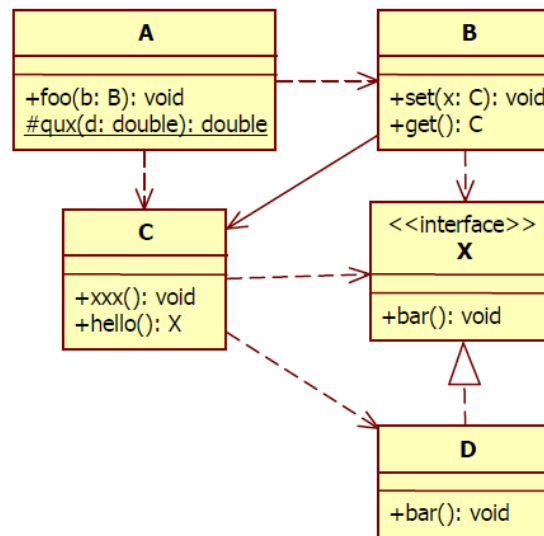
Készítsen Java metódust, amelyik két, X típust tartalmazó halmazt kap paraméterül, és visszaadja a két halmaz uniójának elemszámát. Az eredeti halmazok nem változhatnak meg, az elemekhez egyedileg nem szabad hozzáférni! Ahol lehet, használja a collection framework osztályait és interfészeit! Nem használhat default konstruktort! Törekedjen minél általánosabb megoldásra!

```
int unionSize(Set<X> s1, Set<X> s2) {
    Set<X> r = new HashSet<X>(s1);
    r.addAll(s2);
    return r.size();
}
```

2013.01.08 – 7. Feladat

A Java kódrészletek alapján rajzoljon UML 2 osztálydiagramot!

<pre>public interface X { void bar(); } public class B { C c; public void set(C x) { c = x; c.xxx(); } public C get() { c.hello().bar(); return c; } } public class D implements X { public void bar() {} }</pre>	<pre>public class C { public void xxx() { } public X hello() { xxx(); return new D(); } } public class A { static protected double qux(double d) { return 2*d; } public void foo(B b) { C c = b.get(); c.xxx(); } }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



2013.01.15 – 5. Feladat

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>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[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.

[A] 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.

2013.05.28 – 1. Feladat

Jelölje az alábbi, Java nyelvre vonatkozó állítások igazságtartalmát!

I H konstruktornak nem lehet láthatósága

I H private tag nem szerializálódik

I H statikus tag nem szerializálódik

I H lehet olyan private tag, aminek többször is lehet értéket adni

I H privát módszert csak privát módszerekből lehet hívni

I H statikus módszerben használható a this változó

I H final módszerben használható a this változó

I H statikus módszer nem lehet private

I H final módszer nem lehet statikus

I H final módszer nem lehet abstract

I H absztrakt osztálynak nem lehet final módszere

I H két interfész csak akkor valósítható meg egy osztályban, ha az interfészeknek nincsen közös módszere

2013.06.11 – 3. Feladat

Töltse ki a táblázatot a Java gyűjtemény-keretrendszer osztály és interfészneveivel, amelyre igazak a táblázat peremén található állítások! Egy dobozba egy interfész és egy (az interfészt megvalósító) osztály nevét írja be!

	Elemek egyediek (unique)	Elemek nem egyediek (non unique)
Elemek rendezettek (ordered)	SortedSet: TreeSet	List: ArrayList, Vector, LinkedList
Elemek nem rendezettek (non ordered)	Set: HashSet	ZZZZzzzzzzZZZZZ

Húzza alá azt a módszert, amelyik a fenti osztályok példányain meghívható!

join, sleep, **wait**, interrupt, **notify**

2013.06.11 – 10. Feladat

Adott az alábbi **Student** generikus Java osztály, amelyet tárgy (C) típusal lehet paraméterezni.

```
public class Student<C> implements Cloneable {
    private String name; // név
    private String neptun; // neptunkód
    private Set<C> courses; // felvett tárgyak
    public Student(String na, String ne) {
        name = na;
        neptun = ne;
        courses = new HashSet<C>();
    }

    public void setName(String s) { name = s; }
    public void addCourse(C c) { courses.add(c); }
    public void delCourse(C c) { courses.remove(c); }
}
```

Implementálja meg Javában a fenti **Student** osztály következő metódusait! Nem használható ciklus és a paraméterek állapotát nem módosíthatja!

```
public boolean equals(Object o) {
    /*Két hallgató akkor egyezik, ha a nevük és a neptunkódjuk is azonos.
    Feltételezheti, hogy a metódus csak Student dinamikus típusú objektumot kap
    paraméterül.*/
    Student<C> s = (Student<C>)o;
    return (s.name.equals(name) && s.neptun.equals(neptun));
}

public Object clone() {
    /*Visszatér egy olyan Student példánnyal, amelynek tartalma azonos az
    eredetivel, de a set, add és del kezdetű metódusok hívásakor csak a hívott
    objektum tartalma változik (deep clone).*/
    Student<C> s = new Student<C>(name, neptun);
    s.courses = new HashSet<C>(courses);
    return s;
}

static <C> int numOfSharedCourses(Student<C> s1,
                                  Student<C> s2) {
    /*visszatérési értéke a két hallgató közös tárgyainak darabszáma.*/
    HashSet<C> hs = new HashSet<C>(s1.courses);
    hs.retainAll(s2.courses);
    return hs.size();
}
```

2013.06.18 – 2. Feladat

Mire használjuk a JUnit-ban a **@AfterClass** annotációval megjelölt metódust?

