Tema: Java Operators

Tabela e përmbajtjes

Lista e figurave

Përshëndetje të gjithëve!

Në këtë artikull, do të mësoni për llojet e ndryshme të operatorëve në Java, sintaksën e tyre dhe si t'i përdorni ato me ndihmën e shembujve. Operatori në Java është një simbol që përdoret për të kryer veprime në variabla dhe vlera. Për shembull: operatori + përdoret për mbledhje.

Disa tipe operatorësh janë:

- Arithmetic Operators
- Assignment Operator
- Unary Operators
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise & Shift Operators
- instanceof Operator

1. Arithmetic Operators

Përdoren për të kryer veprime të thjeshta aritmetike si në algjebër. Më poshtë jepen operatorët dhe veprimi që kryejnë.

Operatori	Veprimi
+	Mbledhje
-	Zbritje
*	Shumëzim
/	Pjesëtim
%	Modulo (mbetja pas pjesëtimit)

Shohim një shembull për përdorimin e këtyre operatorëve. Në klasën ArithmeticOpTest kemi deklaruar në metodën main dy variabla të tipi tint (a dhe b). Më pas kemi printuar vlerat e secilit veprim mbledhje, zbritje, shumëzim, pjesëtim dhe modulo, si në figurën 1.

```java
public class ArithmeticOpTest {
    public static void main(String[] args) {
        // deklarojmë variablat
        int a = 17, b = 7;

        // operatori i mbledhjes
        System.out.println("a + b = " + (a + b));

        // operatori i zbritjes
        System.out.println("a - b = " + (a - b));

        // operatori i shumezimit
        System.out.println("a * b = " + (a * b));

        // operatori i pjesetimit
        System.out.println("a / b = " + (a / b));

        // operatori modulo
        System.out.println("a % b = " + (a % b));
    }
}
```

Fig. 1 Arithmetic operators example

Rezultati

```
a + b = 24
a - b = 10
a * b = 119
a / b = 2
a % b = 3
```

Nëse përdorim operatorin e pjesëtimit (/) me dy numra të plotë, atëherë rezultati që marrim do të jetë gjithashtu një numër i plotë. Nëse një nga termat është një numër me presje dhjetore, rezultati do të jetë gjithashtu me presje dhjetore.

Marrim shembullin si më poshtë:

```java
public class DivisionOpTest {
    public static void main(String[] args) {
        int a = 10;
        int b = 7;
        double c = 10.3;
        double d = 3.5;

        System.out.println("a / b = " + (a / b));
        System.out.println("c / a = " + (c / a));
        System.out.println("a / d = " + (a / d));
        System.out.println("c / d = " + (c / d));
    }
}
```

Fig. 2 Division operator example

Rezultati

```
a / b = 1
c / a = 1.03
a / d = 2.857142857142857
c / d = 2.942857142857143
```

2. Assignment Operators

Assignment Operator përdoret për t'i caktuar një vlerë çdo variabli. Vlera e dhënë në anën e djathtë të operatorit i caktohet variablit në të majtë. Sintaksa: **variabla = vlera;**

Në disa raste *Assignment Operator* mund të kombinohet me operatorë të tjerë për të krijuar një version më të shkurtër të *Compound Statement*. Për shembull, në vend të **a = a + 9**, mund të shkruajmë **a += 9**.

Operator	Veprimi	Ekuivalente me
=	Vlera e dhënë në anën e djathtë të operatorit i caktohet variablit në të majtë.	a = b;
+=	Për mbledhjen e variablit në të majtë me atë në të djathtë dhe pastaj caktimin e tij në variablin në të majtë	a = a + b;
-=	Për zbritjen e variablit në të majtë me atë në të djathtë dhe pastaj caktimin e tij në variablin në të majtë	a = a - b;
*=	Për shumëzimin e variablit në të majtë me atë në të djathtë dhe pastaj caktimin e tij në variablin në të majtë	a = a * b;
/=	Për pjesëtimin e variablit në të majtë me atë në të djathtë dhe pastaj caktimin e tij në variablin në të majtë	a = a / b;
%=	Për të gjetur mbetjen e variablit në të majtë me atë në të djathtë dhe pastaj caktimin e tij në variablin në të majtë	a = a % b;

3. Unary Operators

Unary Operators përdoren vetëm me një operand. Ato përdoren për të rritur, zvogëluar ose mohuar një vlerë. Për shembull, ++ është një *Unary Operator* që rrit vlerën e një variabli me 1. Domethënë, ++8 do të kthejë 9.

