

Gyakori programozói hibák

Dolgozz keveset, nem ér baleset - tartja a mondás, s igaz ez a Java programozókra is, hiszen ha dolgozunk, közben hibákat is vétünk, de nem mindegy milyen hibákat. [David Reilly](#) csokorba szedte a leggyakoribb programozói hibákat, lássuk a Top10 listát.

10. Nem statikus tagot akarunk elérni statikus metódusból

A Java programozásban eleinte furcsa lehet a statikus és a nem statikus (példányosított) változók és metódusok megkülönböztetése. A statikusnak jelölt részek nem igényelnek példányosítást, az összes többi helyen a deklarált változók csak példányosítás után érhetk el, vagyis bellük tetszleges számú példány létezhet. Mivel a programozási példák ersen építenek a statikus main metódusra, a kezdket összezavarhatja, hogy mit lehet elérni a main metódusból és mit nem, az alábbi kód nem fog fordulni, hiszen a valami nev változó nem statikus változó:

```
public class StaticDemo
{
    public String valami = "valami cucc";

    public static void main (String args[])
    {
        System.out.println ("Írjunk ki a valami tartalmát: " + valami );
    }
}
```

9. Öröklésnél elírjuk a felüldefiniálandó metódus nevét

Az öröklés az objektum orientált programozás lételeme, mondhatni áldás, de átok is egyben, ha a felüldefiniálandó metódus nevét elírjuk akár egy betvel is. Ez utóbbi esetben ugyanis a leszármazott osztályban nem a *látszólag* meghívott metódus fog lefutni, hanem az osztály megfelel nev metódusa. A hibát éveken keresztül programozók ezrei ejtették mondhatni napi rutinnal, de az ötös verzió óta az `@Override` annotációval végre kivédhetünk, hiszen ezzel az annotációval jelezzük a fordító felé, hogy valamelyik s metódusát felül szeretnénk definiálni, s ha nincs egyik osztályban se ilyen, akkor a kód nem fog fordulni.

8. Összehasonlítás = jellel

Sok egyéb nyelvben az összehasonlítás az egyenlőség jellel történik - ahogy azt a matematika is teszi - viszont sok programozási nyelv esetén az egyenlőség jel a *legyen egyenl* utasítást takarja. A C nyelvhez sokkal jobban hasonlító nyelvek különösebb lelkifurdalás nélkül elfogadják egy feltételben az értékadást és az eredménybl igaz/hamis értéket képezve döntik el, hogy a program végrehajtása melyik ágon folytatódjon. Szerencsére a Java ezt nem teszi lehetővé, és az ersen típusosság is kivédi a hibalehetségeket, így ezt a hibát csak a zöldfűl kezdük vétik a Java karrierjük els pár napján.

7. Példányok összehasonlítása == jellel

A Java nyelven kívül szinte minden egyéb nyelvben két *akármi* között egyenlséget tudunk vonni, és az igaz/hamis eredményt fel tudjuk használni. Java bármely két példány között elfogadott az `==` jel, de nem a példány értékét hasonlítja össze, hanem a referenciát - vagyis hogy a két példány valóban egy és azonos-e. Az alábbi programrészlet szerint mind a két változónak az értéke "23" lesz, mégis két külön példány a memóriában, ezért példány szerint (`==`) nem lesznek egyenlk, de érték szerint (`.equals`) igen.

```
String a="23";
String b="2";
b=b+"3";
System.out.println(a==b);
System.out.println(a.equals(b));
```

6. Érték szerint vagy referencia szerint?

