

به نام خدا



## برنامه‌سازی ییشوفته

دانشگاه شهید بهشتی · دانشکده مهندسی کامپیوتر

دکتر مجتبی وحیدی اصل

### دستورات کنترلی-بخش اول

نیما سلطانی

## فهرست مطالب

1. نوع بولین (Boolean type)

2. عملگرهای بولین (Boolean operators)

3. دستور ساده if

4. دستور if-else

5. دستور switch-case

6. تقدم و شرکت‌پذیری عملگرها

7. نوشتمند چند برنامه

8. مثال‌های بیشتر

## مقدمه

همه ما در زندگی روزمره شرایطی را تجربه کرده‌ایم که باید بین دو یا چند گزینه، یکی را انتخاب کنیم. مثلاً تصور کنید در یک اپ سفارش غذا هستیم: اگر ثبت سفارش را انتخاب کنیم، آنگاه سفارش ما نهایی می‌شود و پیام تأیید دریافت می‌کنیم. و اگر **انتخاب غذای دیگر** را بزنیم، آنگاه منو دوباره برایمان باز می‌شود تا غذای متفاوتی انتخاب کنیم. در غیر این صورت، سفارش لغو می‌شود و به صفحه اصلی برنمی‌گردیم.

## نوع بولین (Boolean)

مثلاً می‌خواهیم بدانیم آیا یک عدد از عدد دیگر در برنامه نویسی، گاهی لازم است دو مقدار را با هم مقایسه کنیم تا بفهمیم چه رابطه‌ای بین آنها وجود دارد برای این کار، در جاوا شش عملگر مقایسه‌ای (که به آن‌ها عملگر رابطه‌ای هم می‌گویند) وجود دارد که می‌توانند دو مقدار را مقایسه کنند و بزرگ‌تر است یا خیر (نادرست) این نتیجه همیشه یک مقدار بولین است، یعنی نتیجه را برگردانند.



```
boolean b = (1 > 2);
```

خواهد بود، چون این عبارت درست نیست **false** به عنوان مثال، وقتی می‌پرسیم: «آیا ۱ بزرگ‌تر از ۲ است؟» پاسخ

## عملگرهای مقایسه‌ای

عملگر	نام	مثال	نتیجه
<	Less than	5 < 8	true
<=	Less than or equal to	6 <= 4	false
>	Greater than	10 > 2	true
>=	Greater than or equal to	3 >= 5	false
==	Equal to	7 == 7	true
!=	Not equal to	4 != 4	false

باشد **true** یا **false** هر کدام از این عملگرها بعد از مقایسه دو مقدار، یک **مقدار بولین** برمی‌گردانند که می‌تواند

## مثال

یک ابزار ساده یادگیری ریاضی

به طور تصادفی تولید شده (مثلًا ۷ و ۹) `number1` و `number2` این برنامه برای آزمون ریاضی یک دانشآموز کلاس اولی نوشته می‌شود. دو عدد تکرقمی پس از این که دانشآموز پاسخ خود را وارد کرد، پاسخ وارد شده با پاسخ واقعی مقایسه می‌شود و پیغامی **What is 7 + 9?** : و پرسش زیر را مطرح می‌کند بودن پاسخ چاپ می‌شود **true** یا **false** مبنی بر

**برای تولید اعداد تصادفی از تابع زیر استفاده کنید:**

```
int num1 = (int)(Math.random() * 10); // returns a number from 0 to 9
```

```
In [ ]: import java.util.Scanner;

public class AdditionQuiz {
    public static void main(String[] args) {
        int number1 = (int) (Math.random() * 10);
        int number2 = (int) (Math.random() * 10);

        //Create a scanner
        Scanner input = new Scanner(System.in);

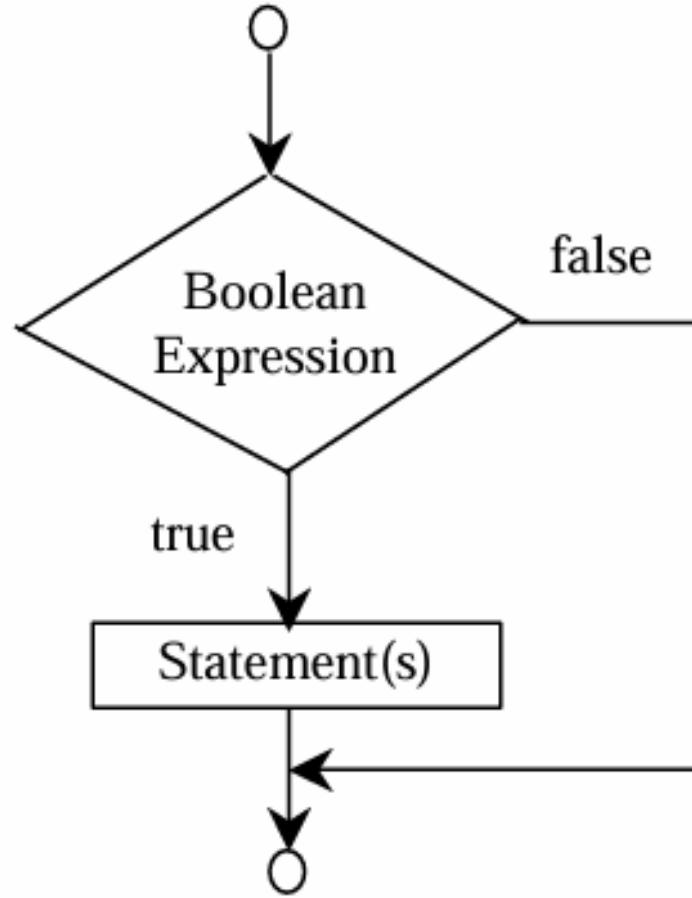
        System.out.print("What is " + number1 + " + " + number2 + "? ");
        int answer = input.nextInt();

        System.out.println(
            number1 + " + " + number2 + " = " + answer + " is " + (number1 + number2 == answer));
    }
}
```

## دستور if تک وضعیتی (یکطرفه)

اجرا می‌شوند if آشنا می‌شویم که در آن شرط بررسی می‌شود و در صورت درست بودن، دستورات مربوطه به همان if در این بخش با ساختار کلی دستور

```
if (boolean-expression) {
    statement(s);
}
```

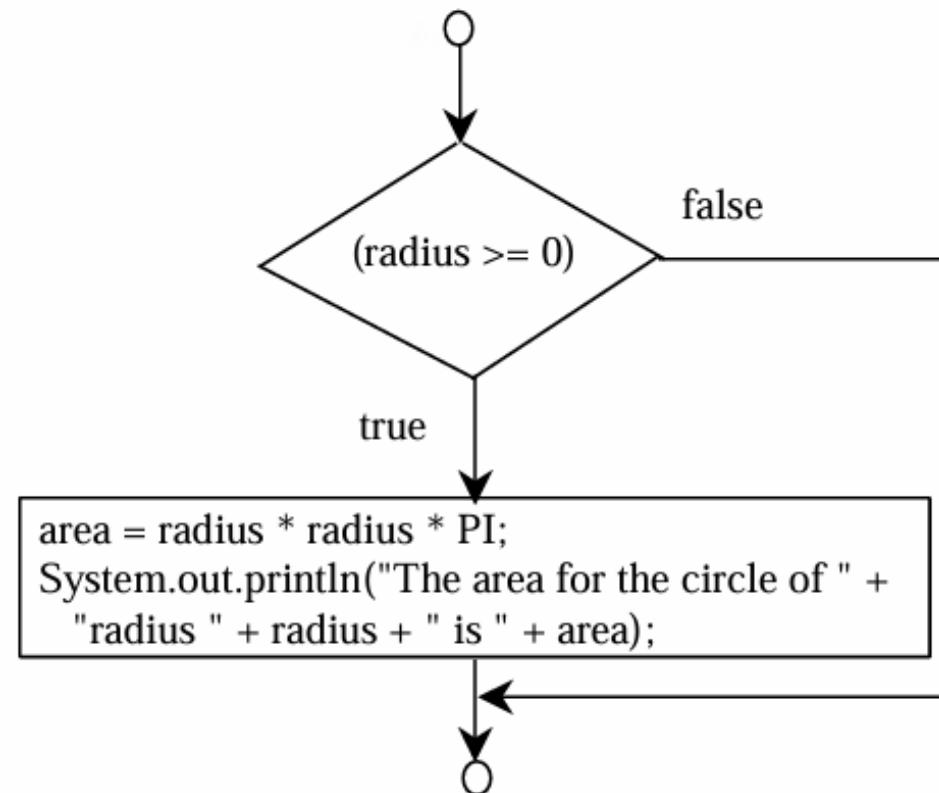


مثال

محاسبه مساحت دایره با بررسی شعاع

برنامه‌ای بنویسید که مقدار شعاع دایره را از کاربر دریافت کند. اگر مقدار شعاع منفی نباشد، مساحت دایره محاسبه و چاپ شود.

```
if (radius >= 0) {  
    area = radius * radius * PI;  
    System.out.println("The area"  
        + " for the circle of radius "  
        + radius + " is " + area);  
}
```