Llojet e ndryshme të *Unary Operators* janë:

Operatori	Shpjegimi
+	**Unary plus:** nuk është e nevojshme të përdoret pasi numrat janë pozitivë dhe pa e përdorur
-	**Unary minus:** ndryshon shenjën e një shprehjeje

++	**Increment operator:** rrit vlerën me 1
--	**Decrement operator:** zvogëlon vlerën me 1
!	**Logical complement operator:** ndryshon vlerën e një Boolean (nga false -> true ose e kundërta)

Shohim shembullin e mëposhtëm për *Unary Operators*.

```
public class UnaryOpTest {
    public static void main(String[] args) {
        // deklarojme variablat
        int a = 7, b = 7;
        int result1, result2;

        // vlera fillestare e a
        System.out.println("Vlera e variablit a: " + a);
        result1 = ++a;
        System.out.println("Pas inkrementimit: " + result1);

        // vlera fillestare e b
        System.out.println("Vlera e variablit b: " + b);
        result2 = --b;
        System.out.println("Pas dekrementimit: " + result2);

        /** Shembull me operatorin ! */
        boolean boolVariable = false;
        boolean isJava = !boolVariable;

        //Vlera e boolVariable
        System.out.println("Vlera e boolVariable: " + boolVariable);

        //Vlera e isJava
        System.out.println("Vlera e isJava: " + isJava);
    }
}
```

Fig. 3 Unary operators example

Rezultati

```
Vlera e variablit a: 7
Pas inkrementimit: 8
Vlera e variablit b: 7
Pas dekrementimit: 6
Vlera e boolVariable: false
Vlera e isJava: true
```

Në programin më sipër operatorët ++ dhe -- i kemi përdorur si **prefixes (++a, --b)**. Këta operatorë mund t'i përdorim dhe si **postfixes (a++, b--)**. Ka një diferencë të vogël kur i përdorim në secilin rast. Shohim shembullin e mëposhtëm:

```java
public class PrefixPostfixTest {
    public static void main(String[] args) {
        /** Shembull me prefix dhe postfix */
        int a = 7, b = 7;

        System.out.println("Vlera e a: " + a);
        System.out.println("Vlera e ++a: " + ++a);
        System.out.println("Vlera e a++: " + a++);
        System.out.println("Vlera e a: " + a);

        System.out.println("\nVlera e b: " + b);
        System.out.println("Vlera e --b: " + --b);
        System.out.println("Vlera e b--: " + b--);
        System.out.println("Vlera e b: " + b);
    }
}
```

Fig. 4 Prefix & Postfix example

Rezultati

```
Vlera e a: 7
Vlera e ++a: 8
Vlera e a++: 8
Vlera e a: 9

Vlera e b: 7
Vlera e --b: 6
Vlera e b--: 6
Vlera e b: 5
```

Nëse përdorim operatorin ++ si **prefix ++a** vlera e a rritet me 1, pastaj kthen vlerën. Nëse përdorim operatorin ++ si **postfix a ++**, atëherë si fillim kthehet vlera e **variablit a** dhe pastaj rritet me 1. Kjo shihet dhe në shembullin më sipër. Si fillim vendosim vlerat e a dhe b = 7. Printojmë vlerën fillestare të a që është 7. Më pas printojmë vlerën e **++a** që është 8, pra vlera e **a** rritet me 1 dhe pastaj printon vlerën 8. Kur printojmë vlerën e **a++** shohim se përsëri del rezultati 8. Kjo sepse në fillim printohet vlera e **a** pastaj rritet me 1. Nëse printojmë përsëri vlerën e **a** shohim se tani rezultati është 9. E njëjta gjë ndodh dhe me operatorin -- të cilin e shohim në shembullin me **variablin b**.

4. Relational Operators

Relational Operators përdoren për të kontrolluar marrëdhënien midis dy kufizave. Këta operatorë përdoren për të kontrolluar marrëdhëniet si barazia, më e mëdhe se, më e vogël se etj. Ata kthejnë rezultat boolean pas krahasimit dhe përdoren gjerësisht në *loops*, si dhe në marrjen e vendimeve; *if else statements*. Llojet e ndryshme të *Relational Operators* janë:

Operatori	Shpjegimi
==	Equal to (i barabartë me)
!=	Not equl to (jo i barabartë me)
>	Greater than (më i madh se)
<	Less than (më i vogël se)
>=	Greater than or Equal to (më i madh ose i barabartë me)
<=	Less than or Equal to (më i vogël ose i barabartë me)