A Java nyelv elrejtí a memóriát a programozó ell, a személygyjt szálla hagyva a memória tisztán tartását. Ha változókat használunk, tudnunk kell mit jelent a referencia szerinti és érték szerinti átadás, hiszen ez nagymértékben befolyásolja a programjaink hibátlan mködését: a primitív típusok (és azok befoglaló osztályai) érték szerint adódnak át, minden más referencia szerint. Ez azt jelenti, hogy ha átadunk egy vektort egy metódusnak, akkor a metódusok belül hozzá tudunk adni új elemet, vagy törölni tudjuk annak tartalmát:

```

public static void add(Vector vector)
{
    vector.add("0");
}

public static void create(Vector vector)
{
    vector = new Vector();
}

public static void main(String[] args)
{
    Vector vector=new Vector();
    add(vector);
    System.out.println(vector.toString());
    create(vector);
    System.out.println(vector.toString());
}

```

A várt eredmény, hogy a második kiírásnál üres vektort kapunk - elmarad, mivel a create metódusban nem a referencián végzünk mveletet, hanem új értéket adunk neki, ezáltal a metódusok belül minden mvelet egy új vektor példányra fog végrehajtódni, egyik mvelet sem befolyásolja a paraméterben kapott változót. Gyakori hiba még haladóbb programozók között is.

5. Üresen hagyott catch ág

A struktúrált programozás leghasznosabb találmánya a try-catch blokk, amely igen hatékony hibakezelést tesz lehetővé. Sokszor találkozni olyan try-catch programrészekkel, amelyben a catch ágban nem történik semmi:

```

try
{
    // ...
}
catch (Exception except)
{
}

```

Bármilyen hiba is keletkezik a try blokkon belül, a catch ág elkapja, majd *jól nem csinál semmit*, a program fut tovább, a programozó pedig a felhasználóra fogja a problémát, hiszen nincs semmi hiba se a konzolon, se a naplóban...

4. Minden index nullával kezdődik...

Sok nyelvben a halmazok vagy listák els eleme az els (1) indexet viseli, sok másik nyelvben pedig a nulladik (0) indexet; a Java ez utóbbi nyelvek közé tartozik. Szerencsére túlcímzés esetén futás közben kivételt kapunk, hogy a tömb nem tartalmaz annyi elemet, ahányadikat használni szeretnénk. Persze ha üres a catch blokk... akkor nem kapunk kivételt... :)

Ha végre megszokjuk a nullával kezdő indexelést, akkor a JDBC ezt a tudást porig rombolja, ugyanis itt minden sorszám egyel kezdődik... a dátumok esetén pedig - teljesen logikusan - a hónapok nullával kezdődnek, a január a 0. hónap, a napok viszont egyessel kezdődnek.

3. Több szálon futás

Még több éves tapasztalatok után is képes az ember hibákat hagyni egy több szálon futó programban, amikor szinkronizálás nélkül használ egy-egy változót - amely általában nem szálbiztos. Persze tesztelni egy szálon tesztel, és a kész rendszer a terhelés növekedtével egyre több rejtélyes hibát okoz...

2. Kisbet-nagybet probléma

Jó pár friss Java programozóknak okoz gondot a Java nyelv metódus és változó elnevezési konvenciója. Minden változót kisbetűvel írunk - az els szót követően minden szó els betje nagybetű, ellenben minden konstans nagybetűvel deklarálunk. Ehhez jön még a getter/setter minta, amikor a változók nevéhez ragasztjuk a get/set szót és ilyen néven szerepeltetjük ezeket a metódusokat. A kezdő programozók pár osztály után, az önfejű programozók sok-sok osztály után döbbennek rá, hogy ezen az elnevezési módszeren egy csomó Java technológia alapul...

1. A null pointer

Minden hibák legalattomosabbika, még a legtapasztaltabb öreg rókák életét is megkeseríti, ha egy változót nem példányosít, majd megpróbálja a példányváltozókat használni. Természetesen nem lehet a nyelvben kivenni a null értéket, hiszen egy csomó metódus ad vissza null értéket - akár üzemszerűen, akár hibát jelezve: sajnos ezzel együtt kell élni. Mint minden nyelvben - a Java nyelvben is vannak érdekes mellékhatások, íme a legrövidebb Java program, amely egyszerűen csak hibásan működik, pedig hiba nélkül lefordítható:

```
public class Main
{
    public static void main(String[] args)
    {
        throw null;
    }
}
```

Vélemény? 😊