In [ ]: `import java.util.Scanner;`

```
public class CircleArea {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter radius: ");
        double radius = input.nextDouble();

        if (radius >= 0) {
            double area = radius * radius * Math.PI;
            System.out.println("The area for the circle of radius " + radius + " is " + area);
        }
    }
}
```

## مثال

چاپ کند **HiFive** و اگر مضرب ۵ بود، پیغام **HiEven** برنامه‌ای بنویسید که یک عدد صحیح را از کاربر بگیرد. اگر عدد مضرب ۰ بود، پیغام

```
In [ ]: import java.util.Scanner;

public class SimpleIfDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int number = input.nextInt();

        if (number % 5 == 0)
            System.out.println("HiFive");

        if (number % 2 == 0)
            System.out.println("HiEven");
    }
}
```

## نکات مهم در نوشتن شرط‌ها در زبان جاوا

تفاوت استفاده صحیح و اشتباه از پرانتز و آکولاد

در جاوا، شرط‌ها باید داخل پرانتز () نوشته شوند و بدنه با آکولاد {} مشخص شود. رعایت نکردن این ساختار باعث خطا می‌شود.

نبود پرانتز باعث خطای نحوی در شرط می‌شود: **نادرست (الف)**.

```
if i > 0 {  
    System.out.println("i is positive");  
}
```

شرط داخل پرانتز و بدنه داخل آکولاد قرار گرفته است: **صحیح (ب)**.

```
if (i > 0) {  
    System.out.println("i is positive");  
}
```

اگر فقط یک دستور در بدنه شرط باشد، حذف آکولاد {} مجاز است. در این حالت فقط همان دستوری که بلافاصله بعد از شرط امده است اجرا می‌شود

**حالت اول - با آکولاد**

```
if (i > 0) {  
    System.out.println("i is positive");  
}
```

حالت دوم - بدون آکولاد

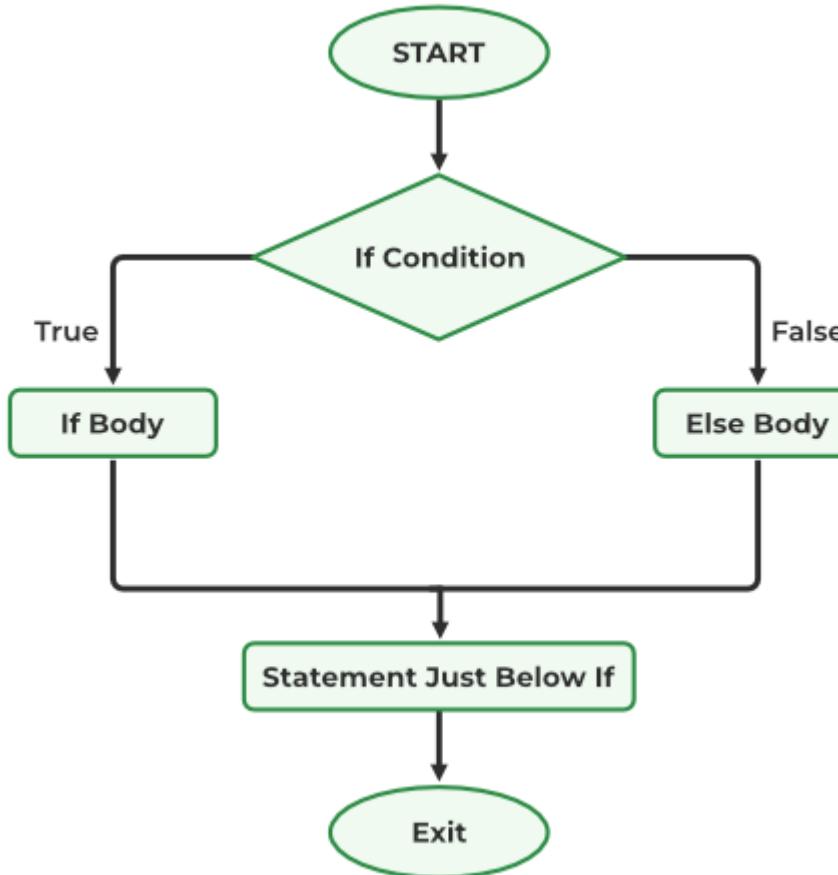
```
if (i > 0)
    System.out.println("i is positive");
```

## شرطهای دوطرفه

استفاده می‌شود. این ساختار شامل یک شرط منطقی و دو if-else زمانی که بخواهیم بر اساس نتیجه یک شرط، یکی از دو مسیر ممکن را اجرا کنیم، از ساختار بلوک مجزا برای حالت درست و نادرست است.

در صورت برقرار بودن شرط، بلوک اول اجرا می‌شود؛ در غیر این صورت، بلوک دوم [ساختار کلی](#).

```
if (condition) {
    // statements for true case
} else {
    // statements for false case
}
```



چاپ می‌شود؛ در غیر این صورت، پیام "Failed" کمتر از 60 باشد، پیام grade در مثال زیر، اگر مقدار **مثلاً**:

```

if (grade < 60) {
    System.out.println("Failed");
} else {
    System.out.println("Passed");
}
  
```

## مثال

برنامه‌ای بنویسید که مقدار شعاع یک دایره را دریافت کند. اگر شعاع مقدار غیرمنفی داشته باشد، مساحت دایره محاسبه و چاپ شود. در غیر این صورت، پیام "Negative input" داده شود.

```
In [ ]: import java.util.Scanner;

public class CircleArea {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter radius: ");
        double radius = input.nextDouble();

        if (radius >= 0) {
            double area = radius * radius * 3.14159;
            System.out.println("The area for the circle of radius " + radius + " is " + area);
        } else {
            System.out.println("Negative input");
        }
    }
}
```

## روش‌های مختلف نوشتن دستورات if

```
if (score >= 90.0)
    grade = 'A';
else
    if (score >= 80.0)
        grade = 'B';
    else
        if (score >= 70.0)
            grade = 'C';
        else
            if (score >= 60.0)
                grade = 'D';
            else
                grade = 'F';
```

کاراً **معادل** است و نتیجه را کسانم می‌دانم. تهه داشته باشد که چون در آن که **if** **else if** **else if** دارد استفاده می‌کند از نظر عملکرد باید این دو شرط را از آنکه از آنها کدامیک را اجرا نماید بررسی کند. اگر می‌خواستیم بیش از یک دستور در هر بخش قرار دهیم، باید آنها را داخا آنها را در یک قرار می‌دادیم.

```
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

## ردگیری اجرای شرط‌ها و تعیین نتیجه

Suppose score is **70.0**

```
if (score >= 90.0) The condition is false
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

چون این شرط برقرار نیست پس وارد دستورات مربوط به آن نمی‌شویم

## ردگیری اجرای شرط‌ها و تعیین نتیجه

```
Suppose score is 70.0
```

```
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0) The condition is false
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

چون شرط دوم برقرار نیست، برنامه به بررسی شرط‌های بعدی ادامه می‌دهد.

## ردگیری اجرای شرط‌ها و تعیین نتیجه

```
Suppose score is 70.0
```

```
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
```

```
else if (score >= 70.0) The condition is true
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

در این گام شرط سوم برقرار است و مسیر اجرا به همان شاخه هدایت می‌شود.

