به نام خدا

برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی · دانشکده مهندسی کامپیوتر

دکتر مجتبی وحیدی اصل

آشنایی با متد ها

آریا شاکو

فهرست مطالب

1. آشنایی با متدها
2. استفاده مجدد از متدها و ماژوله بندی کد
3. سربارگذاری متدها (Method Overloading)
4. حوزه متغیرهای محلی
5. انتزاع متدها
6. متدهای مثلثاتی و توان
7. متدهای روندسازی
8. متدهای ریاضی

مقدمه

در این بخش با یکی از مهم‌ترین مفاهیم برنامه‌نویسی در جاوا یعنی **متدها (Methods)** آشنا می‌شوید. متدها بلوک‌هایی از کد هستند که برای انجام کارهای مشخص نوشته می‌شوند و کمک می‌کنند برنامه‌های شما خواناتر، قابل نگهداری‌تر و انعطاف‌پذیرتر شوند.

با استفاده از متدها می‌توانید کدهای تکراری را حذف کنید، برنامه خود را ماژوله و ساخت‌یافته‌تر بنویسید و از قابلیت‌های مهمی مثل سربارگذاری و انتزاع استفاده کنید. در این فصل، از تعریف و انگیزه متدها شروع می‌کنیم، سپس سراغ امضا، پارامترها، نوع مقدار برگشتی، فراخوانی و حوزه متغیرها در متدها می‌رویم و در ادامه با مفاهیمی مانند سربارگذاری متدها، متدهای آماده جاوا (ریاضی و مثلثاتی)، و ماژوله‌بندی کد آشنا خواهید شد.

هدف این فصل این است که با کاربرد متدها در برنامه‌های واقعی و نحوه تعریف و استفاده صحیح از آن‌ها، آشنا شوید و بتوانید از این ابزار قدرتمند برای حل مسائل پیچیده‌تر بهره ببرید. تمرین و کدنویسی فعال، کلید موفقیت شما در این بخش است.

چرا به متدها نیاز داریم؟

متدها ابزاری ضروری در برنامه‌نویسی هستند که به ما این امکان را می‌دهند تا کدهای خود را سازماندهی کرده و بخش‌های مختلفی از برنامه را از یکدیگر جدا کنیم. این کار به برنامه‌نویس کمک می‌کند تا کد را خواناتر، قابل نگهداری‌تر و استفاده مجدد از آن را ساده‌تر کند.

به عنوان مثال، فرض کنید می‌خواهید مجموع اعداد یک بازه خاص را محاسبه کنید. اگر هر بار این کار را به صورت دستی انجام دهید، کد شما به سرعت طولانی و پیچیده می‌شود. اما با استفاده از متدها می‌توانید یکبار این عملیات را تعریف کرده و هر جا که نیاز به محاسبه مجموع داشتید، فقط متد را فراخوانی کنید.

```
int sum = 0;
for (int i = 1; i <= 10; i++) sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++) sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++) sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

همانطور که مشاهده می‌کنید، این کد چندین بار عملیات مشابهی را انجام می‌دهد و تکرار این عملیات نه تنها باعث پیچیده‌تر شدن برنامه می‌شود، بلکه امکان بروز خطا را نیز افزایش می‌دهد. اینجاست که استفاده از متدها به کمک می‌آید تا کدهای تکراری حذف شوند و کارایی برنامه بهبود یابد.

```
public static int sum(int i1, int i2) {
    int sum = 0;
    for (int i = i1; i <= i2; i++) sum += i;
    return sum;
}

public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 30 is " + sum(20, 30));
    System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
```

با استفاده از متدها، کدهای مشابهی که قبلاً چندین بار تکرار می‌شد، اکنون تنها با فراخوانی یک متد انجام می‌شود. این کار باعث کاهش پیچیدگی برنامه و افزایش قابلیت نگهداری آن می‌شود.

تعریف متدها

متد (Method) در جاوا به مجموعه‌ای از دستورات گفته می‌شود که برای انجام یک کار خاص، با یک نام مشخص و قابل فراخوانی در برنامه نوشته می‌شود. متدها امکان تکرارپذیری کد، ماژوله‌سازی و ساده‌سازی برنامه را فراهم می‌کنند. به کمک متدها، می‌توانید یکبار عملیاتی را بنویسید و بارها با پارامترهای مختلف از آن استفاده کنید؛ به این ترتیب، از تکرار بی‌مورد کد جلوگیری می‌شود و تغییر و نگهداری برنامه نیز ساده‌تر خواهد بود.

- هر متد معمولاً یک هدف مشخص دارد (مثلاً جمع دو عدد، پیدا کردن بزرگ‌ترین مقدار، یا چاپ پیام).
- متدها می‌توانند ورودی داشته باشند (پارامتر) و/یا خروجی (مقدار برگشتی) تولید کنند.
- بدنه متد شامل دستورات اجرایی است که با فراخوانی متد اجرا می‌شوند.

