به نام خدا

برنامه‌سازی پیشرفته

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. نوشتن چند برنامه
8. مثال های 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 برمی‌گردیم. بنابراین نیازمند ساختارهای کنترلی در زبان هستیم که اگر شرایط خاصی برقرار شد فرآیند مد نظرمون انجام بشه.

نوع بولین (Boolean)

در برنامه‌نویسی، گاهی لازم است دو مقدار را با هم مقایسه کنیم تا بفهمیم چه رابطه‌ای بین آن‌ها وجود دارد.

مثلاً می‌خواهیم بدانیم آیا یک عدد از عدد دیگر بزرگ‌تر است یا خیر.

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 نتیجه را برگردانند.

این نتیجه همیشه یک مقدار بولین است، یعنی true (درست) یا false (نادرست).

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

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

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

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

مثال

یک ابزار ساده یادگیری ریاضی

این برنامه برای آزمون ریاضی یک دانش‌آموز کلاس اولی نوشته می‌شود. دو عدد تک‌رقمی `number1` و `number2` (مثلاً ۷ و ۹) به‌طور تصادفی تولید شده و پرسش زیر را مطرح می‌کند:

What is 7 + 9?

پس از این‌که دانش‌آموز پاسخ خود را وارد کرد، پاسخ وارد شده با پاسخ واقعی مقایسه می‌شود و پیغامی مبنی بر **true** یا **false** بودن پاسخ چاپ می‌شود.

برای تولید اعداد تصادفی صحیح در بازه 0 تا 9 از دستور زیر استفاده می شود.

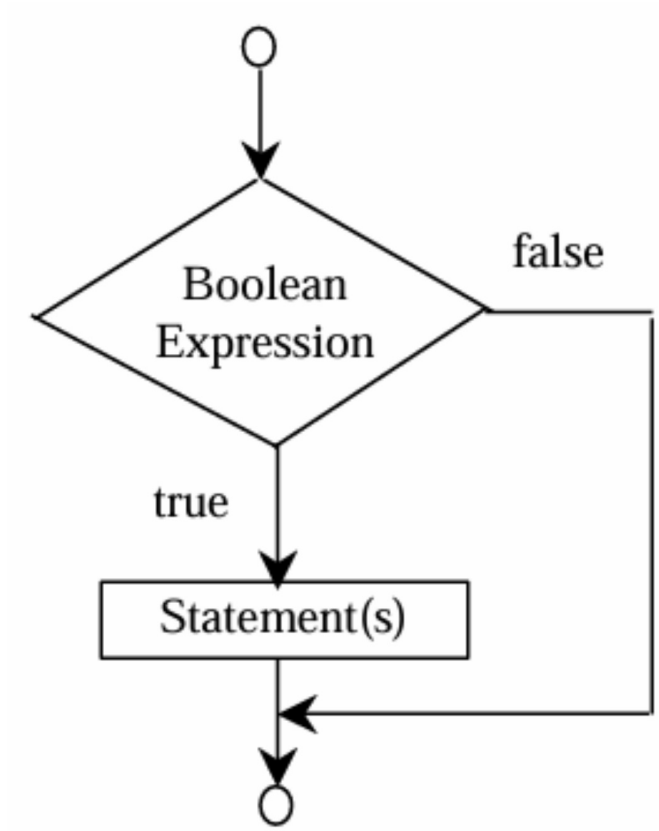
```
int num1 = (int)(Math.random() * 10); // returns a number from 0 to 9
```

```java
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);

        //Creat 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 تک‌وضعیتی (یکطرفه)

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

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

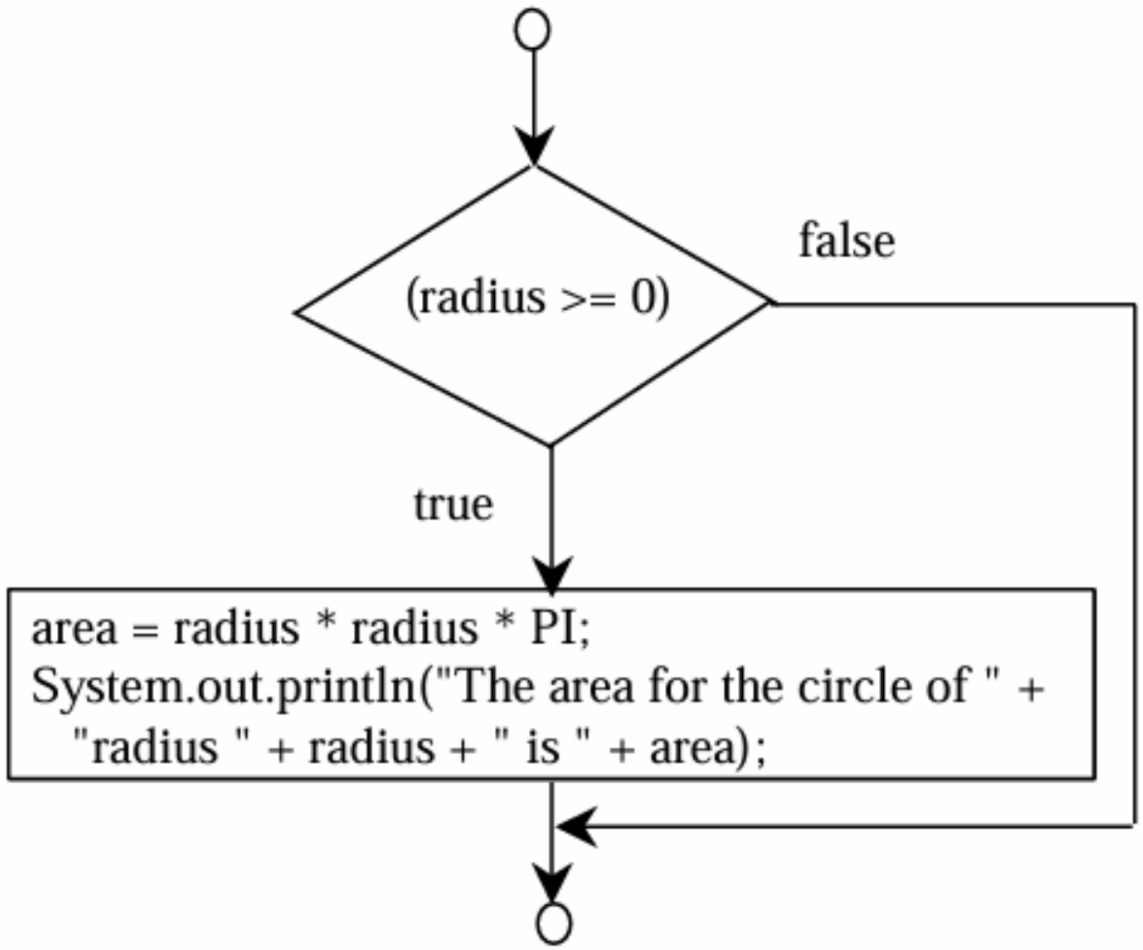


مثال

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);
}
```



```java
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

برنامه‌ای بنویسید که یک عدد صحیح را از کاربر بگیرد. اگر عدد مضرب ۵ بود، پیغام **HiFive** و اگر مضرب ۲ بود، پیغام **HiEven** چاپ کند.