A tesztosztály közös lezárása, erőforrások felszabadítása.

Mit történt, ha a JUnit teszt eredménye **error**?

A tesztmetódus általa nem kezelt kivételt dobott.

2013.06.18 – 3. Feladat

A szabványos Java nyelvben az alábbiak közül mely hívások hatására hagyja el biztosan a szál a futó állapotát?

yield()	sleep(1000)
wait()	stop()
notify()	egy másik élő szálra történő join() hívás
notifyAll()	Thread.killThread()

2013.06.18 – 4. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H egy szál egyszerre csak egy objektum monitorában tartózkodhat.

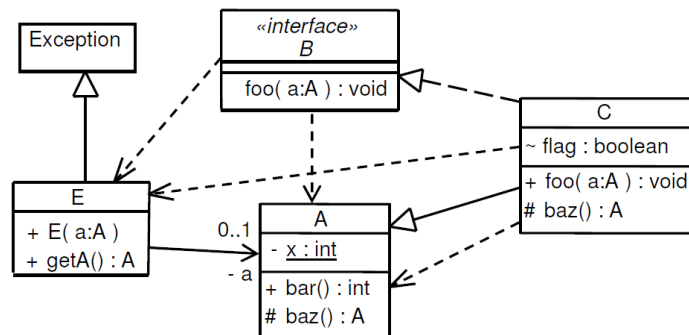
I H szálak nem képesek saját magukat közvetlenül *waiting* állapotból notify-jal felébreszteni.

I H egy változó statikus típusa nem lehet a változó dinamikus típusának leszármazottja.

2013.06.18 – 8. Feladat

Rajzoljon UML 2 osztálydiagramot az alábbi Java kódrészlet alapján!

<pre>class A { private static int x = 0; public int bar() { return ++x; } protected A baz() { return new A(); } } class E extends Exception { private A a; public E(A a) { this.a = a; } public A getA() { return a; } }</pre>	<pre>interface B { void foo(A a) throws E; } class C extends A implements B { boolean flag; public void foo(A a) throws E { flag = !flag; if(a.bar()>5&&flag) throw new E(baz()); } protected A baz() { return new C(); } }</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



2014.01.07 – 2. Feladat

JUnittal tesztelni akarjuk a nem kezelt kivételt. Az alábbi lehetőségek közül mely tesztmetódusok futnak le sikeresen, ha a program nem kezelt `ArithmeticException` (pl. nullával osztáskor) kivételt dob? Jelölje X-szel a helyes választ!

	<code>@Test(throwable=ArithmeticException)</code> <code>public void test() {program, ami kivételt dob}</code>
	<code>@Test</code> <code>public void test() throws ArithmeticException</code> <code>{program, ami kivételt dob}</code>
X	<code>@Test(expected=ArithmeticException.class)</code> <code>public void test() {program, ami kivételt dob}</code>
	JUnittal nem kezelt kivétel nem tesztelhető.

2014.01.07 – 4. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H *final* osztálynak nem lehet *abstract* metódusa.

I H minden generikus osztály használható *Object* paraméterezéssel.

I H generikus osztály paramétere nem lehet primitív típus.

I H primitív típus tömbje is a primitív típusok közé számít.

I H ha egy szál véget ért, akkor *start()* metódushívással újraindítható.

I H egy szál csak akkor hajthat végre *wait()* metódushívást, ha a hívott objektum monitorában tartózkodik.

I H ha egy szál *notify()* hívás hatására hagyja el a *WAIT* állapotot, akkor *RUNNABLE* állapotba kerül.

I H *String* objektum tartalma bármikor megváltoztatható.

I H egy osztály statikus metódusát csak statikus metódusból lehet meghívni.

2014.01.14 – 3. Feladat

Jelölje az alábbi, Java nyelvre vonatkozó állítások igazságtartalmát!

I H catch blokkjában lehet újonnan létrehozott kivételt dobni.

I H statikus metódus nem dobhat kivételt.

I H abstract osztálynak lehet final metódusa.

I H final metódus módosíthatja az objektum állapotát.

I H egy szál csak akkor hajthat végre *notify()* metódushívást, ha a hívott objektum monitorában tartózkodik.

I H generikus osztály példányosításakor lehet másik generikus osztály a paraméter.

I H csak primitív típusokon értelmezett a "természetes rendezés" (natural ordering).

2014.01.21 – 3. Feladat

Tekintsük a következő Java kódot! Jelölje be a táblázatban, hogy a fejlécben szereplő változók által mutatott objektumok a program mely pontján válhatnak először a garbage collector áldozatává!