## ردگیری اجرای شرط‌ها و تعیین نتیجه

Suppose score is **70.0**

```
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C'; grade is C
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

اختصاص می‌باید grade به متغیر C نتیجه نهایی: مقدار

## ردگیری اجرای شرط‌ها و خروج از ساختار if

Suppose score is **70.0**

```
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

// Exit the if statement
```

اجرای برنامه از ساختار شرطی خارج می‌شود، grade پس از تعیین مقدار

## توجه

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
    if (i > k)
        System.out.println("A");
    else
        System.out.println("B");
```

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
```

در یک بلاک متصل می‌شود. برای جلوگیری از ابهام، استفاده از آکولاد `{ }`  (خروجی هر دو کد بالا یکسان `if` همیشه به نذیرکت نه `else` رخشد. است) توصیه می‌شود.

## مثال

### استفاده صحیح از آکولاد

برنامه‌ای که در این مثال می‌بینید، بدون استفاده از آکولادها هیچ خروجی چاپ نمی‌کند. افزودن آکولادها باعث می‌شود بلوک‌های شرطی به درستی مشخص و منطق برنامه واضح شود.

```
int i = 1;
int j = 2;
int k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");

else
    System.out.println("B");
```

توجه کنید که استفاده صحیح از آکولادها در شرط‌های متداخل از بروز ابهام و خطأ در اجرای برنامه جلوگیری می‌کند. توجه کنید که کد بالا هیچ عبارتی را چاپ نمی‌کند

## خطای برنامه نویسی

باعث می شود بدنه شرط عملاً «دستور خالی» باشد و بلوک آکولادی بعدی یک بلوک مستقل تلقی شود که `if` قرار دادن سمی کالن (;) بلافاصله بعد از شرط همیشه اجرا می شود. این خطای نحوی است و نه در زمان اجرا استثنای دهد، به همین دلیل کشف آن دشوار است و صرفاً خروجی / منطق برنامه اشتباه [کد اشتباه](#) می شود:

```
if (radius >= 0); // Empty statement ends the if condition
{
    area = radius * radius * PI; // This block runs regardless of the if condition
    System.out.println(
        "The area for the circle of radius " + radius + " is " + area);
}
```

[منطق خطای آنچه واقعاً اجرا می شود](#):

```
// if statement has an empty body and does nothing
if (radius >= 0)
    ; // Empty statement
```

// This block is independent of the if and always executes { area = radius \* radius \* PI; System.out.println( "The area for the circle of radius " + radius + " is " + area); } [کد صحیح](#)

```
if (radius >= 0) { // Only run this block when the condition is true
    area = radius * radius * PI; // Calculate the area
    System.out.println(
        "The area for the circle of radius " + radius + " is " + area); // Print result
}
```

- پایان دستور شرطی را اعلام می‌کند؛ بنابراین شرط فقط ارزیابی می‌شود و هیچ بدنه‌ای ندارد. آکولادهای if **دلیل بروز خطأ**: سمی‌کالن بعد از
- بعدی یک بلوك جداگانه هستند که همیشه اجرا می‌شوند (چه شرط درست باشد چه غلط)
  - از آکولاد استفاده کنید – حتی اگر یک دستور باشد – تا از ابهام جلوگیری شود if **راه حل**: سمی‌کالن را حذف کنید و همواره برای بدنہ •

## ساده‌تر بنویسیم

```
if (number % 2 == 0)
    even = true;
else
    even = false;
```

هر دو روش از نظر عملکرد **معادل** هستند. روش دوم ساده‌تر و خواناتر است و معمولاً در برنامه‌نویسی ترجیح داده می‌شود.

```
boolean even = number % 2 == 0;
```

## ساده‌تر بنویسیم

```
if (even == true)
    System.out.println("It is even.");
```

هر دو روش از نظر عملکرد **معادل** هستند. روش دوم ساده‌تر و خواناتر است و معمولاً در برنامه‌نویسی ترجیح داده می‌شود.

```
if (even)
    System.out.println("It is even.");
```

## مثال

یک ابزار ساده یادگیری ریاضی

اگر `number1 > number2` باشد، یک سؤال تفریق نمایش داده می‌شود، مانند:

What is  $9 - 2$ ?

"Your answer is **You are correct.**" داشت آمده باسخ باشد می‌گند. اگر باسخ درست باشد، سام همراه با پاسخ صحیح نشان داده می‌شود "Your answer is **wrong.**"

```
In [ ]: import java.util.Scanner;

public class SubtractionQuiz {
    public static void main(String[] args) {
        // 1. Generate two random single-digit integers
```

```

int number1 = (int)(Math.random() * 10);
int number2 = (int)(Math.random() * 10);

// 2. If number1 < number2, swap number1 with number2
if (number1 < number2) {
    int temp = number1;
    number1 = number2;
    number2 = temp;
}

// 3. Prompt the student to answer "what is number1 - number2?"
System.out.print("What is " + number1 + " - " + number2 + "? ");
Scanner input = new Scanner(System.in);
int answer = input.nextInt();

// 4. Grade the answer and display the result
if (number1 - number2 == answer)
    System.out.println("You are correct!");
else
    System.out.println("Your answer is wrong.\n" + number1 + " - " + number2 + " should be " + (number1 - n
}
}

```

## عملگرهای منطقی

عملگر	نام
!	Not
&&	And
	Or

عملگر	نام
^	Exclusive Or

این عملگرها برای ترکیب یا تغییر مقادیر [بولین](#) استفاده می‌شوند و در تصمیم‌گیری‌های منطقی کاربرد دارند.

## جدول درستی برای عملگر ! (نفی)

Expression (p)	Negation (!p)	Example	Result
true	false	<code>!(score &gt;= 90 &amp;&amp; passed)</code> score = 95, passed = true	false
false	true	<code>!(temperature &lt; 0    isSnowing)</code> temperature = 5, isSnowing = false	true
true	false	<code>!(user.isLoggedIn &amp;&amp; user.role == "admin")</code> user = {isLoggedIn: true, role: "admin"}	false
false	true	<code>!(balance &gt; 0 &amp;&amp; account.isActive)</code> balance = -20, account.isActive = true	true

عملگر ! مقدار منطقی را برعکس می‌کند. یعنی اگر شرطی درست باشد، با ! نادرست می‌شود و برعکس.

## جدول درستی برای عملگر **&&** (و منطقی)

p1	p2	p1 && p2	Example	Result
false	false	false	(age > 18) && (gender == 'M') age = 24, gender = 'F'	false
false	true	false	(3 > 2) && (5 > 5) 3 > 2 is true, 5 > 5 is false	false
true	false	false	(age > 18) && (gender != 'F') age = 24, gender = 'F'	false
true	true	true	(3 > 2) && (5 >= 5)	true

عملگر **&&** فقط وقتی نتیجه اش true می شود که هر دو شرط درست باشند. اگر یکی از آنها false باشد، نتیجه false خواهد بود.

## جدول درستی برای عملگر **||** (یا منطقی)

p1	p2	p1    p2	Example	Result
false	false	false	(age > 34)    (gender == 'M') age = 24, gender = 'F'	false
false	true	true	(age > 34)    (gender == 'F') age = 24, gender = 'F'	true
true	false	true	(3 > 2)    (5 > 5)	true
false	false	false	(2 > 3)    (5 > 5)	false

عملگر || وقتی نتیجه اش true می شود که حداقل یکی از شرط ها درست باشد. فقط وقتی هر دو شرط نادرست باشند، نتیجه false خواهد بود.

## جدول درستی برای عملگر ^ (یا انحصاری)

p1	p2	p1 ^ p2	Example	Result
false	false	false	(age > 34) ^ (gender == 'M') age = 24, gender = 'F'	false
false	true	true	(age > 34) ^ (gender == 'F') age = 24, gender = 'F'	true
true	false	false	(3 > 2) ^ (5 > 5)	true

p1	p2	$p1 \wedge p2$	Example	Result
true	true	false	$(3 > 2) \wedge (5 \geq 5)$	false

عملگر  $\wedge$  فقط وقتی نتیجه اش true می شود که دقیقاً یکی از شرط ها درست باشد. اگر هر دو شرط مثل هم باشند (هر دو درست یا هر دو نادرست هستند) خواهد بود.

## مثال

برنامه ای بنویسید که عددی را از کاربر دریافت کند و بررسی کند آیا

بر ۲ و ۳ بخش پذیر است.

بر ۲ یا ۳ بخش پذیر است.

بر ۲ یا ۳ اما نه بر هر دو بخش پذیر است.

