VIII. Szálak és animáció

1. Bevezetés

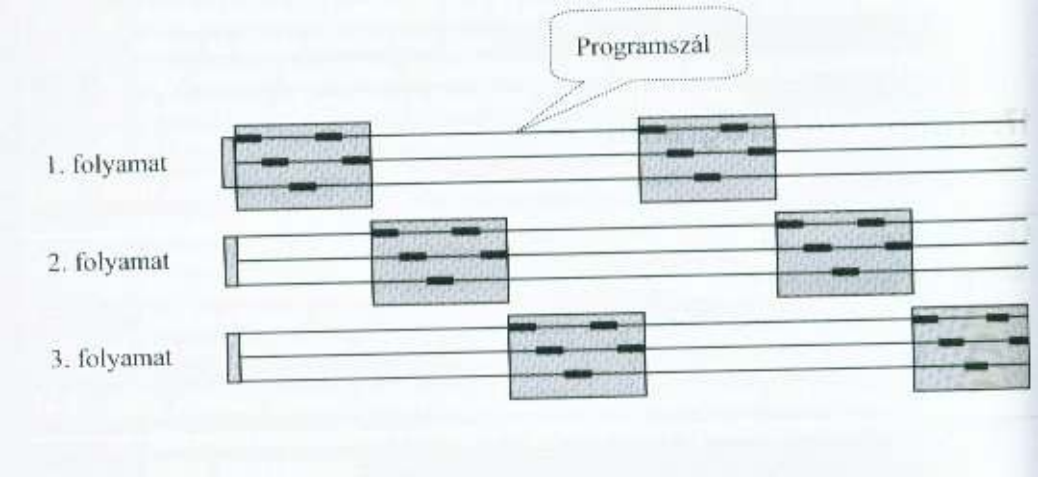
A mai korszerű operációs rendszerek multiuser-multitask rendszerek. Tehát az operációs rendszer egyszerre több feladattal is foglalkozik. Gondoljunk csak arra, hogy amikor elkezdünk egy cd grabbelést akkor mellette még lehetőségünk van internetezni…stb. Ez persze nem azt jelenti, hogy a processzor egyszerre több műveletet el tud végezni, pusztán annyit jelent, hogy több feladattal foglalkozik úgy, hogy bizonyos stratégia szerint minden feladattal foglalkozik egy kicsit. Ezt a stratégiát nevezzük ütemezésnek, ami sok paramétertől függhet (pl.: prioritás), de ezzel most nem kívánunk foglalkozni. Az ütemezés mondja meg, hogy a párhuzamosan futó folyamatok közül éppen melyik kapja meg a vezérlést. Természetesen ez a körforgás nagyon gyorsan megy, így a kívülálló felhasználónak úgy tűnik, hogy a programjai párhuzamosan, egyszerre működnek.

Programjaink írásakor is lehetőségünk van a fent említett elvek használatára. Ezt a programozási technikát nevezzük ***többszálú programozásnak***.

2. Többszálú programozás Java-ban

2.1. A programszál fogalma

A programszál (thread) hasonlít az operációs rendszerben futó folyamatra: ő is csak időnként kapja meg a vezérlést. Az alábbi ábrán az is látható, hogy a folyamatok szálakból állnak. Amikor egy folyamat fut, akkor az ő szálai között oszlik meg a vezérlés.

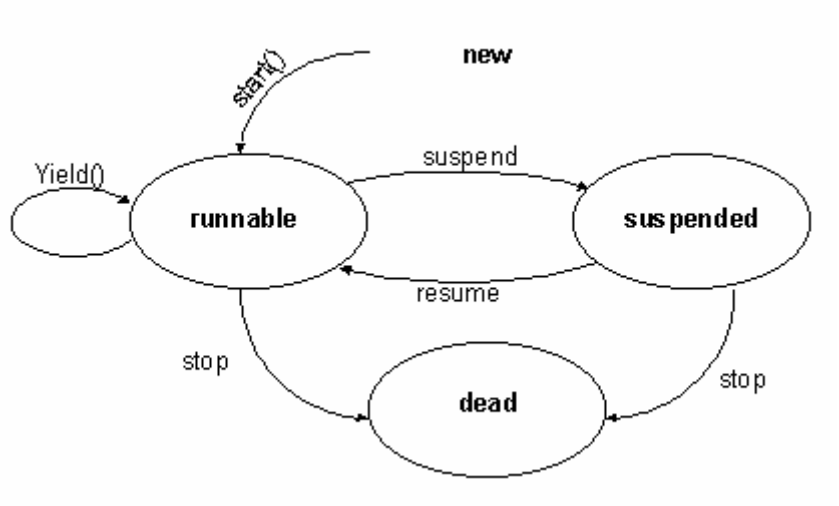


A programszál valójában egy utasítássorozat, amely valahol elkezdődik, és valahol befejeződik. Az ütemezést az ún. szálütemező végzi.