```java
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.

(الف) نادرست:

نبود پرانتز باعث خطای نحوی در شرط می‌شود.

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

(ب) صحیح:

شرط داخل پرانتز و بدنه داخل آکولاد قرار گرفته است.

```
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

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

ساختار کلی:

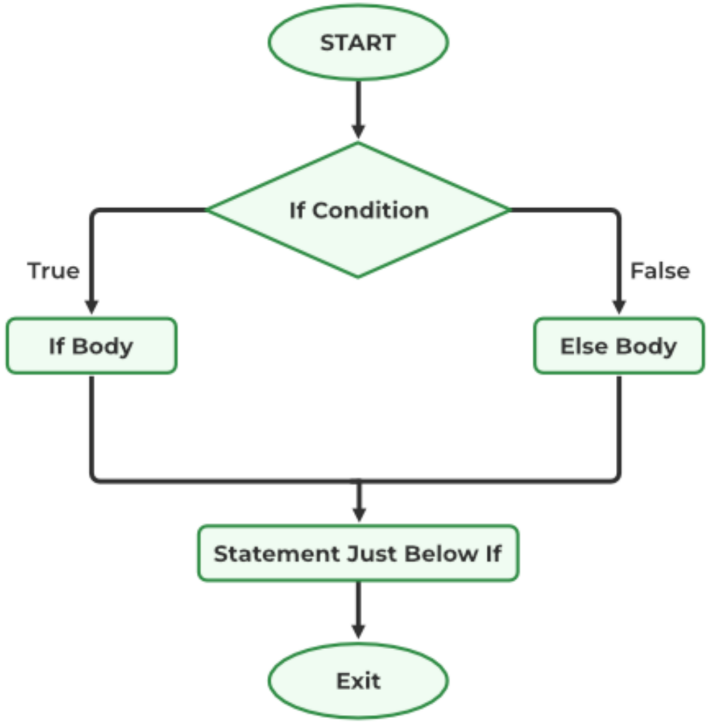
در صورت برقرار بودن شرط، بلوک اول اجرا می‌شود؛ در غیر این‌صورت، بلوک دوم.

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

مثال:

در مثال زیر، اگر مقدار grade کمتر از 60 باشد، پیام "Failed" چاپ می‌شود؛ در غیر این‌صورت، پیام "Passed".

