A Java nyelv alapjai 5.
Literálok

- **Objektumok**: **`null`**. Bármely objektumreferenciának értékül adható
- **Logikai értékek**: **`true`**, **`false`**
- **Egész számok**:
  - 0 kezdetű számok oktális (8-as számrendszer), pl. **`0377`**
  - 0x kezdetű számok hexadecimális (16-os számrendszer), pl. **`0xff`**
  - minden más szám decimális számrendszerben lesz értelmezve, pl. **`255`**
  - egy egész szám után írt **`l`** vagy **`L`** betű long típust deklarál
- **Lebegőpontos számok**:
  - decimális számként adhatók meg tizedesponttal és **`e`** vagy **`E`** exponenssel
  - ez önmagában double típust jelent, `f` vagy `F` végződéssel lehet float típust deklarálni, pl. **`3.14, -123.45, 12.3e4`**
- **Karakterek**: aposztrófok között kell megadni, pl. **`'a', 'Z'`**
- **Szövegek (String)**: idézőjelek között kell megadni, pl. “**`Helló Világ`**”

Forrásprogram szerkezete

- A forrásállomány (fordítási egység) egy vagy több osztályból állhat.
- A forrásállomány neve a **`main`** metódust tartalmazó egyetlen publikus osztály neve lesz.
- Fordításkor több class kiterjesztésű állo-mány keletkezhet.

```
import java.utils.*;
import extra.*;
//Egy vagy több osztály deklarálása, a sorrend mindegy

class C1{
  ...
}

class C2{
  ...
}
...
public class Prog{
  ...
  public static void main(String[] args){
    ...
  }
  ...
}
```

Operátorok

- Unáris postfix és prefix operátorok
  - `[ ]` tömbképző
  - `.` minősítő
  - `( )` metódus képző
  - `~, !` ld. később
  - `new` példányosító
  - `(`*`típus`*`)`*`kifejezés`* típuskényszerítő
  - `+, -` előjel
  - `++, --` léptető, pl. i++ vagy ++i mindkettő növeli i értékét, de az első értéke i eredeti, míg a második i megnövelt értéke lesz.

++, --

```
int a = 0, b, c;
b = ++a;
a = 0;
c = a++;

c = ++b + a++;
c = a+++++b;
c = a++ + ++b;
```

Multiplikatív operátorok

- `*, /`
- `%` maradékos osztás
- Ha az operandusok egészek, akkor az eredmény is egész, ha legalább az egyik operandus valós, akkor az eredmény is valós.
- **Pl.**

```
int ossz = 2 + 3;
double atlag = ossz / 2;
```

- Additív operátorok

  `+, -`
- Relációs operátorok

  `<, <=, >, >=, ==, !=`
- Logikai operátorok
  - `!` nem
  - `&, &&` és
  - `|, ||` vagy
  - `^` kizáró vagy
- Bitenkénti operátorok
  - `~` komplementálás
  - `&, |, ^`
  - (`<<, >>, >>>` léptetések)

- Feltételes operátorok

  `(`*`feltétel`*`) ? `*`kifejezés1`*`: `*`kifejezés2`*

  pl. `kamat = (fiz>200000)?10:0;`

- Értékadó operátorok

  `=, +=, -=, *=, /=, %= …`

Az összetett értékadás szintaktikája:

változó operátor = kifejezés

`a += b`

szemantikája:

változó = (*típus*) *változó operátor kifejezés*

`a = a + b`

ahol *típus* a változó típusa, amelyet rákényszerí-tünk a jobboldalra.

Értékadó operátorok

- Vigyázzunk az összetett ill. a „hagyományos értékadó utasítások nem minden esetben teljesen egyformák!

  ```
  int a=10;
  a+=5;
  ```

  megegyezik az `a=a+5` ut.-al, mindkét oldal int

  ```
  a+=1.5;
  ```

  nem egyezik meg az `a=a+1.5` ut.-al, mely szintaktikai hibás, hiszen a jobb oldal double

- Lehetséges a többszörös értékadás, pl: `a=b=5;`

Kiértékelés

- Egy kifejezés kiértékelési sorrendjét meg-határozzák:
  - zárójel
  - prioritás
  - asszociativitás (balról – jobbra vagy jobbról – balra szabály)

Java utasítások

- deklaráló; pl. `int a;`
- értékadó; pl. `a = b * 2;`
- postfix és prefix növelő és csökkentő; pl. `a++;`
- metódushívás, pl.
  `System.out.println("Hahó!");`
- példányosítás, pl.
  `auto = new Auto("ABC123");`
- Programvezérlő, pl. elágazások, ciklusok
- üres: `;`

