

Bevitel

Fura lenne a programunk, ha nem tudnánk megkérni a felhasználót arra, hogy vigyen be valamilyen értéket.

Az egyszerűség kedvéért most a „konzolról” (*console*) fogunk adatokat beolvasni. A *konzol* a billentyűzetet és a szöveges képernyőt jelenti. Ez azt jelenti, hogy a NetBeansben oda kattintunk, ahova eddig a kimenetet írta a programunk, és ott gépelünk be valamilyen adatot a programunk számára a billentyűzet segítségével.

Processz, be- és kimenetei

Ha az operációs rendszerben (pl. Windows) elindítasz egy programot, akkor egy *processz* (*process*) keletkezik. A futó programnak ez a neve.

Amit itt írok, az első sorban a grafikus felülettel nem rendelkező programokra vonatkozik, bár rájuk is érvényes, de nem annyira lényeges.

Minden processz a külvilággal háromféle csatornán beszélget, ezek neve *szabványos bemenet* (*standard input, stdin*), *szabványos kimenet* (*standard output, stdout*) és *szabványos hibakimenet* (*standard error, stderr*). Alapértelmezés szerint a billentyűzet a szabványos bemenetre van „kötve”, a képernyő pedig a szabványos kimenet és a szabványos hibakimenet végcélja.

Az operációs rendszerek lehetőséget biztosítanak arra, hogy ezeket a csatornákat „átirányítsuk” (redirect), azaz a szabványos bemenetre egy fájl tartalmát „kössük”, vagy a szabványos kimenetet vagy a szabványos hibakimenetet egy fájlba irányítsuk át.

```
program.exe <bemenet.txt >kimenet.txt 2>hiba.txt
```

Grafikus programok

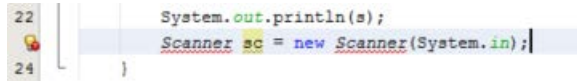
Kérdezhetnéd, hogy mikor fogunk majd grafikus felületű programot írni. A válasz, hogy sokkal később, addig nagyon sok alapvető koncepciót át kell rágnunk magunkat, különben nem értenéd, amit írok. (Ez tuti. Kipróbáltam, tényleg nem megy! Türelem grafikus felületet terem!)

Scanner létrehozása

A konzolról való beolvasáshoz először egy ún. *Scanner*-t hozunk létre. (A nem teljesen kezdőknek: ez egy osztály, de ez most teljesen lényegtelen).

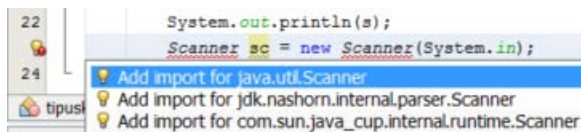
```
Scanner sc = new Scanner(System.in);
```

Ha ezt beírjuk, a NetBeans egy szép, piros hullámvonallal jutalmaz bennünket, mely a `Scanner` szavak alatt található, valahogy így:



Az előző anyagból kifolyólag már tudhatod is a választ (hiányzik az `import`), de azért nézzük meg egy kicsit a NetBeans-t.

Kattintsunk a kis lámpácska ikonra a képernyő bal szélén:



Válasszuk a menüből a felsőt: „Add import for `java.util.Scanner`”.

Ennek hatására a program elejére bekerül az `import java.util.Scanner;` sor.

Emlékeztetőül: A `Scanner` a `java.util` csomagban található, és ahhoz, hogy rövid névvel használni tudjuk, *be kell importálni*.

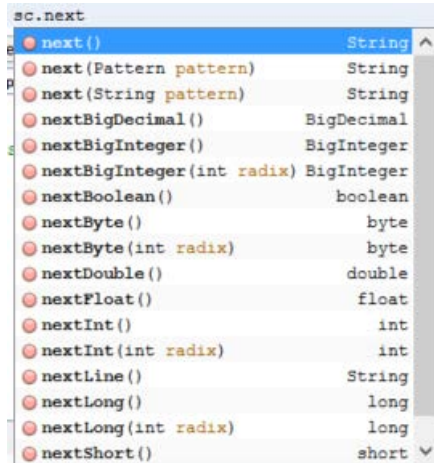
Fontos! Programonként maximum egy Scannert hozzunk létre!