```java
public static int sum(int start, int end) {
  int total = 0;
  for (int i = start; i <= end; i++)
    total += i;
  return total;
}
```

در مثال بالا، متد دو عدد به عنوان ورودی می‌گیرد و مجموع تمام اعداد بین آن دو را محاسبه و بازمی‌گرداند. این متد را می‌توانید هر تعداد بار که خواستید با ورودی‌های متفاوت صدا بزنید.

امضای متد

امضای متد (Method Signature) در جاوا فقط شامل **نام متد** و **تعداد و نوع پارامترهای آن** است. نوع مقدار برگشتی، بخش امضا محسوب نمی‌شود. اهمیت امضای متد در این است که کامپایلر به کمک آن متدهای مشابه را (مثلاً در سربارگذاری) از هم تشخیص می‌دهد.

- متدهایی با نام یکسان اما امضای متفاوت (تعداد/نوع پارامترها) می‌توانند در یک کلاس وجود داشته باشند.
- تغییر فقط نوع مقدار برگشتی باعث تغییر امضا نمی‌شود و کامپایلر آن را به عنوان متد جدید شناسایی نمی‌کند.

```
public static int sum(int a, int b) { ... }

public static double sum(double a, double b) { ... }
```

در این مثال، دو متد با نام یکسان `sum` داریم، اما امضای آن‌ها به دلیل تفاوت در نوع پارامترها متفاوت است و هر دو می‌توانند در یک کلاس قرار بگیرند.

پارامترهای فرمال و واقعی

پارامترهای فرمال (Formal Parameters) پارامترهایی هستند که هنگام تعریف متد در داخل پرانتز ذکر می‌شوند و نقش متغیرهای محلی را دارند.
پارامترهای واقعی یا **آرگومان‌ها** (Actual Parameters) مقادیری هستند که هنگام فراخوانی متد به آن ارسال می‌شوند.

- هر بار که متد صدا زده می‌شود، مقادیر آرگومان‌ها به پارامترهای فرمال کپی می‌شوند و متد با همان مقادیر اجرا می‌شود.
- پارامترهای فرمال فقط در داخل بدنه متد قابل استفاده هستند و بیرون از آن وجود ندارند.
- تعداد و نوع آرگومان‌ها باید با پارامترهای فرمال متد مطابق باشد.

```java
public class MainClass {

    public static void greet(String name) {

        System.out.println("سلام " + name + "!");
    }

    public static void main(String[] args) {
        // فراخوانی متد با پارامتر واقعی
        greet("علی");   // مقدار "علی" آرگومان واقعی است
    }
}
```

در این مثال، متد `greet` یک پارامتر فرمال به نام `name` دارد که هنگام فراخوانی مقدار `"علی"` به آن اختصاص داده می‌شود.

مثالی دیگر از پارامترهای واقعی و فرمال در متد ریاضی:

```java
int maxNum = max(10, 20); // 10 و 20 آرگومان واقعی هستند
public static int max(int a, int b) { // a و b پارامترهای فرمال هستند
  return (a > b) ? a : b;
}
```

نوع مقدار برگشتی متدها

هر متد می‌تواند مقداری را به برنامه بازگرداند که نوع آن قبل از نام متد ذکر می‌شود. اگر متد هیچ مقداری بازنمی‌گرداند (مثلاً فقط عملی را انجام می‌دهد)، نوع برگشتی آن void باید باشد.

- متدهای غیر void باید حتماً با دستور return مقداری از نوع تعیین شده را بازگردانند.
- اگر نوع برگشتی void باشد، وجود دستور return اختیاری است (فقط برای پایان دادن زودهنگام متد قابل استفاده است).

```
public static double circleArea(double radius) {
  return Math.PI * radius * radius;
}

public static void printHello() {
  System.out.println("Hello world!");
}
```

متد `add` مقدار بازگشتی از نوع `int` دارد و متد `printSum` مقدار برگشتی ندارد و فقط پیام چاپ می‌کند.

توجه: اگر متد غیر void باشد و برای همه مسیرهای ممکن مقدار بازگشتی تعریف نشده باشد، برنامه دچار خطای کامپایل می‌شود.

فراخوانی متدها

برای استفاده از متدها، کافی است نام متد را همراه با آرگومان‌های مناسب بنویسید. اگر متد مقدار برمی‌گرداند، می‌توانید نتیجه را در یک متغیر ذخیره کنید یا مستقیماً آن را استفاده کنید. همچنین می‌توانید متدهای بدون مقدار بازگشتی (void) را تنها با نامشان و آرگومان‌های لازم صدا بزنید.

- در صورتی که متد در همان کلاس باشد، کافی است نام متد را بنویسید. در غیر این صورت، باید با `ClassName.methodName` فراخوانی شود.
- در متد `main`، معمولاً متدهای دیگر صدا زده می‌شوند تا کد مرتب‌تر و قابل فهم‌تر شود.

