

به نام خدا



## برنامه‌سازی ییشوفته

دانشگاه شهید بهشتی · دانشکده مهندسی کامپیووتر

دکتر مجتبی وحیدی اصل

آشنایی با جاوا

قائم علی‌آبادی

## فهرست مطالب

1. چرا جاوا؟
2. شاخص‌های محبوبیت زبان‌های برنامه‌نویسی
3. خصوصیات زبان جاوا
4. ابزارهای لازم
5. سرفصل‌های درس
6. نکات مهم موفقیت در درس

## مقدمه

این جلسه شروع مسیر ما با زبان [جاوا](#) است. قبل از اینکه وارد مباحث پیشرفته‌تر بشیم، خوبه اول کمی درباره تاریخچه جاوا بدونیم و با مفاهیم اولیه اون آشنا بشیم. هدف اینه که یک پایه مطمئن داشته باشیم تا در ادامه راحت‌تر سراغ موضوعات جدی‌تر مثل شیءگرایی یا چندریسمانی بريم. بنابراین در این جلسه بیشتر روی مقدمات تمکن داریم: از نوشتن و اجرای اولین برنامه گرفته تا شناخت متغیرها و عملگرها. این مباحث ساده به نظر میان، اما در واقع زیربنای کل یادگیری شما در ادامه خواهد بود.

- مروری کوتاه بر [تاریخچه جاوا](#)
- آشنایی با نحوه [کامپایل](#) و [اجرای](#) برنامه‌های جاوا
- نوشتن اولین برنامه ساده در جاوا
- شناخت [متغیرها](#) و انواع داده‌های پایه

- کار با عملگرهای حسابی و رابطه‌ای
- بررسی ساختارهای اولیه زبان



**نکته:** این جلسه بیشتر برای آشنایی و شروع کار با جاواست؛ پس اگر چیزی برآتون جدید بود نگران نباشید، در ادامه بارها به این مفاهیم برگردیم.

## مروری بر جاوا

- زبان **جاوا** اولین بار توسط **James Gosling** در شرکت **Sun Microsystems** معرفی شد.
- جاوا یک زبان **شیءگرا** است؛ یعنی همه چیز در قالب شیء و کلاس تعریف و مدیریت می‌شود.
- سینتکس و ساختار آن بر پایه زبان‌های **C/C++** طراحی شده است تا برای برنامه‌نویسان آشنا باشد.
- در سال ۲۰۰۹ شرکت **Oracle** خریداری شد و از آن زمان تاکنون، اوراکل مالک اصلی و توسعه‌دهنده جاوا است.



**نکته:** با وجود گذشت سال‌ها از تولد جاوا، این زبان همچنان توسط اوراکل پشتیبانی می‌شود و هر نسخه جدید آن قابلیت‌های مدرن‌تری برای توسعه نرم‌افزار ارائه می‌دهد.

## جاوا چه زمانی متولد شد؟

جاوا در سال ۱۹۹۵ رسماً معرفی شد. در همان سال زبان‌های دیگری مانند [PHP](#) و [Delphi](#), [JavaScript](#) نیز متولد شدند. در این بازه زمانی، اینترنت در حال رشد سریع بود و جاوا با **ویژگی قابلیت حمل (Platform Independence)** توانست خیلی زود جایگاه ویژه‌ای در توسعه نرم‌افزار پیدا کند.

## Java Evolution

Mind mapping by Tirthal Patel  
(last updated on 12-Apr-2014)



**نکته:** اگرچه زبان‌هایی مانند C و C++ قدیمی‌تر از جاوا هستند، اما جاوا با رویکرد شیء‌گرایی و قابلیت اجرا روی همه پلتفرم‌ها توانست در رقابت زنده بماند و محبوب شود.

## مراحل تولید و اجرای یک برنامه جاوا

برای ایجاد و اجرای یک برنامه جاوا، لازم است مجموعه‌ای از مراحل پشت‌سر گذاشته شوند. این مراحل به صورت استاندارد در **Java Development Kit (JDK)** تعریف شده‌اند و معمولاً در پنج فاز اصلی انجام می‌شوند.

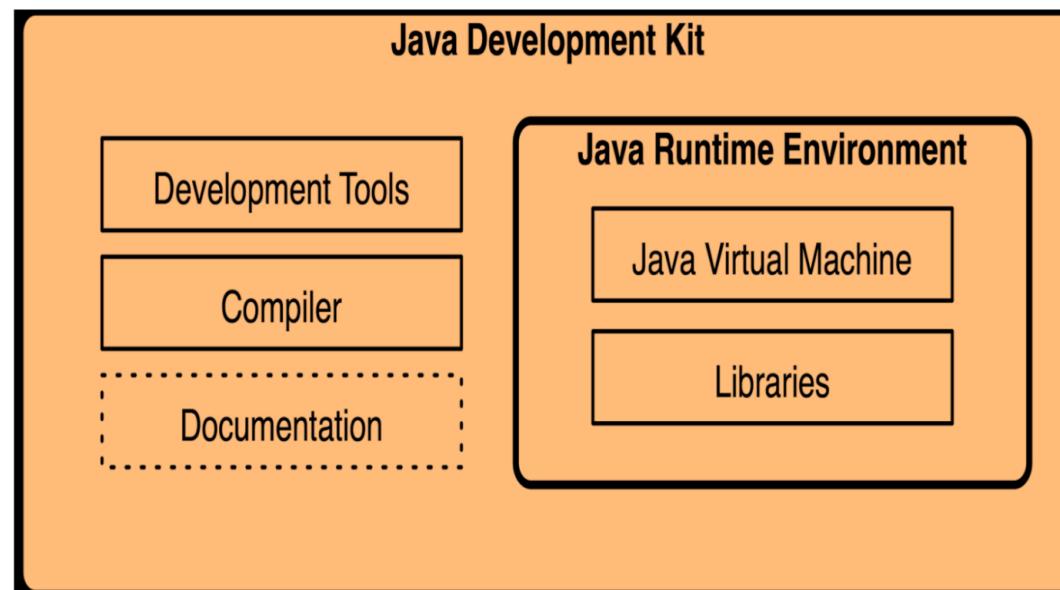
- نوشتن و ویرایش کد منبع (Source Code)
- کامپایل برنامه
- بارگذاری برنامه در حافظه (Verification)
- اجرا توسط ماشین‌های مجازی جاوا (JVM)