```
In [ ]: import java.util.Scanner;

public class DivisibilityChecker {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Get input from user
        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        // Check divisibility by both 2 and 3
        System.out.println("Is " + number + " divisible by 2 and 3? " +
            ((number % 2 == 0) && (number % 3 == 0)));

        // Check divisibility by either 2 or 3
        System.out.println("Is " + number + " divisible by 2 or 3? " +
            ((number % 2 == 0) || (number % 3 == 0)));
    }
}
```

```

    // Check divisibility by 2 or 3, but not both
    System.out.println("Is " + number + " divisible by 2 or 3, but not both? " +
        ((number % 2 == 0) ^ (number % 3 == 0)));
}
}

```

## مثال

تعیین سال کبیسه

برنامه‌ای بنویسید که یک عدد صحیح را به عنوان سال از کاربر دریافت کند و بررسی کند که آیا آن سال کبیسه است یا خیر. سال کبیسه سالی است که یکی از شرایط زیر را داشته باشد  
بر ۴ بخش‌پذیر باشد اما بر ۱۰۰ بخش‌پذیر نباشد.  
یا اینکه بر ۴۰۰ بخش‌پذیر باشد.

```
In [ ]: import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        // Create a Scanner to get user input
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a year: ");
        int year = input.nextInt();

        // Check if the year is a leap year
        boolean isLeapYear =
            (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        // Display the result
        System.out.println(year + " is a leap year? " + isLeapYear);
    }
}
```

## مثال

XOR جابجایی دو عدد با استفاده از عملگر

جابجا کند XOR برنامه‌ای بنویسید که دو عدد صحیح را از کاربر دریافت کند و بدون استفاده از متغیر کمکی، آن‌ها را با استفاده از عملگر

به عنوان مثال، اگر ورودی‌ها به صورت زیر باشند:

(a) عدد اول = ۰

(b) عدد دوم = ۶

پس از اجرای برنامه، خروجی باید به صورت زیر باشد

a = ۶

b = ۰

```
In [ ]: import java.util.Scanner;

public class SwapWithXOR {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Get two integers from the user
        System.out.print("Enter first number (a): ");
        int a = input.nextInt();

        System.out.print("Enter second number (b): ");
        int b = input.nextInt();

        // Display original values
        System.out.println("Before swap: a = " + a + ", b = " + b);
```

```

// Swap using XOR
a = a ^ b; // Step 1: a holds a ^ b
b = a ^ b; // Step 2: b becomes original a
a = a ^ b; // Step 3: a becomes original b

// Display swapped values
System.out.println("After swap: a = " + a + ", b = " + b);
}
}

```

## مثال

| بررسی شرایط با عملگرهای & و

این برنامه نمونه‌ای از استفاده عملگرهای منطقی & و | در عبارات شرطی را نشان می‌دهد.

بزرگتر یا مساوی ۶۰ است age برابر ۱ و مقدار gender شرط اول بررسی می‌کند که آیا مقدار ۱.

مقدار آن بزرگتر یا مساوی ۶۰ می‌شود، age است یا پس از یک واحد اضافه شدن به true برابر birthday شرط دوم بررسی می‌کند که آیا ۱.

نکته: عملگرهای & و | هر دو سمت شرط را همیشه ارزیابی می‌کنند، حتی اگر سمت اول نتیجه نهایی را تعیین کند. این رفتار با عملگرهای && و || (نسخه کوتاه‌سازی شده) متفاوت است که ممکن است سمت دوم را چک نکنند.

```

( gender == 1 ) & ( age >= 65 )
( birthday == true ) | ( ++age >= 65 )

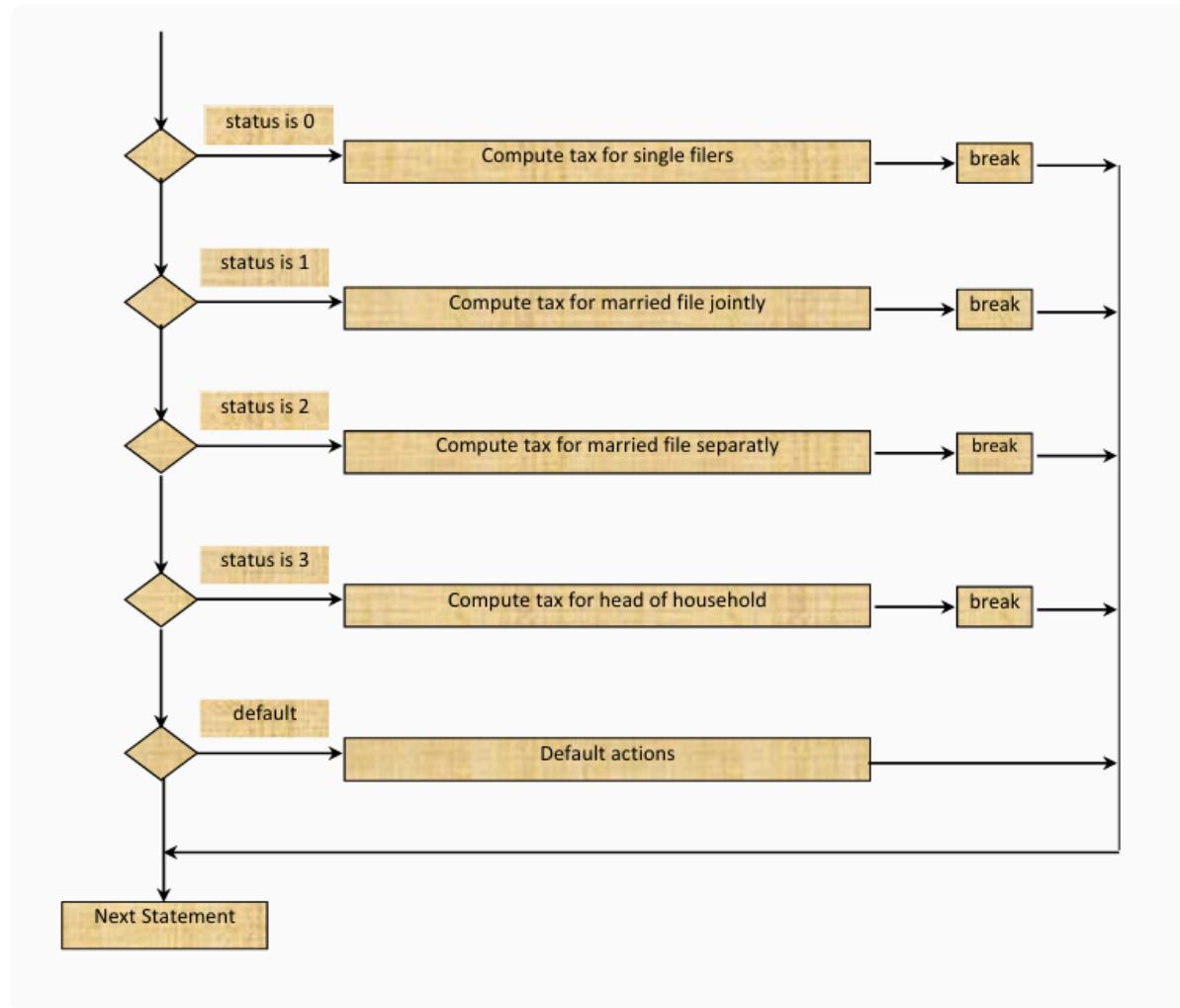
```

توجه کنید که انتخاب بین (& / |) و (&& / ||) می‌تواند بر منطق و عملکرد برنامه اثر بگذارد، به ویژه زمانی که سمت دوم شرط دارای اثر جانبی باشد age با ++age مثل افزایش مقدار

## switch دستور معرفی

فلوچارت تصمیم‌گیری بر اساس وضعیت‌های مختلف

برای تصمیم‌گیری بین چند حالت مختلف بر اساس مقدار یک متغیر استفاده می‌شود `switch` دستور.



قواعد دستور `switch`

ساختار و شرایط استفاده از دستور switch

باشد. توجه کنید که این مقادیر باید ثابت باشند، یعنی نباید از متغیر یا عباراتی مانند `switch` `value1, value2, ... valueN` باشد. باید همنوع با عبارت داخل `switch` باشند. همچنان که در `case` استفاده شود  $(x+1)$ .