```
public static void main(String[] args) {
  int s = sum(1, 100);
  System.out.println("مجموع از ۱ تا ۱۰۰: " + s);
  printHello();
}
```

در این مثال، متد `sum` با پارامترهای ۱ و ۱۰۰ فراخوانی شده و نتیجه‌اش در متغیر `s` ذخیره شده است. همچنین متد `printHello` که void است، بدون مقدار برگشتی فراخوانی شده است.

اگر متد در کلاس دیگری قرار دارد (مثلاً `TestMath.sum()`)، می‌توانید به شکل زیر فراخوانی کنید:

```
int result = Math.max(7, 12);
```

ردگیری فراخوانی متد

وقتی یک متد فراخوانی می‌شود، برنامه کنترل اجرا را به آن متد منتقل می‌کند. پارامترهای واقعی به پارامترهای فرمال متد کپی می‌شوند. پس از اتمام دستورات متد و اجرای دستور `return` (یا رسیدن به انتهای متد void)، کنترل برنامه به نقطه‌ای که متد از آن صدا زده شده برمی‌گردد.

- در حافظه، هر بار که متد فراخوانی می‌شود، یک "بلاک" مخصوص آن روی پشته (stack) ساخته می‌شود تا متغیرهای محلی و پارامترهای آن جداگانه نگهداری شوند.
- پس از پایان کار متد، این بلاک حذف می‌شود و مقدار برگشتی (در صورت وجود) جایگزین فراخوانی متد می‌شود.
- می‌توان با ردگیری اجرای برنامه (step by step) روند مقداردهی پارامترها و مقدار برگشتی را دنبال کرد.

```
public static void main(String[] args) {
  int i = 5, j = 8;
  int bigger = max(i, j);
  System.out.println("بزرگتر بین " + i + " و " + j + " عدد " + bigger + " است.");
}
```

```
 public static int max(int a, int b) {
    if (a > b)
      return a;
    else
      return b;
 }
```

گام به گام اجرای مثال:

1. برنامه از `main` شروع می‌شود. متغیرهای `i` و `j` مقداردهی می‌شوند.
2. متد `max(i, j)` فراخوانی می‌شود؛ مقادیر ۵ و ۸ به پارامترهای `a` و `b` کپی می‌شوند.
3. در داخل متد `max` شرط `a > b` بررسی می‌شود (در اینجا ۵ < ۸ که غلط است)، پس مقدار `b` (یعنی ۸) بازگردانده می‌شود.
4. کنترل برنامه به `main` برمی‌گردد و مقدار برگشتی متد در متغیر `bigger` قرار می‌گیرد.
5. در نهایت خروجی چاپ می‌شود.

این مکانیزم باعث می‌شود هر متد کاملاً مستقل و با متغیرهای محلی خودش کار کند و تداخلی با دیگر بخش‌های برنامه نداشته باشد.

استفاده مجدد از متدها و ماژوله‌بندی کد

یکی از مزیت‌های مهم متدها، **امکان استفاده‌ی مجدد** از کد است. کافیست متدی را یکبار بنویسید و در بخش‌های مختلف برنامه، یا حتی برنامه‌های دیگر، هر زمان نیاز داشتید فقط آن را فراخوانی کنید. این ویژگی باعث می‌شود برنامه‌ها ماژوله (قطعه‌قطعه) و ساختارمند باشند، رفع اشکال و گسترش آن‌ها هم بسیار راحت‌تر می‌شود.

استفاده مجدد از متدها از کلاس‌های دیگر

متدهایی که به صورت `public static` در یک کلاس تعریف شده‌اند، از هر کلاس دیگری هم قابل فراخوانی هستند. فقط کافیست نام کلاس را قبل از نام متد بنویسید.
به مثال زیر توجه کنید:

```
public class MathUtil {
  public static int square(int n) {
    return n * n;
  }
}
```

```
// فراخوانی متد از کلاس دیگر
int result = MathUtil.square(5);  // 25 :خروجی
```

در این مثال، متد `square` فقط یکبار نوشته شده اما می‌تواند هرجا در برنامه یا حتی پروژه‌های دیگر فراخوانی شود. این همان ماژوله‌بندی است: کدها را به بلوک‌های قابل‌استفاده و مستقل تقسیم کنیم.

هدف اصلی ماژوله‌بندی، تقسیم یک مسئله بزرگ به بخش‌های کوچکتر (ماژول) است. هر ماژول معمولاً یک کار خاص انجام می‌دهد و هر زمان نیاز باشد، می‌توانیم آن را جداگانه تغییر یا تست کنیم.