<pre>class X { private String f = "fff"; public String g = "ggg"; public void gee() { System.out.println(f+g); this.g = null; } @Override public void finalize() { this.f = null; } }</pre>	<pre>public class Program { public static void main(String[] args) { X x1 = new X(); X x2 = new X(); x1.finalize(); /*A*/ x2.gee(); /*B*/ x2 = null; /*C*/ } /*D*/ }</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	x1	x1.f	x1.g	x2	x2.f	x2.g
/*A*/						
/*B*/						
/*C*/						
/*D*/						

2014.01.21 – 8. Feladat

Jelölje az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H minden primitív típusnak van csomagoló (wrapper) osztálya.

I H primitív típus is lehet generikus osztály template-paramétere.

I H ha egy szál véget ért, nem lehet újraindítani.

I H szálakat a *run()* metódus meghívásával indíthatunk.

I H szerializálás körkörös hivatkozású adatszerkezeten (pl. *gy_r_*) kivételt dob.

I H minden objektumnak van *wait()* metódusa.

I H nincs olyan várakozó szál, amelyik egyből RUNNABLE állapotú lesz a *notifyAll()* hatására.

I H szálakon a *join()* metódust csak az indításuk sorrendjében szabad meghívni.

2014.05.27 – 3. Feladat

Jelölje az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H Lehet olyan objektumot létrehozni, amely nem dobható kivételként.

I H Ha a standard inputról ciklusban olvasunk be sorokat, akkor a *BufferedReader(new InputStreamReader(System.in))* objektum létrehozását a cikluson kívül kell elhelyezni, nem pedig a cikluson belül.

I H A standard *Java.lang* csomagban vannak olyan osztályok, amelyek példányai nem szerializálhatók.

I H A primitív típusokhoz tartozó csomagoló osztályok (wrapper classes) nem változtathatók (immutable)

I H Egy szálakat csak a szál *start()* függvényével szabad elindítani, és csak a *stop()* függvényével szabad leállítani.

I H A *wait()* függvény csak olyan objektumon hívható, amelyre rászinkronizáltunk.

I H Egy változó statikus típusa nem lehet a változó dinamikus típusának leszármazottja

I H A *String* osztályhoz úgy adhatunk saját függvényeket, hogy egy leszármazottat készítünk belőle.

I H Egy szál egyszerre csak egy objektum monitorában tartózkodhat.

2014.06.03 – 3. Feladat

Jelölje az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk!

I H abstract osztálynak lehet final metódusa.

I H primitív típus lehet generikus osztály template-paramétere.

I H generikus osztály példányosításakor nem lehet másik generikus osztály a paraméter.

I H ha egy szál véget ért, nem lehet újraindítani.

I H primitív típus tömbje is a primitív típusok közé számít.

I H szerializálás körkörös hivatkozású adatszerkezeten (pl. gyűrű) kivételt dob.

I H minden objektumnak van *wait()* metódusa.

I H nincs olyan várakozó szál, amelyik egyből RUNNABLE állapotú lesz a *notifyAll()* hatására.

2014.06.17 – 5. Feladat

Jellemezze az alábbi Java nyelvre vonatkozó állításokat a kulcs felhasználásával! (6 pont)

[C] A String osztályból nem lehet leszármazottat készíteni, mert a String osztály immutábilis.

[E] A *finalize()* metódust csak egyszer szabad meghívni, mert a hívás hatására a garbage collector felszabadítja az objektumot.

[E] Egy objektum nem szinkronizálhat saját magára, mert az holtponthoz vezetne.

[E] Egy szál futását nem lehet a saját *wait()* metódusával felfüggeszteni, mert egy szál futásának felfüggesztéséhez a *sleep()* függvényt kell használni.

[B] A RuntimeException-ből származó kivételeket kötelező elkapni, mert a RuntimeException az Exception leszármazottja.

[D] Egy interfész nem tartalmazhat privát láthatóságú metódusokat, mert egy interfész minden függvénye absztrakt és publikus.

2015.01.06 – 1. Feladat

Jelölje (karikázza be) az állítások igazságtartalmát, ha feltesszük, hogy szabványos Java nyelvet használunk! (8 pont)

I H a primitív típusokhoz tartozó csomagoló osztályok (wrapper classes) nem változtathatók (immutable).

I H egy szál egyszerre csak egy objektum monitorában tartózkodhat.

I H *synchronized* blokkok nem ágyazhatók egymásba.

I H szálak nem képesek saját magukat közvetlenül *waiting* állapotból notify-jal felébreszteni.

I H előfordulhat, hogy két szál (T1 és T2) ugyanazon objektum ugyanazon *synchronized* metódusát futtatva T1 T2 sorrendben lép be, de T2 T1 sorrendben lép ki.

I H egy változó statikus típusa nem lehet a változó dinamikus típusának leszármazottja.

I H egy osztály lehet akkor is absztrakt, ha nincs absztrakt metódusa.

I H két interfész csak akkor valósítható meg egy osztályban, ha az interfészeknek nincsen közös metódusa.

asd

asd