```
switch (switch-expression) {  
    case value1: statement(s)1;  
        break;  
    case value2: statement(s)2;  
        break;  
    ...  
    case valueN: statement(s)N;  
        break;  
    default: statement(s)-for-default;  
}
```

## قواعد دستور switch

نقش و break در کنترل جریان

بعدی جلوگیری کند case استفاده می‌شود تا از اجرای ناخواسته case اختیاری است، اما معمولاً در انتهای هر break کلمه کلیدی

بعدی هم خواهد شد case نوشته نشود، اجرای برنامه وارد `break` اگر

برابر نباشند `default` بخش با `switch-expression` اجرا می‌شود که هیچ‌یک از مقادیر `case` اختیاری است و زمانی اجرا می‌شود که هیچ‌یک از مقادیر `case` نباشد.

```
switch (switch-expression) {  
    case value1: statement(s)1;  
        break;  
    case value2: statement(s)2;  
        break;  
    ...  
    case valueN: statement(s)N;  
        break;  
    default: statement(s)-for-default;  
}
```

## ردگیری دستور `switch`

```
switch (ch) {  
    case 'a': System.out.println(ch);  
    case 'b': System.out.println(ch);  
    case 'c': System.out.println(ch);  
}
```

نیز ادامه پیدا `case b` اجرای برنامه به ، `break` اجرا می‌شود و به دلیل نبود `System.out.println(ch)` برقرار است، خط `case a` چون مقدار می‌کند.

## ردگیری دستور `switch`

```
switch (ch) {  
    case 'a': System.out.println(ch);  
    case 'b': System.out.println(ch);  
    case 'c': System.out.println(ch);  
}
```

یا پایان `break` وجود نداشت، بنابراین عبارت مربوط به این کس نیز اجرا می‌شود و این روند تا رسیدن به یک `case a` قللی جمیع داده ادامه پیدا می‌کند `switch`.

## ردگیری دستور `switch`

```
switch (ch) {  
    case 'a': System.out.println(ch);  
    case 'b': System.out.println(ch);
```

```
    case 'c': System.out.println(ch);
}
```

ادامه پیدا می‌کند `switch` یا اتمام `break` نداشتم، پس عبارت این کیس هم اجرا می‌شود و این روند تا رسیدن به یک `break` چون در کیس قبلی

## ردگیری دستور `switch`

```
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
Next statement;
```

کنترل برنامه به اولین دستور بعد از آن منتقل می‌شود و روند اجرای کد از همانجا ادامه پیدا می‌کند، `switch` پس از اجرای آخرین دستور در بخش

## ردگیری دستور `switch`

```
// suppose 'ch' is 'a'  
switch (ch) {  
    case 'a': System.out.println(ch);  
        break;  
    case 'b': System.out.println(ch);  
        break;  
    case 'c': System.out.println(ch);  
}
```

فعال می‌شود 'a' مربوط به case است، بنابراین 'a' برابر ch مقدار

## اجرای case انتخاب شده

```
switch (ch) {  
    case 'a': System.out.println(ch);  
        break;  
    case 'b': System.out.println(ch);  
        break;  
    case 'c': System.out.println(ch);  
}
```

چاپ خواهد شد 'a' اجرا می‌شود و مقدار System.out.println(ch); دستور

## اجرای break

```
switch (ch) {  
    case 'a': System.out.println(ch);  
        break;  
    case 'b': System.out.println(ch);  
        break;  
    case 'c': System.out.println(ch);  
}
```

میگردد switch اجرا میشود و باعث خروج از ساختار break دستور

## ادامه اجرای برنامه

```
switch (ch) {  
    case 'a': System.out.println(ch);  
        break;  
    case 'b': System.out.println(ch);  
        break;  
    case 'c': System.out.println(ch);
```

```
}
```

Next statement;

منتقل می‌شود `switch` کنترل برنامه به اولین دستور بعد از `break` پس از اجرای

## ادامه اجرای برنامه

```
switch (ch) {
    case 'a': System.out.println(ch);
                break;
    case 'b': System.out.println(ch);
                break;
    case 'c': System.out.println(ch);
}
Next statement;
```

منتقل می‌شود `switch` کنترل برنامه به اولین دستور بعد از `break` پس از اجرای

## عملگر شرطی ؟

عملگر شرطی ؟ : یک بیان تکخطی برای تصمیم‌گیری است: اگر شرط بقدام باشد مقدار آها، هگنه مقدار دهم بگدانده مه، شهد. این عملگر نک «سا:» است، بنابراین باید در متغیر ریخته شود یا در جایی که انتظار مقدار می‌رود به کار رود.

```
(boolean-expression) ? expression1 : expression2
```

```
int y = (x > 0) ? 1 : -1;
```

## مثال

باشد  $y = -1$  و در غیر این صورت  $y = 1$  باشد، مقدار  $x > 0$  اگر.

```
if (x > 0)
    y = 1;
else
    y = -1;
```

is equivalent to

```
y = (x > 0) ? 1 : -1;
```

## مثال

نمره یک دانشجو را دریافت کنید و وضعیت او را در یکی از سه دسته زیر قرار دهید:

- اگر نمره بزرگتر یا مساوی 85 باشد → **عالی**
- اگر نمره بین 50 تا 84 باشد → **قبول**
- اگر کمتر از 50 باشد → **مردود**

هدف، استفاده از عملگر شرطی متداخل برای حل این مسئله است، به طوری که همه شرط‌ها در یک عبارت بررسی شوند.