```
public class StringUtil {
      public static String reverse(String s) {
        String result = "";
        for (int i = s.length() - 1; i >= 0; i--) {
          result += s.charAt(i);
        }
        return result;
      }
}
```

```
String reversed = StringUtil.reverse("ChatGPT"); // خروجی: TPGtahC
```

اینجا هم ماژوله‌بندی باعث شده متد `reverse` را در هرجای برنامه فقط با یک خط صدا بزنیم و نیاز نباشد کد معکوس کردن رشته را هر بار بنویسیم.

سربارگذاری متدها (Method Overloading)

یکی از قابلیت‌های مهم و کاربردی در جاوا **سربارگذاری متدها** (Overloading) است.سربارگذاری به این معنی است که می‌توان چندین متد با نام یکسان اما با **امضای متفاوت** (تعداد، نوع یا ترتیب پارامترها) در یک کلاس داشت.

سربارگذاری به این معنی است که می‌توان چندین متد با نام یکسان اما با **امضای متفاوت** (تعداد، نوع یا ترتیب پارامترها) در یک کلاس داشت.

این قابلیت باعث می‌شود بتوانید یک عملیات را با ورودی‌های مختلف پیاده‌سازی کنید و برنامه‌تان تمیزتر و خواناتر باشد.

مفهوم سربارگذاری متدها

- امضاهای متد باید متفاوت باشند (نوع/تعداد/ترتیب پارامترها).
- تغییر فقط نوع مقدار برگشتی باعث سربارگذاری نمی‌شود و خطای کامپایل می‌دهد.
- انتخاب متد مناسب هنگام فراخوانی، به صورت خودکار توسط کامپایلر براساس ورودی‌ها انجام می‌شود.

```
public class Calculator {
  public static int add(int a, int b) {
    return a + b;
  }
  public static double add(double a, double b) {
    return a + b;
  }
  public static int add(int a, int b, int c) {
    return a + b + c;
  }
}
```

```
// استفاده از متدهای سربارگذاری‌شده:
int x = Calculator.add(2, 3);        // 5 :خروجی
double y = Calculator.add(2.5, 3.1); // 5.6 :خروجی
int z = Calculator.add(1, 2, 3);     // 6 :خروجی
```

مثال‌هایی از سربارگذاری متدها

```
public static void print(String msg) {
  System.out.println(msg);
}
public static void print(int num) {
  System.out.println(num);
}
public static void print(double val) {
  System.out.println(val);
}
```

```
    // فراخوانی متد print با ورودی‌های مختلف:
    print("Hello!");
    print(100);
    print(3.14);
```

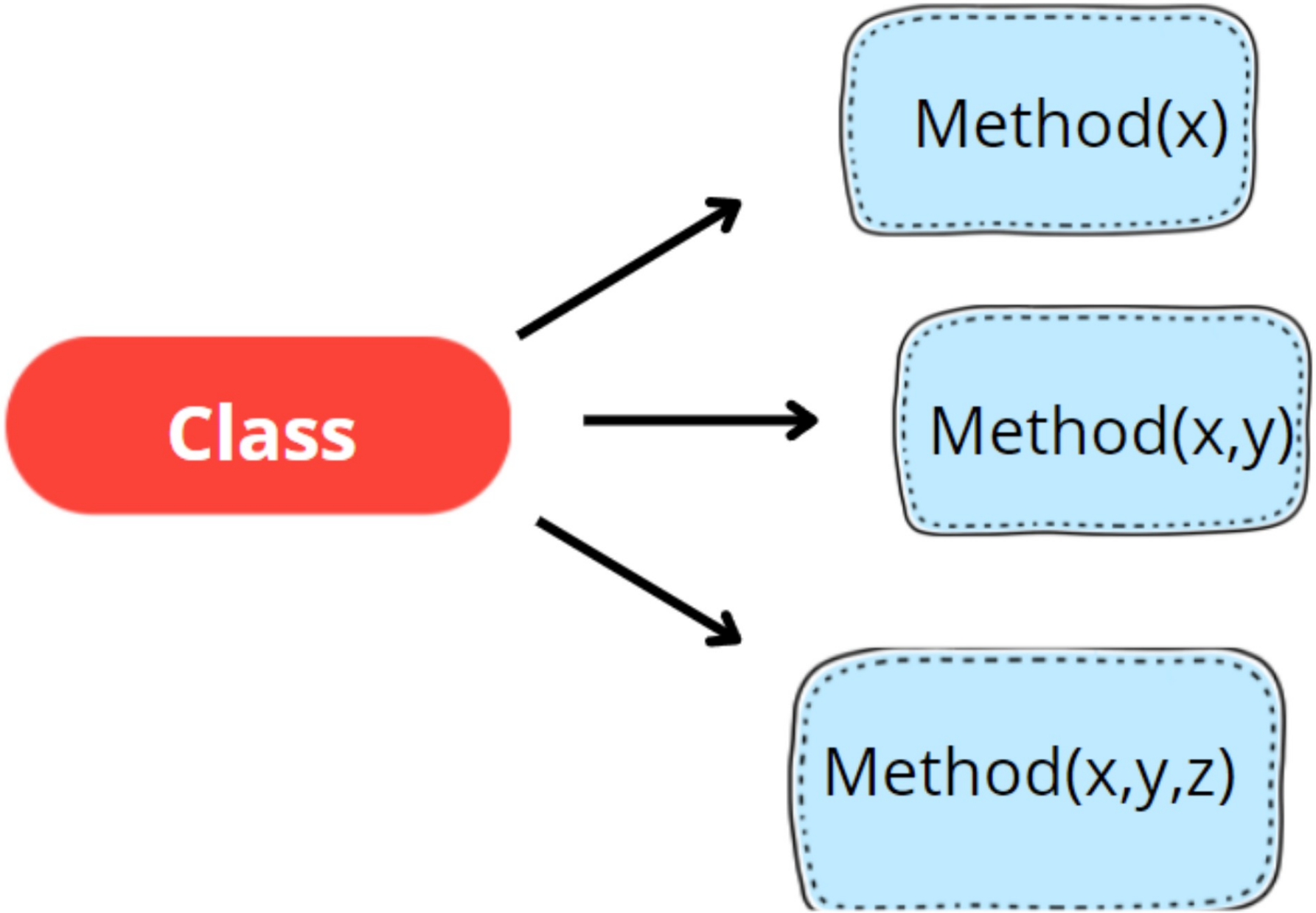
فراخوانی مبهم متدها (Ambiguous Calls)

گاهی کامپایلر به دلیل شباهت زیاد امضاها نمی‌تواند متد مناسب را انتخاب کند و خطای **فراخوانی مبهم** (Ambiguous Call) رخ می‌دهد.