```
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 این‌صورت، پیام "Negative input" نمایش داده شود.

```java
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';
```

```
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';
```

در روش اول از **ifهای تودرتو (nested if)** استفاده شده است؛ یعنی هر شرط درون بخش `else` شرط قبلی نوشته می‌شود و در نتیجه سطح تورفتگی (indent) افزایش می‌یابد. از نظر عملکرد، نتیجه‌ی این روش با روش **else if** کاملاً **یکسان** است، فقط شکل نگارش متفاوت است و روش دوم خواناتر است. همچنین چون در هر بخش تنها یک دستور وجود دارد، از آکولاد استفاده نشده است؛ اما اگر بیش از یک دستور در هر بخش داشتیم، باید آن‌ها را داخل آکولاد `{ }` قرار دهیم.

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 نمی شویم

```
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.

```
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.

```
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';
```

نتیجه نهایی: مقدار C به متغیر grade اختصاص می‌یابد.

```
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
```

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

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");
```

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

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 به‌درستی 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");
```

خروجی کد گفته شده در صورتی که از آکولاد استفاده کنیم به شکل زیر هست:

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

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

خطای برنامه‌نویسی

قرار دادن نقطه ویرگول (;) بلافاصله بعد از شرط `if` باعث می‌شود بدنهٔ شرط عملاً «دستور خالی» باشد و بلوک آکولادیِ بعدی یک بلوک مستقل تلقی شود که همیشه اجرا می‌شود. این خطا نه خطای نحوی است و نه در زمان اجرا استثنا می‌دهد، به همین دلیل کشف آن دشوار است و صرفاً خروجی/منطق برنامه اشتباه می‌شود.

کد اشتباه:

```
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 خطا(آنچه واقعا اجرا میشود):

```
// 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
}
```

- **دلیل بروز خطا:** نقطه ویرگول بعد از if پایان دستور شرطی را اعلام می‌کند؛ بنابراین شرط فقط ارزیابی می‌شود و هیچ بدنه‌ای ندارد. آکولادهای بعدی یک بلوک جداگانه هستند که همیشه اجرا می‌شوند (چه شرط درست باشد چه غلط).
- **راه‌حل:** نقطه ویرگول را حذف کنید و همواره برای بدنهٔ if از آکولاد استفاده کنید — حتی اگر یک دستور باشد — تا از ابهام جلوگیری شود.

ساده تر 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;
```

```
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

این برنامه دو عدد تصادفی تولید می‌کند: `number1` و `number2`. اگر `number1 > number2` باشد، یک سؤال تفریق نمایش داده می‌شود، مانند:

نکته: تضمین می‌شود خروجی حتما مثبت است.

```
What is 9 - 2?
```

دانش‌آموز پاسخ را وارد می‌کند. اگر پاسخ درست باشد، پیام **"You are correct."** نمایش داده می‌شود. در صورت اشتباه، پیام **"Your answer is wrong."** همراه با پاسخ صحیح نشان داده می‌شود.

```java
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 - number2));
    }
}
```

عملگرهای منطقی

عملگر	نام
!	Not
&&	And
||	Or
^	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	`!(score >= 90 && passed)` score = 95, passed = true	false
false	true	`!(temperature < 0 || isSnowing)` temperature = 5, isSnowing = false	true
true	false	`!(user.isLoggedIn && user.role == "admin")` user = {isLoggedIn: true, role: "admin"}	false
false	true	`!(balance > 0 && account.isActive)` 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 = 14, gender = 'F'	false
false	true	false	`(1 > 2) && (6 > 5)` 1 > 2 is false, 6 > 5 is true	false
true	false	false	`(age > 18) && (gender != 'F')` age = 24, gender = 'F'	false
true	true	true	`(3 > 2) && (5 >= 5)`	true

عملگر && فقط وقتی نتیجه‌اش true می‌شود 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
true	true	true	`(5 > 3) || (7 > 5)`	true

عملگر || وقتی نتیجه‌اش true می‌شود که حداقل یکی از شرط‌ها درست باشد. فقط وقتی هر دو شرط نادرست باشند، نتیجه false خواهد بود.

مثال

جابجایی دو عدد با استفاده از عملگر XOR

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

به عنوان مثال، اگر ورودی‌ها به صورت زیر باشند:
عدد اول (a) = ۵
عدد دوم (b) = ۶
پس از اجرای برنامه، خروجی باید به صورت زیر باشد:
a = ۶
b = ۵

پرسش: پس از خواندن کد بررسی کنید که عملیات انجام شده چگونه باعث جابه‌جایی دو عدد شده است؟

```java
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

برنامه‌ای بنویسید که عددی را از کاربر دریافت کند و بررسی کند آیا:

۱. بر ۲ و ۳ بخش‌پذیر است.
۲. بر ۲ یا ۳ بخش‌پذیر است.
۳. بر ۲ یا ۳ اما نه بر هر دو بخش‌پذیر است.

```java
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:
۱. بر ۴ بخش‌پذیر باشد اما بر ۱۰۰ بخش‌پذیر نباشد.
۲. یا اینکه بر ۴۰۰ بخش‌پذیر باشد.