```
In [ ]: public class Main {  
    public static void main(String[] args) {  
        int score = 73; // sample input  
  
        String status = (score >= 85) ? "Excellent": (score >= 50) ? "Pass": "Fail";  
  
        System.out.println("Student status: " + status);  
    }  
}
```

## مثال

هدف این عددی از کاربر دریافت کنید. اگر عدد بر ۲ بخش‌پذیر بود، پیام "عدد زوج است" چاپ شود. در غیر این صورت، پیام "عدد فرد است" چاپ شود و سپس با استفاده از عملگر شرطی `? :` حل کنیم تا تفاوت و شباهت‌ها مشخص شود `if/else` است که این مسئله را ابتدا با

```
In [ ]: import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = input.nextInt();

        // Using if/else
        if (num % 2 == 0) {
            System.out.println(num + " is even");
        } else {
            System.out.println(num + " is odd");
        }

        input.close();
    }
}
```

```
In [ ]: import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = input.nextInt();

        // Using ternary operator
        System.out.println((num % 2 == 0) ? num + " is even" : num + " is odd");

        input.close();
    }
}
```

## ساختاربندی خروجی با printf

خروجی را به شکل دلخواه قالببندی کرد. این روش باعث `System.out.printf(format, items)` در زبان حاوا می‌توان، را استفاده از دستور، رشته‌ای است که از ترکیب زیررشته‌ها و تعیین‌گرهای فرمت تشکیل **format** بخش می‌شود متن و مقادیر متغیرها با نظم و ظاهر مشخصی نمایش داده شوند با علامت `%` شروع می‌شود و نوع و نحوه نمایش یک آیتم را مشخص می‌کند. این آیتم می‌تواند یک مقدار (**format specifier**) شده است. هر **تعیین‌گر فرمت** عددی، کاراکتر، مقدار بولین یا رشته باشد.

### تعیین‌گرهای پرکاربرد در printf

Specifier	خرجی	Example
<code>%b</code>	مقدار بولین (درست/نادرست)	<b>true or false</b>
<code>%c</code>	کاراکتر	<code>'a'</code>
<code>%d</code>	عدد صحیح دهدی	<code>200</code>
<code>%f</code>	عدد اعشاری (اعشار ثابت)	<code>45.460000</code>
<code>%e</code>	نمایش علمی عدد	<code>4.556000e+01</code>
<code>%s</code>	رشته متنی	<code>Java is cool</code>

### :مثال:

- یک عدد صحیح (count)
- دریافت و در یک جمله چاپ شود (amount) یک عدد اعشاری

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
// Output: count is 5 and amount is 45.560000
```

### عملگرهای بیتی (Bitwise Operators) در جاوا

د، حماه، **عملگرها**، بته، محوظه‌ها، از عملگرها که عملیات خود را برای داده‌ها، عددی، سطح **ست** انجام می‌دهند. آن عملگرها تواند اینهای این است که نتیجه را بر (Logical) قابل استفاده‌اند. تفاوت آن‌ها با عملگرها، بیشتر int، long، short، char، byte.

## مثال

- › a = 60 1100 0011: نمایش باینری
  - › b = 13 1101 0000: نمایش باینری

## انواع عملگرهای بیتی و نتیجه آن‌ها

عملگر	نام	توضیح	نتیجه باینری
a & b	AND	هر بیت فقط زمانی 1 می‌شود که هر دو بیت متناظر برابر 1 باشند.	0000 1100

عملگر	نام	توضیح	نتیجه باینری
a   b	OR	هر بیت زمانی 1 می‌شود که حداقل یکی از بیت‌های متناظر 1 باشد.	0011 1101
a ^ b	XOR	هر بیت زمانی 1 می‌شود که دقیقاً یکی از بیت‌های متناظر 1 باشد.	0011 0001
~a	NOT	همه بیت‌ها معکوس می‌شوند (0 به 1 و 1 به 0 تبدیل می‌شود)	1100 0011

### نکات مهم:

- عملگرهای بیتی با عملگرهای منطقی (AND و OR) متفاوت هستند و نتیجه عددی بر می‌گردانند.
- برای مشاهده بهتر، معمولاً اعداد به صورت باینری نمایش داده می‌شوند.
- در جاوا نوع نتیجه وابسته به نوع عملوندهاست. در صورت نیاز به تبدیل، باید نوع داده را تغییر دهید.

## عملگرهای بیتی (Bitwise Operators) در جاوا

عملگر	نام	توضیح	مثال
&	AND	هر بیت فقط وقتی 1 می‌شود که در هر دو عدد همان بیت برابر 1 باشد (مثل لامپی که باید دو کلیدش همزمان روشن باشند)	(A & B) → 12 (0000 1100)
	OR	بیت 1 می‌شود اگر حداقل یکی از دو بیت همان موقعیت 1 باشد (مثل	(A   B) → 61 (0011 1101)

عملگر	نام	توضیح	مثال
		لامپی که با روشن کردن یکی از دو کلیدش روشن می شود)	
$\wedge$	<b>XOR</b>	بیت ۱ می شود اگر دقیقاً یکی از دو بیت همان موقعیت ۱ باشد (مثل سوییچ انتخاب که فقط یک طرف باید فعال باشد)	$(A \wedge B) \rightarrow 49 (0011\ 0001)$
$\sim$	<b>NOT</b>	تمام بیت‌ها معکوس می‌شوند (۰، ۱ به ۱ و ۰، ۰ به ۱ مثلاً نگاتیو عکس)	$(\sim A) \rightarrow -61 (1100\ 0011)$
$<<$	<b>Left Shift</b>	بیت‌ها را به چپ می‌برد و صفر اضافه می‌کند (هر شیفت چپ یعنی ضرب در ۲)	$A << 2 \rightarrow 240 (1111\ 0000)$
$>>$	<b>Right Shift</b>	بیت‌ها را به راست می‌برد و علامت را حفظ می‌کند (هر شیفت راست یعنی تقسیم بر ۲)	$A >> 2 \rightarrow 15 (0000\ 1111)$
$>>>$	<b>Unsigned Right Shift</b>	مثل شیفت راست، ولی جای خالی همیشه صفر می‌آید (علامت بی‌اثر می‌شود)	$A >>> 2 \rightarrow 15 (0000\ 1111)$

## اولویت عملگرها

در این جدول، عملگرهای رایج در زبان‌های برنامه‌نویسی به ترتیب اولویت اجرا فهرست شده‌اند. این ترتیب به شما کمک می‌کند تا عبارات پیچیده را بهتر تحلیل و پیاده‌سازی کنید.

1. `var++`, `var--` : افزایش یا کاهش مقدار متغیر (پسوندی)
2. `++var`, `--var` : افزایش یا کاهش مقدار متغیر (پیشوندی)

3.  $!$  (NOT) عملگر منفی منطقی :
4.  $\sim$  (bitwise NOT) مکمل بیت‌ها :
5. (type) Casting تبدیل نوع داده :
6.  $*$ ,  $/$ ,  $\%$  ضرب، تقسیم و باقیمانده :
7.  $+$ ,  $-$  جمع و تفریق :
8.  $<<$ ,  $>>$ ,  $>>>$  شیفت بیت‌ها به چپ یا راست :
9.  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  مقایسه کوچکتر، بزرگتر و مساوی :
10.  $\neq$ ,  $\equiv$  بررسی برابری یا نابرابری :
11.  $\oplus$  (XOR) یا منطقی انحصاری :
12.  $\&$  (AND) و منطقی یا بیتبه‌بیت :
13.  $\wedge$  (XOR) یا منطقی انحصاری بیتبه‌بیت :
14.  $\mid$  (OR) یا منطقی یا بیتبه‌بیت :
15.  $\&\&$  (AND) و شرطی :
16.  $\mid\mid$  (OR) یا شرطی :
17.  $\text{?:}$  عملگر شرطی (سه‌تایی) :
18.  $=$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\&=$ ,  $\wedge=$ ,  $\mid=$ ,  $<=>$ ,  $>=>$ ,  $>>=>$  عملگرهای انتساب :

این لیست به شما کمک می‌کند تا بدانید کدام عملگرها ابتدا اجرا می‌شوند و چگونه می‌توان عبارات منطقی و ریاضی را به درستی ساختار داد.

### تقدم عملکردها و شرکت‌پذیری:

- شرکت‌پذیری همه عملکردهای دو عملوندی (به جز انتساب) از چپ به راست می‌باشد.
- شرکت‌پذیری عملکردهای تک عملوندی افزایشی (پیشوندی و پسوندی) از راست به چپ می‌باشد.
- شرکت‌پذیری عملکرد شرطی  $\text{?:}$  از راست به چپ می‌باشد.

### نکته درباره عملگر تقسیم در جاوا:

هر دو عدد صحیح هستند، بنابراین تقسیم به صورت صحیح انجام شده و حاصل یک عدد صحیح برابر با یک است.

3.0 / 2

چون یکی از عملوند‌ها اعشاری است، تقسیم به صورت اعشاری انجام می‌شود و حاصل برابر یک و نیم خواهد بود.