Shohim shembullin e mëposhtëm. Deklarojmë variablat a me vlerë 8 dhe b me vlerë 15. Në bazë të krahasimeve që kryejmë mes a dhe b marrim rezultatet boolean. a == b rezulton false pasi nuk janë të barabarta. a != b rezulton true, sepse janë të ndryshme. E njëjta logjikë funskionon dhe për operatorët e tjerë.

```java
public class RelationalOpTest {
    public static void main(String[] args) {
        // deklarojme variablat
        int a = 8, b = 15;

        // vlerat e a dhe b
        System.out.println("a eshte: " + a + " dhe b eshte: " + b);

        // operatori ==
        System.out.println(a == b);  // false

        // operatori !=
        System.out.println(a != b);  // true

        // operatori >
        System.out.println(a > b);  // false

        // operatori <
        System.out.println(a < b);  // true

        // operatori >=
        System.out.println(a >= b);  // false

        // operatori <=
        System.out.println(a <= b);  // true
    }
}
```

Fig. 5 Relational operators example

Rezultati

```
a eshte: 8 dhe b eshte: 15
false
true
false
true
false
true
```

5. Logical Operators

Logical Operators përdoren për të kontrolluar nëse një shprehje është ***true*** ose ***false***. Ato përdoren në vendimmarrje. Llojet e ndryshme të *Logical Operators* janë:

Operatori	Shembull	Shpjegimi
&& (logical AND)	shprehja1 && shprehja2	e vërtetë vetëm nëse shprehja1 dhe shprehja2 janë të vërteta
|| (logical OR)	shprehja1 && shprehja2	e vërtetë nëse një prej shprehja1 dhe shprehja2 është e vërtetë

Një gjë që duhet mbajtur parasysh është se Logical Operators vlerësojnë shprehjen e dytë vetëm kur është e nevojshme. Për shembull, (true || shprehja2). Shprehja e parë, në këtë rast "true", është gjithmonë e vërtetë, kështu që Java nuk ka nevojë të vlerësojë shprehja2. E njëjta logjikë dhe me operatorin && (AND). Kjo quhet Short-circuit evaluation.

I shpjegojmë më qartë me shembullin e figurës 6. Kemi dy variablat a dhe b me vlerat përkatësisht 7 dhe 23. Në shembujt me opertatorin && (AND) shohim se vetëm një vlerë printohet. Kjo sepse duhet që të dy vlerat të jenë të vërteta që të plotësohet kushti. Në rastin e pare (a < 5) është **false** dhe si u shpjegua më sipër kjo mjafton për ta përjashtuar futjen brenda bllokut ku printohet vlera "Test 1 AND". Në këtë rast nuk kryhet kontrolli i dytë (b > 20). Në rastin e dytë (a > 5) && (b < 3) shprehja e parë është **true** dhe kalojmë në kontrollin e shprehjes së dytë e cila është **false** pasi b = 23 është më e madhe se 3. Në shembullin e tretë të dyja shprehjen (a > 5) && (b < 24) rezultojnë true dhe printohet "Test 3 AND".

Përsa i përket operatorit || (OR) mjafton që një prej shprehjeve të rezultojë **true**. Në shembullin e parë (a < 5) || (b > 20) shprehja e pare është **false** dhe kalohet në vlerësimin e shprehjes së dytë e cila është trëue, prandaj printohet teksti "Test 1 OR". Në shembullin e dytë kontrollojmë (a > 5) || (b < 3). Në këtë rast shprehja e parë (a > 5) rezulton **true** dhe nuk kryejmë kontrollin e shprehjes së dytë. Printojmë tekstin "Test 2 OR". Në shembullin e fundit (a < 5) || (b > 24) shprehja e pare rezulton **false** dhe kalojmë në kontrollin e shprehjes së dytë. Dhe shprehja e dytë rezulton **false**, kështu që nuk printohet teksti.

```java
public class LogicalOpTest {
    public static void main(String[] args) {
        int a = 7, b = 23;

        /** Shembuj me operatorin && (AND) */
        if((a < 5) && (b > 20)) {
            System.out.println("Test 1 AND"); //nuk printohet
        }

        if((a > 5) && (b < 3)) {
            System.out.println("Test 2 AND"); //nuk printohet
        }

        if((a > 5) && (b < 24)) {
            System.out.println("Test 3 AND"); //printohet
        }

        /** Shembuj me operatorin || (OR) */
        if((a < 5) || (b > 20)) {
            System.out.println("Test 1 OR"); //printohet
        }

        if((a > 5) || (b < 3)) {
            System.out.println("Test 2 OR"); //printohet
        }

        if((a > 5) || (b < 24)) {
            System.out.println("Test 3 OR"); //printohet
        }

        if((a < 5) || (b > 24)) {
            System.out.println("Test 4 OR"); //nuk printohet
        }
    }
}
```

Fig. 6 Logical operators example

Rezultati

```
Test 3 AND
Test 1 OR
Test 2 OR
Test 3 OR
```

6. Ternary Operator

Ternary Operator (operatori i kushtëzuar) është shkurtim për **if-then-else**.
Sintaksa: `condition ? if true : if false`
Kjo do të thotë që nëse shprehja rezulton **true**, atëherë ekzekutohet shprehja pas "?" përndryshe ekzekutohet shprehja pas ":". Shohim kodin e mëposhtëm.

```java
public class TernaryOpTest {
    public static void main(String[] args) {
        int daysInAYear = 366;
        String result;
        result = (daysInAYear == 366) ? "Eshte vit i brishte" : "Nuk eshte vit i brishte";
        System.out.println(result);
    }
}
```