```java
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);
    }
}
```

عملگر بیتی & (AND)

عملگر & در سطح **بیتی (Bitwise)** عمل می‌کند و هر بیت از دو عدد را با هم مقایسه می‌کند. اگر هر دو بیت مقدار **۱** داشته باشند، نتیجهٔ آن بیت نیز **۱** می‌شود؛ در غیر این صورت مقدار آن **۰** خواهد بود.

مثال:

```
int a = 5;   // 0101
int b = 3;   // 0011
int result = a & b; // Bitwise AND
System.out.println(result);
```

خروجی:

```
// 0101
// & 0011
// = 0001 → 1
1
```

- **توضیح:** تنها بیت‌هایی که در هر دو عدد مقدار **۱** دارند، در نتیجه باقی می‌مانند.
- **کاربرد:** استفاده برای مقایسه یا فیلتر کردن بیت‌ها در عملیات سطح پایین (مثلاً بررسی بیت فعال در پرچم‌ها).

عملگر بیتی | (OR)

عملگر | نیز در سطح **بیتی (Bitwise)** دو عدد را مقایسه می‌کند. اگر حداقل یکی از دو بیت مقدار ۱ داشته باشد، نتیجهٔ آن بیت ۱ می‌شود. تنها در صورتی که هر دو بیت ۰ باشند، مقدار نتیجه ۰ خواهد بود.

مثال:

```
int a = 5;   // 0101
int b = 3;   // 0011
int result = a | b; // Bitwise OR
System.out.println(result);
```

خروجی:

```
// 0101
// | 0011
// = 0111 → 7
7
```

- **توضیح:** اگر یکی از بیت‌ها ۱ باشد، خروجی آن بیت نیز ۱ خواهد بود.
- **کاربرد:** برای ترکیب بیت‌ها یا فعال‌سازی چند پرچم به‌صورت هم‌زمان در عملیات بیتی.

مثال

بررسی شرایط با عملگرهای & و |

این برنامه نمونه‌ای از استفاده عملگرهای منطقی & و | در عبارات شرطی را نشان می‌دهد.
۱. شرط اول بررسی می‌کند که آیا مقدار gender برابر ۱ و مقدار age بزرگ‌تر یا مساوی ۶۵ است.
۲. شرط دوم بررسی می‌کند که آیا birthday برابر true است یا پس از یک واحد اضافه شدن به age، مقدار آن بزرگ‌تر یا مساوی ۶۵ می‌شود.

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.

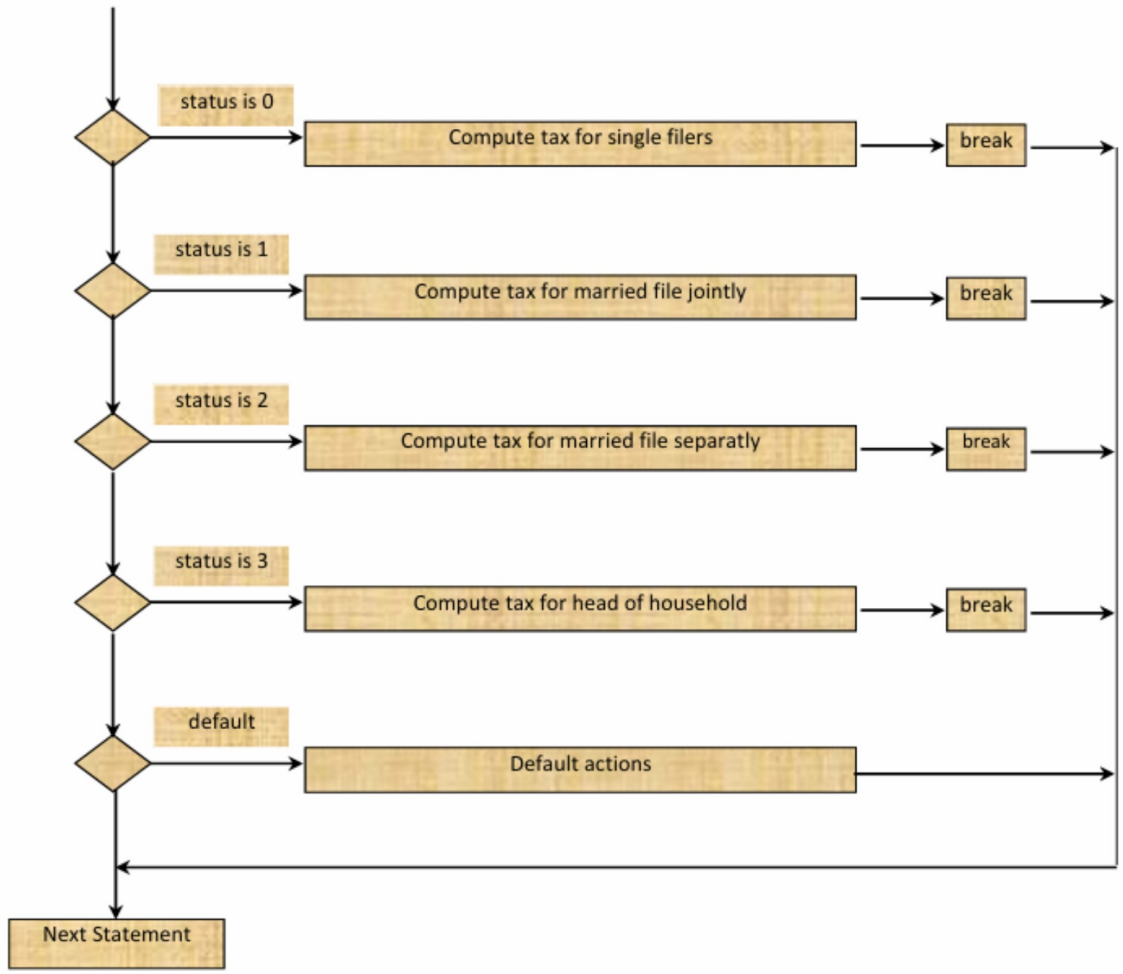
```
( gender == 1 ) & ( age >= 65 )
( birthday == true ) | ( ++age >= 65 )
```

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

معرفی دستور switch

فلوچارت تصمیم‌گیری بر اساس وضعیت‌های مختلف

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



قواعد دستور switch

ساختار و شرایط استفاده از دستور switch

عبارت switch-expression می‌تواند حاوی یک مقدار از نوع **int**، **short**، **byte**، **char** یا **string** باشد که درون پرانتز بعد از کلمهٔ کلیدی `switch` قرار می‌گیرد.

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