2.2. A programszál állapotai

A programszál megszületik, működik és meghal. Születésétől a haláláig a következő állapotokba juthat:

- **Új** (new) : a szál megszületésekor ebbe az állapotba kerül amíg el nem indítják
- **Futtatható** (runnable) : A szál akkor futtatható, ha elindították és még él. Alállapotai a következők:
  - ***Futó*** (run) : a szál aktív, legfeljebb CPU-ra vár
  - ***Blokkolt*** (suspended) : ha valami akadályozza a szál futását vagy
    - *Alvó* (sleep) : a szálat el lehet altatni valamennyi időre. Ha ez az időtartam letelik, akkor felébred és fut tovább
    - *Várakozó* (wait): A szál addig vár, amíg fel nem ébresztik (notify). Ha jön az ébresztés, akkor a programszál tovább fut.
- **Halott** (dead) : egy szál halott, ha befejezte futását (a run végetért) vagy meghívtuk a stop() metódust



2.3. Java programszál létrehozása

A Java-ban a programszál is egy objektum; osztálya a Thread osztály leszármazottja. A Thread osztály a java.lang csomag része, ezért importálásáról nem kell külön gondoskodnunk.

Szálakat kétféle módon hozhatunk létre:

- (1) a Thread osztály kiterjesztésével
- (2) vagy a Runnable interfész implementálásával *(tudjuk, hogy Java-ban többszörös öröklődés nem lehetséges. Ha appletekben szeretnénk szálakat használni, akkor a Thread osztály kiterjesztése nem*

járható őt, mivel egy applet az Applet osztály kiterjesztésével jön létre. Ekkor használható a Runnable interfész implementálása)

(1) Ha a Thread osztály kiterjesztésével létrehozott osztályunkat példányosítjuk, akkor kapunk egy szálat, de ez még nem fog futni. A futtatáshoz meg kell hívnunk a `Thread` osztályból örökölt **`start()`** metódust. Ez inicializálja a szálat és elindítja annak **`run()`** metódusát. A `run()` metódus a szál főprogramja, itt kap helyet az összes olyan művelet, amit a szálnak el kell végeznie.

Tehát összefoglalva azt mondhatjuk, hogy szál programozásakor át kell írnunk a `run()` metódust (az alapértelmezett run() metódus nem csinál semmit).

(2) A Runnable interfész egyetlen metódust definiál a `run()` metódust. Ebben az esetben ennek a metódusnak az implementálása a feladatunk.

2.4. Példa programszálra

Hozzunk létre 2 szálat és írjuk ki, hogy éppen melyik szál működik. Figyeljük meg a két megoldás közötti különbségeket.

2.4.1. Megoldás az (1) módszerrel

```
class Szal extends Thread
{
  public void run ()
  {
    System.out.println("Ez a "+getName()+" szal");
  }
}

public class Proba
{
   public static void main (String args[])
   {
    Szal a, b;
    a = new Szal ();
    b = new Szal ();
    a.start();
    b.start();
  }
}
```

2.4.2. Megoldás az (2) módszerrel

```
class Szal implements Runnable
{
  public void run ()
  {
    System.out.println("Ez a "+Thread.currentThread().getName()+" szal");
  }
}
public class Proba
{
   public static void main (String args[])
   {
    Szal programszal=new Szal();
    Thread a = new Thread (programszal);
    Thread b = new Thread (programszal);
    a.start();
    b.start();
  }
}
```

2.5. A Thread osztály további fontos metódusai

2.5.1. `start()`

Elindítja a programszálat. Meghívja a `run()` metódust.

2.5.2. `sleep()`

A sleep() metódus segítségével egy szálat el lehet altatni a paraméterként megadott ezredmásodpercig. Például a `Thread.sleep(1000)` 1 másodpercre elaltatja a szálat.

2.5.3. `isAlive()`

Visszaadja, hogy a szál élő-e. Egy szál akkor élő, ha elindították és még nem halott.

2.5.4. `currentThread()`

Visszaadja az éppen futó szálobjektumot.

2.5.5. `getName()`

Visszaadja az éppen futó szálobjektum nevét.

2.6. Szálak szinkronizációja

Egy objektumhoz több programszál is hozzáférhet. Ilyenkor szükség lehet arra, hogy az egyik programszál bevárja a másikat. Erre szolgálnak a **wait** és **notify** metódusok. Az a szál, amelyik meghívja a wait metódust, blokkolódik, és addig vár, amíg egy bizonyos feltétel nem teljesül. Amikor jelzést kap, hogy újra futhat, megpróbálja folytatni a futást.

A notify metódus szolgál arra, hogy egy várakozó szálat tovább engedjünk. Amikor a várakozó szál megkapja az üzenetet, megpróbál továbbhaladni. Néha szükség lehet arra, hogy az összes várakozó szál feléledjen. Erre szolgál a **notifyAll** metódus.

3. Animáció

Appletek esetén az animáció a paint() és a repaint() metódusok hívásával valósítható meg. A szálak pedig hasznos segítőtársak lehetnek az animációk megírásában. Az animációk esetén fontos szerepet kap a sleep() metódus is, ugyanis az emberi szemnek felfogható sebességgel szeretnénk megjeleníteni a grafikáinkat.