3 / 2.0

در این حالت هم وجود یک عملوند اعشاری باعث می‌شود خروجی یک عدد اعشاری برابر با یک و نیم باشد.

```
int b = 3; int c = 2; float a = b / c;
```

ابتدا تقسیم صحیح انجام می‌شود و سپس نتیجه به نوع اعشاری تبدیل می‌گردد، بنابراین مقدار نهایی یک اعشاری خواهد بود.

```
double b = 3.0; int c = 2; double a = b / c;
```

یکی از عملوند‌ها اعشاری است، پس تقسیم به صورت اعشاری انجام شده و نتیجه برابر یک و نیم خواهد بود.

## اولویت عملگرها

### مرحله ۱: داخل پرانتز

```
3 + 4 * 4 > 5 * (4 + 3) - 1
```

ابتدا عملیات داخل ( ) انجام می‌شود.

## مرحله ۲: ضرب اول

$$3 + 4 * 4 > 5 * 7 - 1$$

ضرب با اولویت بالاتر از جمع، ابتدا انجام می‌شود.

## مرحله ۳: ضرب دوم

$$3 + 16 > 5 * 7 - 1$$

دومین ضرب اجرا می‌شود.

## مرحله ۴: جمع

$$3 + 16 > 35 - 1$$

حالا جمع انجام می شود.

## مرحله ۰: تفريقي

$19 > 35 - 1$

سپس تفريقي اجرا می شود.

## مرحله ۶: مقاييسه

$19 > 34$

مقاييسه انجام می شود و نتيجه `false` است.

## Confirmation Dialog (GUI)

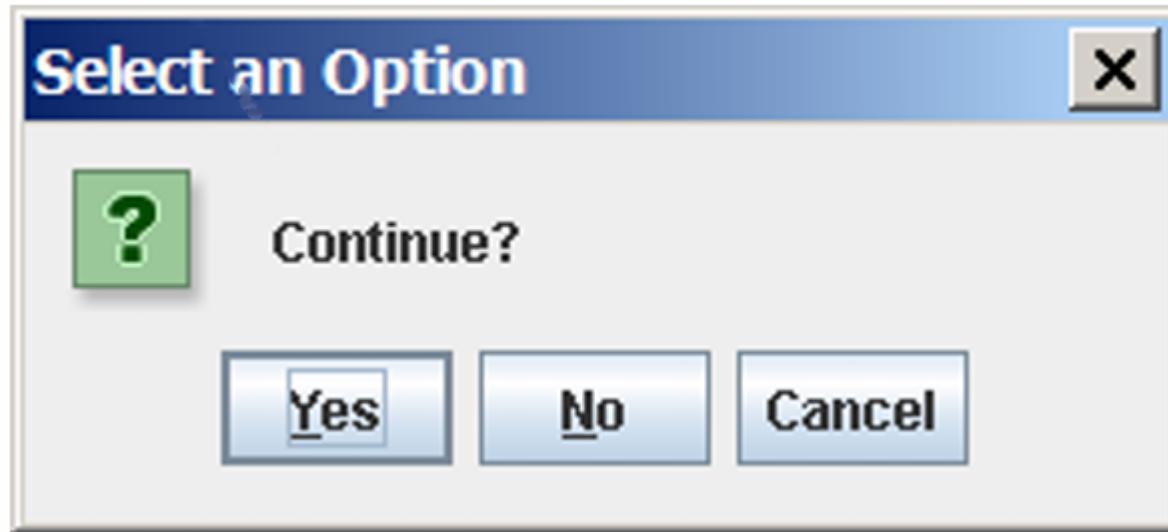
مرحله ۱: ايجاد ديالوگ تأييد

```
int option = JOptionPane.showConfirmDialog(null, "Continue");
```

این دستور یک پنجره محاوره‌ای (Dialog) از نوع Confirm ایجاد می‌کند که پیام "Continue" را نمایش می‌دهد.

## Confirmation Dialog (GUI)

مرحله ۵: نمایش به کاربر



این پنجره شامل سه دکمه پیشفرض Yes, No و Cancel است که کاربر می‌تواند یکی را انتخاب کند.

## Confirmation Dialog (GUI)

### مرحله ۳: مقدار بازگشتی

```
// 'option' stores the integer value corresponding to the chosen button
if (option == JOptionPane.YES_OPTION) {
    // Code for Yes case
} else if (option == JOptionPane.NO_OPTION) {
    // Code for No case
} else if (option == JOptionPane.CANCEL_OPTION) {
    // Code for Cancel case
}
```

متدهای `showConfirmDialog` یک عدد برمی‌گرداند که با ثابت‌های `YES_OPTION`، `NO_OPTION` یا `CANCEL_OPTION` مقایسه می‌شود.

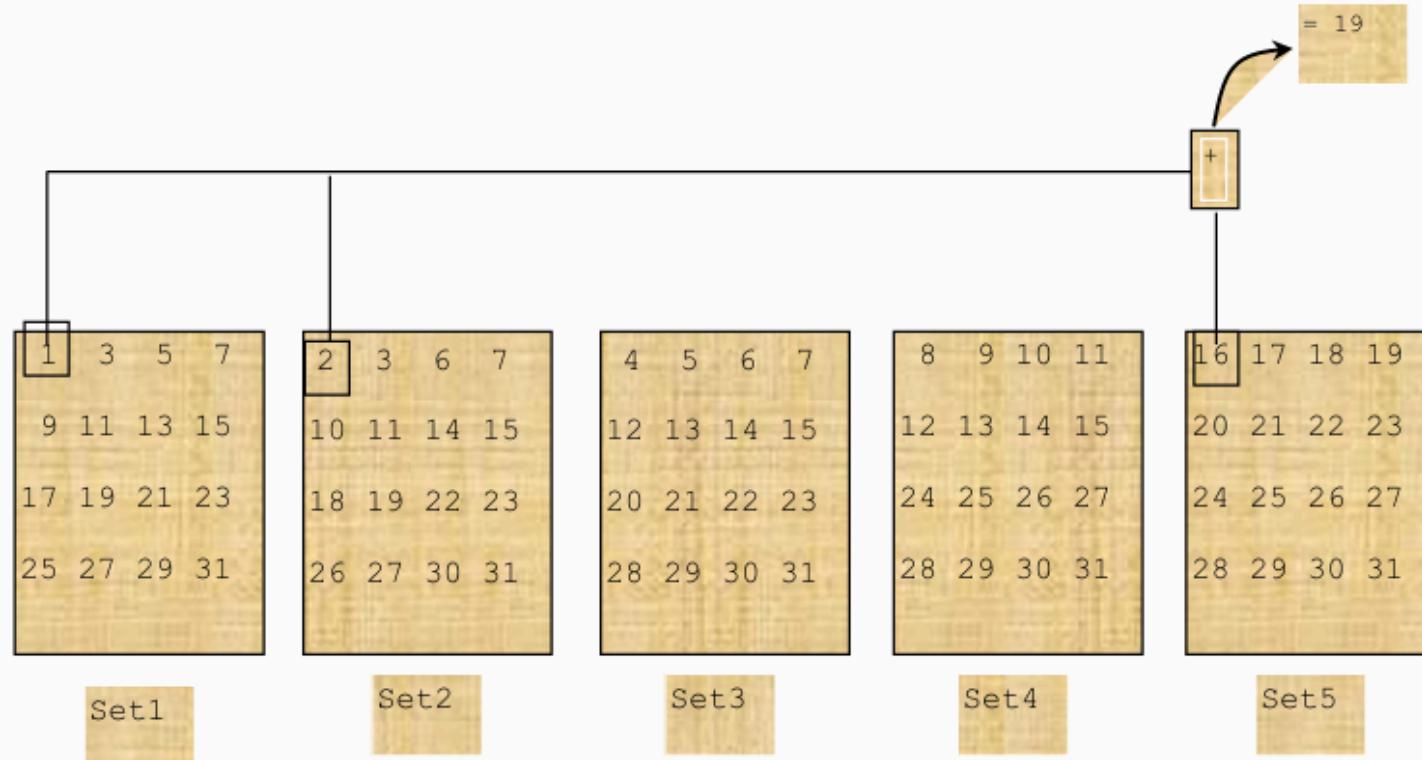
### مثال

در این برنامه، از کاربر استفاده از نمودهای تأیید (`JOptionPane.showConfirmDialog`) حند سؤال، بسیاره مهتم شد. هر سؤال شامل یک مجموعه‌ای از اعداد است که روزهای ماه را نشان می‌دهد. کاربر باید مشخص کند که تاریخ تولدش در آن مجموعه هست یا خیر.