```
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 break و default در کنترل جریان

کلمهٔ کلیدی `break` اختیاری است، اما معمولاً در انتهای هر case استفاده می‌شود تا از اجرای ناخواستهٔ case بعدی جلوگیری کند.

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

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

```
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

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

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

ردگیری دستور switch

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

چون در `case a` قبلی `break` وجود نداشت، بنابراین عبارت مربوط به این کیس نیز اجرا می‌شود و این روند تا رسیدن به یک `break` یا پایان `switch` ادامه پیدا می‌کند.

ردگیری دستور switch

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

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

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;
```

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

ردگیری دستور switch

```
// suppuse '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);
}
```

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

اجرای case انتخاب‌شده

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

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

اجرای break

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

دستور `break` اجرا می‌شود و باعث خروج از ساختار `switch` می‌گردد.

ادامهٔ 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;
```

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

ادامهٔ 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;
```

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

عملگر شرطی ?:

عملگر شرطی ? : یک بیان تک‌خطی برای تصمیم‌گیری است: اگر شرط برقرار باشد مقدار اول، وگرنه مقدار دوم برگردانده می‌شود. این عملگر یک «بیان» است، نابراین نتیجه عبارت حاوی این عملگر باید در یک متغیر ریخته شود یا در جای مناسب مثل نمایش در خروجی (println) استفاده شود.

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

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

مثال

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

```
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، به‌طوری که همه شرط‌ها در یک عبارت بررسی شوند.

```
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

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.

هدف این است که این مسئله را ابتدا با `if/else` و سپس با استفاده از عملگر شرطی `?:` حل کنیم تا تفاوت و شباهت‌ها مشخص شود.

```java
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();
    }
}
```

```java
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

در زبان جاوا می‌توان با استفاده از دستور `System.out.printf(format, items)` خروجی را به شکل دلخواه قالب‌بندی کرد. این روش باعث می‌شود متن و مقادیر متغیرها با نظم و ظاهر مشخصی نمایش داده شوند.

بخش **format** رشته‌ای است که از ترکیب زیررشته‌ها و تعیین‌گرهای فرمت تشکیل شده است. هر **تعیین‌گر فرمت (format specifier)** با علامت `%` شروع می‌شود و نوع و نحوه نمایش یک آیتم را مشخص می‌کند. این آیتم می‌تواند یک مقدار عددی، کاراکتر، مقدار بولین یا رشته باشد.

تعیین‌گرهای پرکاربرد در printf

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

مثال:

یک عدد صحیحی و یک عدد اعشاری چاپ شود:

```
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

در جاوا، **عملگرهای بیتی** مجموعه‌ای از عملگرها هستند که عملیات خود را بر روی داده‌های عددی در سطح **بیت** انجام می‌دهند. این عملگرها تنها بر روی انواع داده صحیح مانند `char`، `short`، `long`، `int` و `byte` قابل استفاده‌اند. تفاوت اصلی آن‌ها با عملگرهای منطقی (Logical) این است که نتیجه را بر اساس مقایسه و ترکیب بیت‌ها به دست می‌آورند، نه بر اساس درست و نادرست بودن یک شرط.