```
public static void test(int a, double b) { ... }
public static void test(double a, int b) { ... }
```

```
// این فراخوانی مبهم است:
test(10, 20); // کامپایلر نمی‌داند کدام متد را انتخاب کند!
```

در مثال بالا، `test(10, 20)` می‌تواند هم به صورت `(int, double)` و هم `(double, int)` تفسیر شود و کامپایلر خطا می‌دهد.


Class
Method(x)
Method(x,y)
Method(x,y,z)

حوزه متغیرهای محلی (Local Variable Scope)

هر متغیر در برنامه، یک **حوزه اعتبار (Scope)** مشخص دارد. **متغیرهای محلی** فقط در همان بخشی از برنامه (معمولاً داخل متد یا بلوک) که تعریف شده‌اند معتبر هستند و بیرون از آن بخش قابل دسترسی نیستند.

تعریف و محدوده متغیرهای محلی

- متغیر محلی معمولاً در ابتدای یک متد یا بلوک (مثلاً یک حلقه یا دستور شرطی) تعریف می‌شود.
- این متغیر فقط تا پایان آن متد یا بلوک معتبر است.
- پس از خروج از بلوک، فضای حافظه متغیر آزاد می‌شود و دسترسی به آن دیگر ممکن نیست.

```java
public static void example() {
  int x = 10;
  if (x > 5) {
    int y = 20;
    System.out.println(x + y);  // OK
  }
  // System.out.println(y);
}
```

```
public static void main(String[] args) {
  int num = 42;
  for (int i = 0; i < 3; i++) {
    int temp = num + i;
    System.out.println(temp);
  }
  // System.out.println(temp);
}
```

متغیر `temp` فقط در محدوده حلقه for معتبر است و بیرون از آن قابل استفاده نیست. تعریف دوباره یک متغیر با همان نام در بلوک‌های مختلف مجاز است:

```
public static void test() {
  int a = 5;
  {
    int a = 10; // خطا: نمی‌توان در همان بلوک متغیر همنام داشت!
  }
}
```

انتزاع متدها (Method Abstraction)

انتزاع متدها به معنی پنهان کردن جزییات پیاده‌سازی و نمایش فقط بخش مهم و مورد نیاز برای کاربر است. شما هنگام استفاده از یک متد، فقط به نام و ورودی/خروجی آن توجه می‌کنید و لازم نیست از جزییات داخلی آن باخبر باشید. این اصل، برنامه را خواناتر، ایمن‌تر و قابل نگهداری‌تر می‌کند.

تعریف و کاربردهای انتزاع متدها

- کاربر فقط با **امضای متد** کار دارد (نام متد، پارامترها و نوع مقدار بازگشتی).
- جزییات اینکه متد چطور خروجی را محاسبه می‌کند، مخفی است.
- این رویکرد باعث می‌شود بتوانیم متدها را بدون نگرانی از تاثیر روی بخش‌های دیگر برنامه تغییر دهیم.
- در استفاده از کتابخانه‌ها و کدهای آماده، انتزاع بسیار مهم است (مثلاً متدهای کتابخانه `Math` در جاوا).