Java utasítások

- Minden utasítást pontosvessző zár le.
- Blokk (vagy összetett utasítás): `{ ... }`

Értékadó utasítás, típuskonverziók

- *`változó = kifejezés;`*
- A kifejezés típusának értékadás szerint kompatíbilisnek kell lenni a változó típusával:
  - azonos típusok
  - a jobb oldal szűkebb, akkor implicit bővítő konverzió
  - a bal oldal `byte, short` vagy `char`, a jobb oldal `int`, és a fordító el tudja dönteni, hogy a jobboldal belefér a baloldalba, akkor implicit szűkítő konverzió.
    pl. `byte b = 100;` (az egész literál automatikusan `int` típusú).
  - minden más esetben fordítási hiba keletkezik

Értékadó utasítás, típuskonverziók

```
int i;
double d;
d = i;   //implicit bővítő konverzió
i = d;   //szintaktikai hiba
i = (int)d; //explicit szűkítő konv.
```

Metódushívás

- *Osztály*.*metódus*(*paraméterek*)
  pl. `y=Math.sin(x);`
- *Objektum*.*metódus*(*paraméterek*)
  pl. `hossz=szoveg.length();`
- Saját osztályból elég csak a metódus neve:
  metódus(*paraméterek*)

- Egy metódus lehet eljárás- vagy függvényszerű.
- Túlterhelés (overloading): lehet több azonos nevű metódus, melyek a paraméterezésben és/vagy a visszatérési érték típusában térhetnek el egymástól.

Pl. `float max(float a, float b)` illetve
`int max(int a, int b)`

Szelekciók - `if` utasítás

```
if(feltétel)
  utasítás1;
else
  utasítás2;
```

```
if(a>b)
  c=a;
else
  c=b;
```

- *feltétel*: logikai kifejezés
- az `else` ág elhagyható
- a feltétel után nincs pontosvessző
- az utasítás esetén viszont van pontosvessző
- minden feltétel zárójelben
- egy ágban több utasítás: blokk {...}
- egymásba ágyazás

Szelekciók - **`switch`** utasítás

```
switch(kifejezés){
   case érték1: utasítások;
                break;
   case érték2: utasítások;
                break;
   ...
   default:     utasítások;
}
```

- akkor alkalmazható, ha egy kifejezés jól meghatározott, különálló értékeire szeretnénk bizonyos utasításokat végrehajtani
- *`kifejezés`*: byte, short, int vagy char
- a `break` hatására a `switch` blokk végére kerül a vezérlés, e nélkül a következő `case` ágra kerülne a vezérlés
- egy `case` kulcsszóhoz csak egy érték tartozhat

JAVA program 2 ☺

```
public class Kiiras1 {
	public static void main(String[] args) {
		System.out.println(2.8/3*(1+3));
	}
}
```

```
public class Kiiras2 {
	public static void main(String[] args) {
		System.out.println("2.8/3*(1+3) erteke: "+2.8/3*(1+3)+".");
	}
}
```

Elágazás

```
public class If1 {

	private int szam;

	public If1(int szam_)
	{
		szam=szam_;
	}
	public int getSzam()
	{
		return szam;
	}
	public String getif1()
	{
		if (szam<0)
			return(" negativ");
		else
			return(" pozitiv");
	}
	public static void main(String[] args)
	{
		If1 ha = new If1(2);
		System.out.println("A "+ha.getSzam()+ha.getif1());
	}
}
```

```
public class If2 {

	private int szam;

	public If2(int szam_)
	{
		szam=szam_;
	}
	public int getSzam()
	{
		return szam;
	}
	public String getif2()
	{
		if (szam<0)
			return (" negativ");
		else
			if (szam==0)
				return (" nulla");
			else
				return (" pozitiv");
	}
	public static void main(String[] args)
	{
		If2 ha = new If2(2);
		System.out.println("A "+ha.getSzam()+ha.getif2());
	}
}
```

```
public class Kor {
		private double sugar;
		public double getSugar()
		{
			return sugar;
		}
		public Kor(double sugar_)
		{
			sugar=sugar_;
		}
		public double kerulet()
		{
			return 2*sugar*Math.PI;
		}
		public double terulet()
		{
			return sugar*sugar*Math.PI;
		}
		public static void main(String[] args)
		{
			Kor k = new Kor(1);

			System.out.println(k.getSugar()+" egyseg sugaru kor");
			System.out.println(" kerulete: "+k.kerulet());
			System.out.println(" terulete: "+k.terulet());
		}
}
```