هر عدد در حافظه به صورت یک رشته باینری (0 و 1) نمایش داده می‌شود. عملگرهای بیتی این رشته باینری را بیت به بیت بررسی کرده و خروجی را بر اساس قوانین خود ایجاد می‌کنند.

مثال

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

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`
`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`

عملگرهای بیتی (Bitwise Operators) در جاوا

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

عملگر	نام	توضیح	مثال
&	AND	هر بیت فقط وقتی ۱ می‌شود که در هر دو عدد همان بیت برابر ۱ باشد (مثل لامپی که باید دو کلیدش همزمان روشن باشند)	(A & B) → 12 (0000 1100)
|	OR	بیت ۱ می‌شود اگر حداقل یکی از دو بیت همان موقعیت ۱ باشد (مثل لامپی که با روشن کردن یکی از دو کلیدش روشن می‌شود)	(A | B) → 61 (0011 1101)
^	XOR	بیت ۱ می‌شود اگر دقیقا یکی از دو بیت همان موقعیت ۱ باشد (مثل سوییچ انتخاب که فقط یک طرف باید فعال باشد)	(A ^ B) → 49 (0011 0001)
~	NOT	تمام بیت‌ها معکوس می‌شوند (۰ها به ۱ و ۱ها به ۰ مثل نگاتیو عکس)	(~A) → -61 (1100 0011)
<<	Left Shift	بیت‌ها را به چپ می‌برد و صفر اضافه می‌کند (هر شیفت چپ یعنی ضرب در ۲)	A << 2 → 240 (1111 0000)
>>	Right Shift	بیت‌ها را به راست می‌برد و علامت را حفظ می‌کند (هر شیفت راست یعنی تقسیم بر ۲)	A >> 2 → 15 (0000 1111)
>>>	Unsigned Right Shift	مثل شیفت راست، ولی جای خالی همیشه صفر می‌آید (علامت بی‌اثر می‌شود)	A >>> 2 → 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. ! : عملگر منفی منطقی (NOT)
4. ~ : مکمل بیت‌ها (bitwise NOT)
5. (type) : تبدیل نوع داده (Casting)
6. *, /, % : ضرب، تقسیم و باقیمانده
7. +, - : جمع و تفریق
8. <<, >>, >>> : شیفت بیت‌ها به چپ یا راست
9. <, <=, >, >= : مقایسه کوچکتر، بزرگتر و مساوی
10. ==, != : بررسی برابری یا نابرابری
11. ^ : یا منطقی انحصاری (XOR)
12. & : و منطقی یا بیت‌به‌بیت (AND)
13. ^ : یا منطقی انحصاری بیت‌به‌بیت (XOR)
14. | : یا منطقی یا بیت‌به‌بیت (OR)
15. && : و شرطی (AND کوتاه‌شده)
16. || : یا شرطی (OR کوتاه‌شده)
17. ?: : عملگر شرطی (سه‌تایی)
18. =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>= : عملگرهای انتساب
```

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

تقدم عملگرها و شرکت‌پذیری:

- شرکت‌پذیری همه عملگرها دوعملوندی (به جز انتساب) از چپ به راست می‌باشد.
- شرکت‌پذیری عملگرها تک‌عملوندی افزایشی (پیشوندی و پسوندی) از راست به چپ می‌باشد.
- شرکت‌پذیری عملگرها شرطی `:?` از راست به چپ می‌باشد.

نکته درباره عملگر تقسیم در جاوا:

```
3 / 2
```

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

```
3.0 /2
```

چون یکی از عملوندها اعشاری است، تقسیم به‌صورت 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 به‌صورت 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 می‌شود.

مرحله ۵: تفریق