```
public static int square(int n) {
  return n * n;
}
```

```
    public static void main(String[] args) {
      int num = 7;
      int result = square(num); // فقط امضا را می‌بینیم
      System.out.println(result); // 49 :خروجی
    }
```

در مثال بالا، استفاده‌کننده فقط باید بداند متد `square` چه ورودی می‌گیرد و چه خروجی می‌دهد. نحوه محاسبه داخل متد برای او مهم نیست.

متدهای مثلثاتی و توان (Trigonometric and Power Methods)

در جاوا، بسیاری از محاسبات ریاضی به کمک متدهای آماده کلاس `Math` انجام می‌شود. متدهای مثلثاتی و توانی از جمله پرکاربردترین این متدها هستند که محاسبات سینوس، کسینوس، تانژانت، لگاریتم، توان و جذر را به سادگی انجام می‌دهند.

متدهای سینوس، کسینوس و تانژانت

- `Math.sin(x)` : مقدار سینوس زاویه `x` (بر حسب رادیان)
- `Math.cos(x)` : مقدار کسینوس زاویه `x` (بر حسب رادیان)
- `Math.tan(x)` : مقدار تانژانت زاویه `x` (بر حسب رادیان)
- تبدیل درجه به رادیان: `Math.toRadians(degree)`

```
double angle = 30; // درجه
double radian = Math.toRadians(angle);
double s = Math.sin(radian); // سینوس ۳۰ درجه
double c = Math.cos(radian); // کسینوس ۳۰ درجه
double t = Math.tan(radian); // تانژانت ۳۰ درجه
```

متدهای pow، log10، log، exp و sqrt

- `Math.exp(x)` : مقدار e به توان x
- `Math.log(x)` : لگاریتم طبیعی (بر مبنای e)
- `Math.log10(x)` : لگاریتم بر مبنای ۱۰
- `Math.pow(a, b)` : عدد a به توان b
- `Math.sqrt(x)` : جذر عدد x

```
double power = Math.pow(2, 3);      // 8.0
double root = Math.sqrt(16);        // 4.0
double exponential = Math.exp(2);   // e^2
double ln = Math.log(10);           // ln(10)
double log10 = Math.log10(100);     // 2.0
```

استفاده از این متدها نیاز به هیچ کتابخانه اضافه‌ای ندارد و فقط کافی است `Math` را صدا بزنید!

نکته: ورودی متدهای مثلثاتی `Math` باید بر حسب رادیان باشد.

متدهای روندسازی (Rounding Methods)

متدهای روندسازی برای گرد کردن اعداد اعشاری به کار می‌روند. در جاوا این متدها همگی در کلاس `Math` قرار دارند و شامل `floor`، `ceil`، `rint` و `round` هستند.

استفاده از متدهای rint، floor، ceil و round

- `Math.ceil(x)` : کوچک‌ترین عدد صحیح بزرگ‌تر یا مساوی x (گرد کردن به بالا)
- `Math.floor(x)` : بزرگ‌ترین عدد صحیح کوچک‌تر یا مساوی x (گرد کردن به پایین)
- `Math.rint(x)` : نزدیک‌ترین عدد صحیح به x به صورت عدد اعشاری (double)
- `Math.round(x)` : نزدیک‌ترین عدد صحیح به x (نتیجه long برای double، و int برای float)

```java
double a = 5.3;
double b = -5.7;
```

```java
System.out.println(Math.ceil(a));   // 6.0
System.out.println(Math.floor(a));  // 5.0
System.out.println(Math.rint(a));   // 5.0
System.out.println(Math.round(a));  // 5
```

```java
System.out.println(Math.ceil(b));   // -5.0
System.out.println(Math.floor(b));  // -6.0
System.out.println(Math.rint(b));   // -6.0
System.out.println(Math.round(b));  // -6
```

نکته: `ceil` همیشه به بالا گرد می‌کند (حتی برای اعداد منفی)، `floor` به پایین، `rint` به نزدیک‌ترین عدد صحیح به صورت double و `round` به نزدیک‌ترین عدد صحیح (int/long) گرد می‌کند.

متدهای ریاضی (Math Methods)

کلاس `Math` در جاوا مجموعه‌ای از متدهای آماده برای انجام محاسبات رایج ریاضی فراهم کرده است که استفاده از آن‌ها بسیار ساده و سریع است.
از جمله این متدها می‌توان به **abs**، **min**، **max** و **random** اشاره کرد.

- `Math.max(a, b)` : بزرگ‌ترین مقدار بین `a` و `b`
- `Math.min(a, b)` : کوچک‌ترین مقدار بین `a` و `b`
- `Math.abs(x)` : قدر مطلق عدد `x`
- `Math.random()` : یک عدد اعشاری تصادفی بین ۰ تا ۱

```
int m1 = Math.max(5, 10);       // 10
int m2 = Math.min(5, 10);       // 5
int abs1 = Math.abs(-7);        // 7
double rand = Math.random();    // 0.253489 :مثل
```

مثال‌های تکمیلی

```
System.out.println(Math.max(-5, 3));    // 3
System.out.println(Math.min(0, -2));    // -2
System.out.println(Math.abs(-3.8));     // 3.8
```

```
// تولید یک عدد صحیح تصادفی بین 1 تا 6 (مثل تاس):
int dice = 1 + (int)(Math.random() * 6);
```

مثال‌های بیشتر (سربارگذاری متدها)

مثال ساده:

```
public static int multiply(int a, int b) {
  return a * b;
}
public static double multiply(double a, double b) {
  return a * b;
}
// استفاده:
int x = multiply(2, 3);      // 6
double y = multiply(2.5, 4); // 10.0
```

در اینجا ضرب اعداد صحیح و اعشاری هر دو با یک نام متد انجام شده است.

مثال متوسط:

```
public static void showInfo(String name) {
  System.out.println("Name: " + name);
}
public static void showInfo(String name, int age) {
  System.out.println("Name: " + name + ", Age: " + age);
}
public static void showInfo(String name, int age, String city) {
  System.out.println("Name: " + name + ", Age: " + age + ", City: " + city);
}
// استفاده:
showInfo("Ali");
showInfo("Sara", 20);
showInfo("Mehdi", 23, "Tehran");
```

نمایش اطلاعات کاربر با تعداد ورودی‌های متفاوت به کمک سربارگذاری متد.

مثال سخت:

```
public static int sum(int[] arr) {
  int s = 0;
  for(int i : arr) s += i;
  return s;
}
public static double sum(double[] arr) {
  double s = 0;
  for(double d : arr) s += d;
  return s;
}
public static int sum(int a, int b, int c) {
  return a + b + c;
}
// استفاده:
int[] nums = {1, 2, 3, 4};
double[] scores = {2.5, 3.5, 4.5};
System.out.println(sum(nums));     // 10
System.out.println(sum(scores));   // 10.5
System.out.println(sum(1, 2, 3));  // 6
```

در این مثال جمع آرایه اعداد صحیح، آرایه اعشاری و سه عدد جدا با سربارگذاری متد انجام شده است.

مثال‌های بیشتر (حوزه متغیرهای محلی)

مثال ساده:

```
public static void printNumber() {
  int num = 5;
  System.out.println(num); // معتبر است num فقط در این متد
}
```

متغیر num فقط در محدوده متد printNumber اعتبار دارد و بیرون از آن ناشناخته است.

مثال متوسط:

```
public static void main(String[] args) {
  for(int i = 0; i < 3; i++) {
    int temp = i * 2;
    System.out.println(temp);
  }
  // System.out.println(temp); // خطا! temp فقط داخل حلقه معتبر است
}
```

در این مثال، temp در هر بار اجرای حلقه تعریف و فقط داخل حلقه اعتبار دارد.

مثال سخت:

```
public static void main(String[] args) {
  int a = 10;
  if(a > 5) {
    int b = a + 2;
    System.out.println(b);
  }
  // System.out.println(b); // خطا: b فقط در if معتبر است!
}
```

b فقط در محدوده if معتبر است؛ بیرون از بلوک if وجود ندارد.

مثال‌های بیشتر (متدهای مثلثاتی و توان)

مثال ساده:

```
// محاسبه سینوس ۹۰ درجه
double deg = 90;
double rad = Math.toRadians(deg);
double s = Math.sin(rad); // 1.0
```

ورودی باید بر حسب رادیان باشد. خروجی سینوس ۹۰ درجه، ۱ است.

مثال متوسط:

```
// محاسبه جذر و توان
double r = Math.sqrt(25);      // 5.0
double p = Math.pow(3, 4);     // 81.0
```

مثال استفاده همزمان از متد sqrt و pow برای جذر و توان عدد.

مثال سخت:

```
// محاسبه لگاریتم و نمایی
double exp1 = Math.exp(1);          // e^1 ≈ 2.718
double ln100 = Math.log(100);       // ln(100) ≈ 4.605
double log10_1000 = Math.log10(1000); // 3.0
```

نمونه‌ای از استفاده ترکیبی از توابع نمایی و لگاریتم در جاوا.

مثال‌های بیشتر (متدهای ریاضی)

مثال ساده:

```
int max = Math.max(11, 5); // 11
int min = Math.min(-2, 7); // -2
```

بدست آوردن بیشینه و کمینه دو عدد به کمک متد max و min.

مثال متوسط:

```
System.out.println(Math.abs(-12));   // 12
System.out.println(Math.abs(3.5));   // 3.5
```

محاسبه قدرمطلق یک عدد صحیح و اعشاری با abs.

مثال سخت:

```
// تولید عدد صحیح تصادفی بین 1 تا 100
int rnd = 1 + (int)(Math.random() * 100);
System.out.println(rnd);
```

با ضرب Math.random در ۱۰۰ و جمع کردن ۱، عدد تصادفی بین ۱ تا ۱۰۰ تولید می‌شود.

مثال چالشی (ماژوله‌بندی و استفاده مجدد از متدها)

یک متد بنویسید که یک آرایه از اعداد صحیح بگیرد و دو متد مجزا برای جمع اعداد مثبت و جمع اعداد منفی آرایه بنویسد. در نهایت، حاصل جمع اعداد مثبت و جمع اعداد منفی را در خروجی چاپ کنید.

راه‌حل (ماژوله‌بندی و استفاده مجدد از متدها)