Fig. 7 Ternary operator example

Rezultati

```
Eshte vit i brishte
```

Interpretimi: Meqënëse vlera e variablit *daysInAYear* është 366, atëherë (daysInAYear == 366) rezulton **true** prandaj vlera e **result = "Eshte vit i brishte".**

7. Bitwise & Shift Operators

Në Java, operatorët **Bitwise** kryejnë operacione në të dhëna numerike të plota në nivelin individual të bitit. Të dhënat numerike të plota përfshijnë të dhëna *byte*, *short*, *int* dhe *long*. Operatorët e diskutuar në këtë pjesë përdoren më rrallë se të tjerët.

Nëse doni të lexoni më tepër mbi këta operatorë klikoni këtu.

Operatori	Shpjegimi
~	Bitwise Complement (është një *unary operator* që kthen përfaqësuesin e komplementit të njëshit të variablit)
<<	Left Shift (zhvendos të gjitha bitet në të majtë me një numër të caktuar bitesh të specifikuar)
>>	Right Shift (zhvendos të gjitha bitet në të djathtë me një numër të caktuar bitesh të specifikuar)
>>>	Unsigned Right Shift
&	Bitwise AND (kthen bit për bit AND të vlerës së variablit)
|	Bitwise OR (kthen bit për bit OR të vlerës së variablit)
^	Bitwise exclusive OR (kthen bit për bit XOR të vlerës së variablit)

8. instanceof Operator

Operatori **instanceof** përdoret për kontrollimin e tipit. Mund të përdoret për të testuar nëse një objekt është një instancë e një klase, një nënklase ose një ndërfaqeje (interface). Shohim shembullin më poshtë për ta kuptuar më mirë. Kemi klasën **Shpend**, klasën **Harabel,** e cila *trashëgon* klasën **Shpend** dhe *implementon* ndërfaqen **TestInterface**. Te rezultatet shohim se **obj1** ka rezultat **false** për *instanceof Harabel* dhe *instanceof TestInterface*, pasi **obj1** është objekt i tipit **Shpend**. Në anën tjetër **obj2** është objekt i tipit **Harabel** i cili ka *parent class Shpend* dhe implementon *TestInterface*. Prandaj **obj2** është instancë e tyre. Kjo shihet dhe te rezultati ku të gjitha rezultatet janë **true**.

```java
public class InstanceOfOpTest {
    public static void main(String[] args) {
        Shpend obj1 = new Shpend();
        Shpend obj2 = new Harabel();

        // Shembuj me objektin e tipit Shpend
        System.out.println("obj1 instanceof Shpend: "
                + (obj1 instanceof Shpend));
        System.out.println("obj1 instanceof Harabel: "
                + (obj1 instanceof Harabel));
        System.out.println("obj1 instanceof TestInterface: "
                + (obj1 instanceof TestInterface));

        // Shembuj me objektin e tipit Harabel
        System.out.println("obj2 instanceof Shpend: "
                + (obj2 instanceof Shpend));
        System.out.println("obj2 instanceof Harabel: "
                + (obj2 instanceof Harabel));
        System.out.println("obj2 instanceof TestInterface: "
                + (obj2 instanceof TestInterface));
    }
}

class Shpend {
}

class Harabel extends Shpend implements TestInterface {
}

interface TestInterface {
}
```

Fig. 8 Instanceof operator example

Rezultati

```
obj1 instanceof Shpend: true
obj1 instanceof Harabel: false
obj1 instanceof TestInterface: false
obj2 instanceof Shpend: true
obj2 instanceof Harabel: true
obj2 instanceof TestInterface: true
```

Tabela e përparësisë së operatorëve

Tabela më poshtë rendit përparësinë e operatorëve në Java. Sa më lartë të shfaqet në tabelë, aq më e lartë është përparësia e tij.

Tipi	Operatori
postfix increment dhe decrement	++ --
prefix increment dhe decrement, dhe unary	++ -- + - ~ !
shumëzues	* / %
shtues	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
barazia	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %=

Përfundim

Në këtë artikull folëm për operatorët në Java, sintaksën e tyre dhe përse përdoren. Nuk keni përse i mbani mend përmendësh, pasi do ju fiksohen kur të filloni t'i përdorni.

Për më tepër, nëse keni pyetje, mos ngurroni të pyesni në instagram **@programuesja** ose në adresën email programuesja@gmail.com.