```
19 > 35 - 1
```

سپس تفریق اجرا می‌شود.

مرحله ۶: مقایسه

```
19 > 34
```

مقایسه انجام می‌شود و نتیجه `false` است.

Confirmation Dialog (GUI)

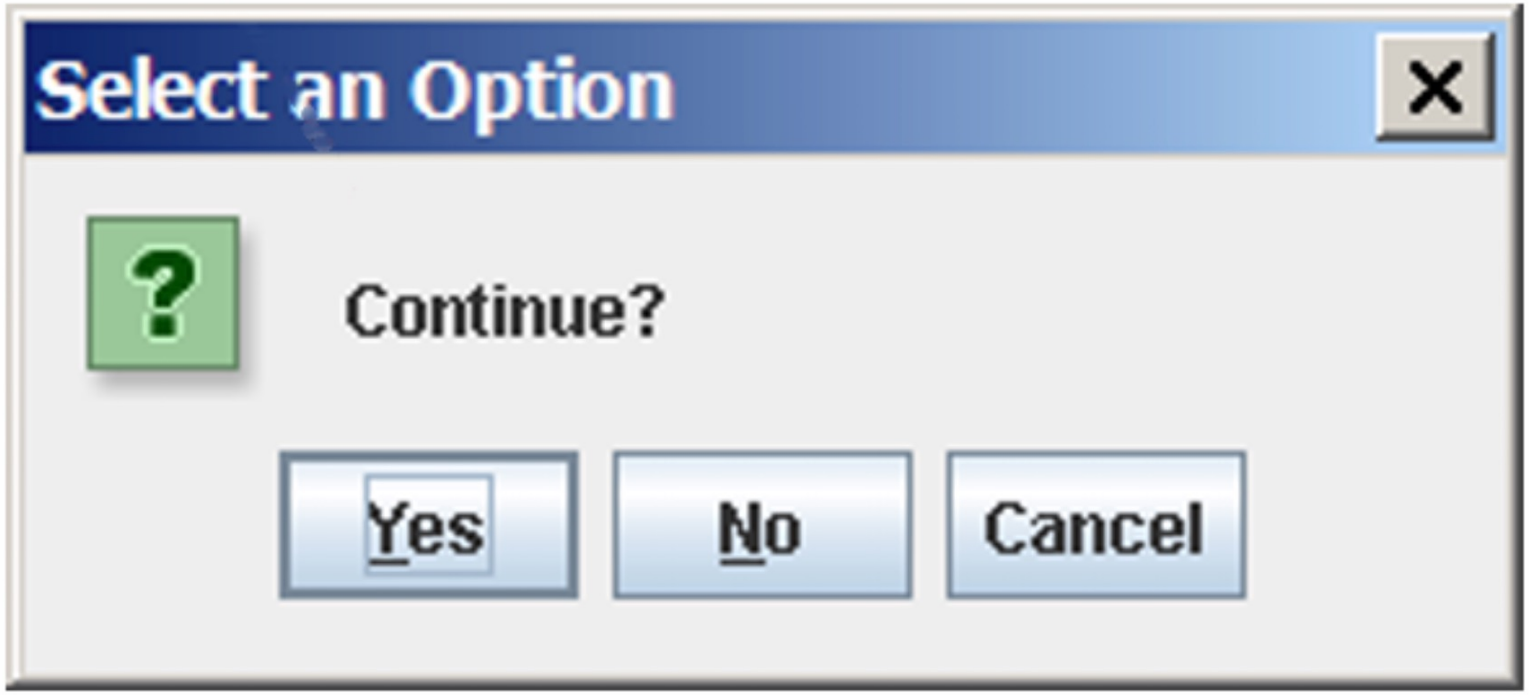
مرحله ۱: ایجاد دیالوگ تأیید

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

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

Confirmation Dialog (GUI)

مرحله ۲: نمایش به کاربر



این پنجره شامل سه دکمهٔ پیش‌فرض **No**، **Yes** 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
}
```

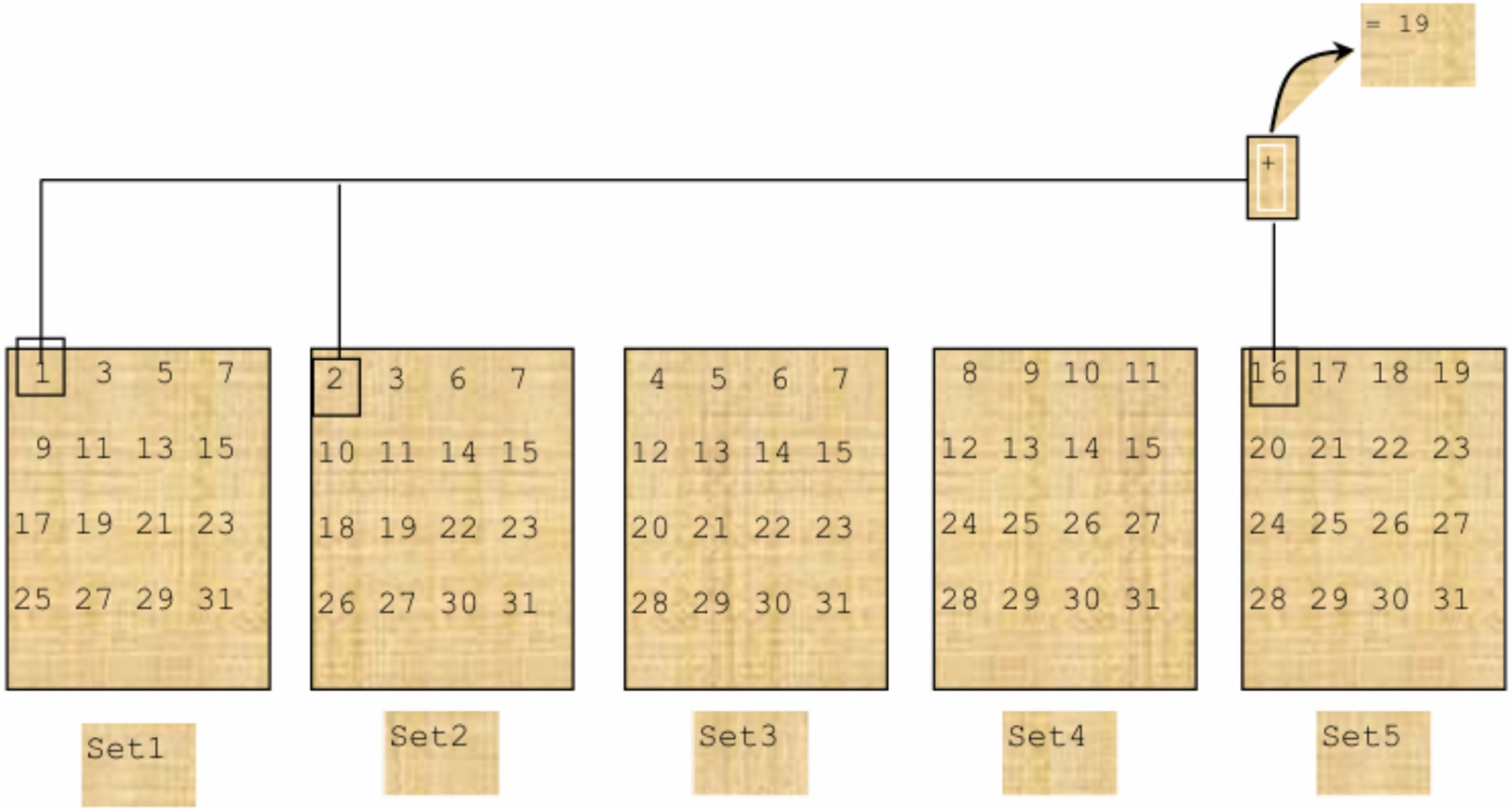
متد `showConfirmDialog` یک عدد برمی‌گرداند که با ثابت‌های `NO_OPTION`، `YES_OPTION` یا `CANCEL_OPTION` مقایسه می‌شود.