```
public static int sumPositives(int[] arr) {
  int sum = 0;
  for(int x : arr)
    if(x > 0) sum += x;
  return sum;
}
public static int sumNegatives(int[] arr) {
  int sum = 0;
  for(int x : arr)
    if(x < 0) sum += x;
  return sum;
}
public static void main(String[] args) {
  int[] nums = {4, -2, 7, -5, 3};
  System.out.println("Sum of positives: " + sumPositives(nums));
  System.out.println("Sum of negatives: " + sumNegatives(nums));
}
```

مثال چالشی (حوزه متغیرهای محلی)

یک متد بنویسید که یک آرایه از اعداد صحیح دریافت کند و فقط اعداد اول (prime) آرایه را در خروجی چاپ کند. حتماً بررسی اول بودن عدد را با متغیر محلی انجام دهید که فقط در همان محدوده حلقه معتبر باشد.

راه‌حل (حوزه متغیرهای محلی)

```
public static boolean isPrime(int n) {
  if(n < 2) return false;
  for(int i = 2; i <= Math.sqrt(n); i++)
    if(n % i == 0) return false;
  return true;
}
public static void printPrimes(int[] arr) {
  for(int num : arr) {
    boolean primeHere = isPrime(num); // متغیر محلی
    if(primeHere)
      System.out.print(num + " ");
  }
}
public static void main(String[] args) {
  int[] arr = {2, 4, 5, 9, 13};
  printPrimes(arr); // 13 5 2 :خروجی
}
```

مثال چالشی (انتزاع متدها)

چند متد شرطی (مثلاً بررسی زوج بودن، بزرگ‌تر از ۵ بودن و غیره) بنویسید و یک متد بنویسید که با دریافت آرایه و یک پارامتر شرطی (مثلاً یک رشته) فقط اعدادی را چاپ کند که شرط مربوطه را داشته باشند.

راه‌حل (انتزاع متدها)

```
public static boolean isEven(int n) { return n % 2 == 0; }
public static boolean isGreaterThanFive(int n) { return n > 5; }


public static void printByCondition(int[] arr, String cond) {
  for(int n : arr) {
    if(cond.equals("even") && isEven(n))
      System.out.print(n + " ");
    else if(cond.equals("gt5") && isGreaterThanFive(n))
      System.out.print(n + " ");
  }
}
public static void main(String[] args) {
  int[] arr = {2, 7, 4, 9};
  printByCondition(arr, "even"); // 4 2 :خروجی
  System.out.println();
  printByCondition(arr, "gt5"); // 9 7 :خروجی
}
```

مثال چالشی (متدهای مثلثاتی و توان)

برنامه‌ای بنویسید که زاویه‌های ۰ تا ۱۸۰ را با گام ۳۰ درجه به رادیان تبدیل کند و مقدار سینوس هر زاویه را محاسبه و به صورت جدولی نمایش دهد.

راه‌حل (متدهای مثلثاتی و توان)

```
public static void main(String[] args) {
  System.out.println("Angle\tSin");
  for(int angle = 0; angle <= 180; angle += 30) {
    double rad = Math.toRadians(angle);
    double sinVal = Math.sin(rad);
    System.out.printf("%3d\t%.2f\n", angle, sinVal);
  }
}
```

مثال چالشی (متدهای روندسازی)

برنامه‌ای بنویسید که ۱۰ عدد اعشاری تصادفی بین ۰ تا ۱۰۰ تولید کند و هر عدد را به سه روش round، ceil و floor گرد کند و نتیجه هر روش را به صورت ستونی چاپ کند.

راه‌حل (متدهای روندسازی)

```
public static void main(String[] args) {
  System.out.println("Num\tRound\tCeil\tFloor");
  for(int i = 0; i < 10; i++) {
    double num = Math.random() * 100;
    System.out.printf("%.2f\t%d\t%.0f\t%.0f\n",
      num, Math.round(num), Math.ceil(num), Math.floor(num));
  }
}
```

مثال چالشی (متدهای ریاضی)

برنامه‌ای بنویسید که n بار عدد تصادفی بین -۵۰ تا ۵۰ تولید کند و بیشینه، کمینه و میانگین قدر مطلق این اعداد را محاسبه و چاپ کند.

راه‌حل (متدهای ریاضی)

```
public static void main(String[] args) {
  int n = 10, max = -51, min = 51, sumAbs = 0;
  for(int i = 0; i < n; i++) {
    int rnd = -50 + (int)(Math.random() * 101);
    max = Math.max(max, rnd);
    min = Math.min(min, rnd);
    sumAbs += Math.abs(rnd);
  }
  System.out.println("Max: " + max + ", Min: " + min +
                     ", Avg Abs: " + (sumAbs / (double)n));
}
```

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.

- **پارامتر (parameter)** و **آرگومان (argument)** چه تفاوتی با هم دارند؟
- **سربارگذاری متد (Method Overloading)** را با یک مثال توضیح دهید.
- کلمه کلیدی **void** در تعریف یک متد به چه معناست؟
- متدهای **ماژوله** چه مزایایی در توسعه نرم‌افزار دارند؟

پایان

در صورت هرگونه سوال یا پیشنهاد میتونید با من در ارتباط باشید :)

gmail: ariashakoo1@gmail.com
telegram: @Ariashakoo