### نکات مهم

شامل برخی روزهای ماه است (`set1` تا `set5`) هر مجموعه

- اضافه می شود (۱، ۳، ۴، ۸، ۱۶ بسته به شماره مجموعه) day بددهد، مقداری به متغیر "Yes" اگر کاربر پاسخ
- در پایان، مجموع این مقادیر روز واقعی تولد را مشخص می کند.
- روزها طراحی شده است (binary) این روش بر پایه نمایش دودویی
- به کاربر نمایش داده می شود JOptionPane.showMessageDialog نتیجه با استفاده از



```
In [ ]: import javax.swing.JOptionPane;
public class GuessBirthdayUsingConfirmationDialog {
    public static void main(String[] args) {
```

```
String set1 =
    " 1 3 5 7\n" +
    " 9 11 13 15\n" +
    "17 19 21 23\n" +
    "25 27 29 31";

String set2 =
    " 2 3 6 7\n" +
    "10 11 14 15\n" +
    "18 19 22 23\n" +
    "26 27 30 31";

String set3 =
    " 4 5 6 7\n" +
    "12 13 14 15\n" +
    "20 21 22 23\n" +
    "28 29 30 31";

String set4 =
    " 8 9 10 11\n" +
    "12 13 14 15\n" +
    "24 25 26 27\n" +
    "28 29 30 31";

String set5 =
    "16 17 18 19\n" +
    "20 21 22 23\n" +
    "24 25 26 27\n" +
    "28 29 30 31";

int day = 0;

// Ask about each set and update 'day'
int answer = JOptionPane.showConfirmDialog(null,"Is your birthday in these numbers?\n" + set1);
if (answer == JOptionPane.YES_OPTION) day += 1;

answer = JOptionPane.showConfirmDialog(null,"Is your birthday in these numbers?\n" + set2);
if (answer == JOptionPane.YES_OPTION) day += 2;

answer = JOptionPane.showConfirmDialog(null,"Is your birthday in these numbers?\n" + set3);
if (answer == JOptionPane.YES_OPTION) day += 4;
```

```

        answer = JOptionPane.showConfirmDialog(null,"Is your birthday in these numbers?\n" + set4);
        if (answer == JOptionPane.YES_OPTION) day += 8;

        answer = JOptionPane.showConfirmDialog(null,"Is your birthday in these numbers?\n" + set5);
        if (answer == JOptionPane.YES_OPTION) day += 16;

        JOptionPane.showMessageDialog(null, "Your birthday is " + day + "!");
    }
}

```

## فهرست انواع خطاهای Programming Errors

— شناسایی توسط کامپایلر — **Syntax Errors**

— باعث توقف ناگهانی برنامه می‌شود — **Runtime Errors**

— خروجی نادرست تولید می‌کند — **Logic Errors**

### Syntax Error

خطاهای نامناسب (Syntax Error) یعنی خطاها که چنانچه نهشتادی، زبان برنامه‌نویسی را از خطاها آتسویه کامپایل شناسایی می‌شوند و مانع اجراء برنامه می‌گردند. برای مثال، استفاده از متغیری که قبل از آن اعلان نشده یا فراموش کردن یک علامت «;» در پایان دستور، از رایج‌ترین Syntax Error ها هستند.

## مثال

```
public class ShowSyntaxErrors {  
    public static void main(String[] args) {  
        i = 30; // Error: variable 'i' is not declared  
        System.out.println(i + 4);  
    }  
}
```

این خطا به این دلیل است که متغیر `i` قبل از استفاده و مقداردهی [اعلان نشده](#) است.

```
public class TestError {  
    public static void main(String[] args) {  
        int number;  
        System.out.println(number + 5)  
        total = number + extra;  
        System.out.println("Done!");  
    }  
}
```

حدس بزنید چه مشکلی در این برنامه وجود دارد و در کد زیر آنها را اصلاح کنید

```
In [ ]: public class TestError {
    public static void main(String[] args) {
        int number;
        System.out.println(number + 5)
        total = number + extra;
        System.out.println("Done!");
    }
}
```

## Runtime Error

خطای زمان اجرا (Runtime Error) زمانی رخ می‌دهد که زمانیه از نظر نحوی، دستوری باشد و کامپایل آن را بدهد مشکل ترجمه کند، اما آنگاه احراری بناهه آنها، بروزد که راعت تههف آن شهد. این نه خطایها معمولاً به دلیل شرایط غیرمنتظره مثل تقسیم بر صفر، دسترسی به فایل ناموجود، یا استفاده از اندیس خارج از محدوده در آرایه رخ می‌دهند.

## مثال

```
public class ShowRuntimeErrors {
    public static void main(String[] args) {
        int i = 1 / 0; // Error: Division by zero at runtime
    }
}
```

این برنامه به درستی کامپایل می‌شود ولی هنگام اجرا، به دلیل تقسیم بر صفر متوقف می‌شود.

```
public class TestRuntimeError {  
    public static void main(String[] args) {  
        int numbers = 12;  
        int x=6;  
        number=number-2*x;  
        System.out.println(x/number);  
    }  
}
```

حدس بزندید چه مشکلی در این برنامه وجود دارد و در کد زیر آن را اصلاح کنید

```
In [ ]: public class Main {  
    public static void main(String[] args) {  
        int numbers = 12;  
        int x=6;  
        number=number-2*x;  
        System.out.println(x/number);  
    }  
}
```

Logical Error

خطا، میله‌خواست (Logical Error) نهانه را خواهد که بین این از نظر نجات داده شده باشد و نه تنها اینها شده، اما **تنها تملید شده نادست** باشد. این نوع خطا معمولاً به دلیل اشتباه در منطق محاسبه یا الگوریتم به وجود می‌آید. مثال ساده: اشتباه در اولویت عملگرها هنگام محاسبه میانگین.

## مثال

```
int average(int a, int b)
{
    return a + b / 2; // Error: should be (a + b) / 2
}
```

این برنامه به درستی کامپایل و اجرا می‌شود، ولی به دلیل اشتباه در اولویت عملگرها، نتیجه محاسبه میانگین **اشتباه** است.

```
double calcSpeed(double distance, double time) {
    return distance / time + 10;
}
```

حدس بزنید چه مشکلی در این محاسبه وجود دارد و کد را طوری اصلاح کنید که سرعت واقعی برگردانده شود.

```
In [ ]: public class TestLogicalError {
    public static void main(String[] args) {
        double distance = 100;
        double time = 2;
```

```

        double speed = calcSpeed(distance, time);
        System.out.println("Calculated speed: " + speed + " km/h");
    }

    public static double calcSpeed(double distance, double time) {
        return distance / time * 10;
    }
}

```

## خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزئی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر رو مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- چه اتفاقی رخ میدهد؟ if در صورت قرار دادن سمتی کالن(); بعد از دستور
- استفاده نکنیم، اجرا این دستور تا چه زمانی ادامه پیدا میکند؟ break اگر از switch در دستورات
- انواع خطاهای برنامه نویسی را مرور کنید و بگویید هر کدام در چه صورتی رخ میدهند و چگونه شناسایی میشوند؟

### پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من در  
gmail: nimasoltani.ce@gmail.com : ارتباط باشید  
telegram: @ni\_s11