مثال

در این برنامه، از کاربر با استفاده از پنجره‌های تأیید (`JOptionPane.showConfirmDialog`) چند سؤال پرسیده می‌شود. 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

- هر مجموعه (`set1` تا `set5`) شامل برخی روزهای ماه است.
- اگر کاربر پاسخ "Yes" بدهد، مقداری به متغیر `day` اضافه می‌شود (۱، ۲، ۴، ۸، ۱۶ بسته به شماره مجموعه).
- در پایان، مجموع این مقادیر روز واقعی تولد را مشخص می‌کند.
- این روش بر پایه نمایش دودویی (binary) روزها طراحی شده است.
- نتیجه با استفاده از `JOptionPane.showMessageDialog` به کاربر نمایش داده می‌شود.


= 19
+
1 3 5 7
9 11 13 15
17 19 21 23
25 27 29 31
2 3 6 7
10 11 14 15
18 19 22 23
26 27 30 31
4 5 6 7
12 13 14 15
20 21 22 23
28 29 30 31
8 9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
Set1
Set2
Set3
Set4
Set5

```java
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";
```

```java
        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 خطاها - Programming Errors

- **Syntax Errors** — شناسایی توسط کامپایلر
- **Runtime Errors** — باعث توقف ناگهانی برنامه می‌شود
- **Logic Errors** — خروجی نادرست تولید می‌کند

Syntax Error

خطای نحوی (Syntax Error) زمانی رخ می‌دهد که قوانین نوشتاری زبان برنامه‌نویسی رعایت نشده باشد. این خطاها توسط **کامپایلر** شناسایی می‌شوند و مانع اجرای 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

Runtime Error

خطای زمان اجرا (Runtime Error) زمانی رخ می‌دهد که برنامه از نظر نحوی درست باشد و کامپایلر آن را بدون مشکل ترجمه کند، اما هنگام اجرای برنامه اتفاقی بیفتد که باعث توقف آن شود. این نوع خطاها 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 به‌درستی 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 بزنید چه مشکلی در این برنامه وجود دارد و در کد زیر آن را اصلاح کنید

Logical Error

خطای منطقی (Logical Error) زمانی رخ می‌دهد که برنامه از نظر نحوی درست باشد و بدون خطا اجرا شود، اما **نتیجهٔ تولیدشده نادرست** 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 به‌درستی کامپایل و اجرا می‌شود، ولی به دلیل اشتباه در اولویت عملگرها، نتیجهٔ 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.

خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی 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.

- درصورت قرار دادن نقطه ویرگول (;) بعد از دستور if چه اتفاقی رخ میدهد؟
- در دستورات switch اگر از break استفاده نکنیم،اجرا این دستور تا چه زمانی ادامه پیدا میکند؟
- انواع خطا های 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 میتونید با من در ارتباط باشید :)

gmail: nimasoltani.ce@gmail.com

telegram: @ni_s11