Scanner használata

Mielőtt a billentyűzetről olvasunk, írjuk ki a felhasználónak, mit szeretnénk tőle beolvasni!

```
int szam = sc.nextInt();  
double tort = sc.nextDouble();  
float f = sc.nextFloat();
```

... (a többi ki tudod találni, de a NetBeans is segít. Ha beírjuk, hogy ``sc.next``, akkor kapunk egy listát a ``Scanner`` a ``next`` szóval kezdődő szolgáltatásairól (metódusai), és választhatunk közülük).



Scanner használata Stringek beolvasására

```
String szo = sc.next();
String sor = sc.nextLine();
```

Ha egy szót akarunk beolvasni, akkor az `sc.next()` teljesen jó. Ha egy egész sort, akkor az `sc.nextLine()` a megfelelő.

A Scanner és a *pufferelt bemenet*

A programok, amikor a konzolról olvasnak (olvassák azt, amit a felhasználók beírtak), akkor az a következőképpen működik:

Ha akár csak egyetlen karakterre is szükségük van (a számok is karakterekből állnak), akkor megkérlik az operációs rendszert, hogy adjon nekik egy vagy több karaktert. Az operációs rendszer pedig – fejlett program lévén – nem egyesével adja a karaktereket a programnak, hanem – hogy a felhasználó menet közben szerkeszteni, javítani tudja a beírt dolgokat – soronként. A program csak akkor szerez tudomást a bemenetről, ha a felhasználó már lenyomta az ENTER-t.

Az ENTER lenyomásakor az egész bemenet egy átmeneti tárolóba kerül (ún. bemeneti pufferbe (*input buffer*)), és onnan a program annyi karaktert kérhet, amennyire szüksége van. Ha a program több karaktert olvasna, mint amennyi rendelkezésre áll, akkor az operációs rendszer egy újabb sort olvas a felhasználótól, hogy el tudja látni a programot adatokkal.

Pl. ha kérünk egy számot az `sc.nextInt()`-tel, az a következőképpen hajtódik végre:

1. Mivel nincs a bemeneti pufferben adat (a `Scanner` létrehozásakor sosincs) az operációs rendszer beolvas egy sort a felhasználtól.
2. A felhasználó begépel pl. a `12[szóköz]23[ENTER]` jeleket.
3. Az `sc.nextInt()` a bemeneti pufferből (átugorva az esetleges elválasztó karaktereket (white space-ek: szóköz, tab, ENTER)) elkéri az 1-et és a 2-t, a `[szóköz]`-t viszont már otthagyja, számmá alakítja és beteszi a megfelelő változóba. (Maradt: `[szóköz]23[ENTER]`).
4. A következő `sc.nextInt()` hívás folytatja az olvasást a pufferből, de nem kéri az operációs rendszert újabb bevitel olvasására – eldobja a `[szóköz]`-t, majd feldolgozza a 23-at és benthagyja az ENTER-t. Maradék: `[ENTER]`.
5. Tegyük fel, hogy ezek után a programunk egy sort akar olvasni: `sc.nextLine()`. Az `sc.nextLine()` úgy működik, hogy minden karaktert beolvas, és betesz a `String`-be, amíg ENTER-rel nem találkozik. Az ENTER-t is eltávolítja, de nem teszi bele a `String`-be. Ha ezek után az `sc.nextLine()` lefut, akkor azonnal találkozik az olvasás végét jelentő ENTER-rel, berakja az addig olvasott karaktereket (a semmit) a `String`-be, eltávolítja az ENTER-t és befejezi az olvasást.

Mi ebből a tanulság?

Ha szám után sort olvasunk, akkor a szám olvasása után ürítsük a puffert egy önálló `sc.nextLine()` olvasással, így:

```
System.out.println("Írjon be egy számot!");  
  
int szam = sc.nextInt();  
  
sc.nextLine(); // üríti a puffert  
  
System.out.println("Írjon be egy sor szöveget!");  
  
String s = sc.nextLine();
```