**نکته:** این فرایند به برنامه‌نویس کمک می‌کند که نوشته شده را از سطح متن ساده به یک برنامه اجرایی مستقل تبدیل کند.

## Java Development Kit (JDK)

برای کامپایل و اجرای برنامه‌های جاوا به بسته نرم‌افزاری **JDK** نیاز داریم. این بسته شامل ابزارها و محیط‌های موردنیاز برای توسعه و اجرای برنامه است.

- ابزارهای توسعه و دیباگ : **Development Tools**
- مبدل کد منبع جاوا به بایت‌کد : **Compiler**
- مستندات و راهنمای زبان : **Documentation**
- شامل JVM و کتابخانه‌ها : **Java Runtime Environment (JRE)**



**یادآوری:** JDK مجموعه کامل است؛ در حالی که **JRE** فقط برای اجرای برنامه‌ها کافی است.

## فاز اول: نوشتن کد برنامه

در این مرحله کد برنامه با استفاده از یک ویرایشگر متن نوشته می‌شود. فایل ایجادشده، همان **کد منبع (Source Code)** است که باید با پسوند **.java** ذخیره گردد.

- نوشتن کد با ویرایشگرهای ساده (مثلًا WordPad یا Notepad) یا
- انجام اصلاحات لازم و ذخیره فایل در حافظه
- حفظ پسوند **.java** برای فایل

**نکته:** فایل منبع تنها متن ساده است و بدون کامپایل قابلیت اجرا ندارد.

```
In [ ]: public class Welcome3 {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java Programming!");  
    }  
}
```

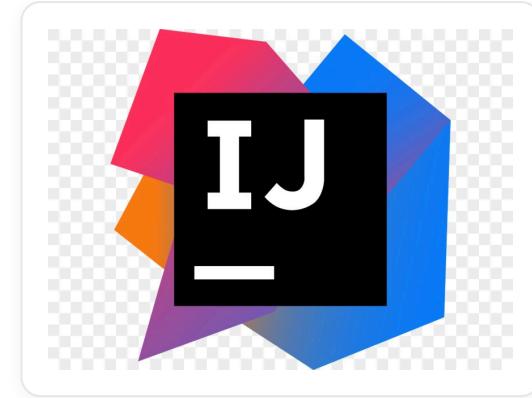
## محیط‌های توسعه یکپارچه (IDE)

برای سهولت در نوشتن، ویرایش و اشکال‌زدایی، برنامه‌نویسان از محیط‌های توسعه یکپارچه یا **IDE** استفاده می‌کنند. این ابزارها امکاناتی فراتر از ویرایشگر متن ساده ارائه می‌دهند.

- ویرایشگر پیشرفته کد
- اشکال‌زدایی (Debugger) برای یافتن خطاهای منطقی
- مدیریت پروژه و فایل‌ها



**NetBeans**



مثال: چهار رایج برای جاوا `VS Code`, `IntelliJ IDEA`, `Eclipse`, `NetBeans` هستند.

## فاز دوم: کامپایل برنامه

پس از نوشتن کد منبع، آن را با استفاده از دستور `javac` کامپایل می‌کنیم. خروجی این مرحله یک فایل بایت‌کد با پسوند `.class` است.

- اجرای دستور زیر در محیط Command Prompt:

```
javac Welcome.java
```

- فایل جدیدی به نام `Welcome.class` تولید می‌شود
- این فایل حاوی نسخه میانی برنامه برای اجرای در JVM است

نکته: بایت‌کد مستقل از سیستم‌عامل بوده و روی هر دستگاهی با JVM قابل اجراست.

## ماشین مجازی جاوا (JVM)

JVM بایت کد تولید شده را اجرا می‌کند. این ماشین مجازی لایه‌ای میان برنامه جاوا و سیستم‌عامل ایجاد می‌کند تا برنامه روی هر سکوی نرم‌افزاری یا سخت‌افزاری قابل اجرا باشد.

- بخشی از **Java Platform** و **JDK**
- شبیه‌ساز سخت‌افزار و سیستم‌عامل برای اجرایی بایت کد
- امکان اجرای یک برنامه روی سیستم‌های مختلف بدون تغییر
- برخلاف زبان‌های وابسته به سخت‌افزار، برنامه‌های جاوا قابل حمل هستند

**مقایسه:** JVM را می‌توان مشابه ماشین مجازی دات‌نت (.NET) دانست.

## فاز سوم: بارگذاری برنامه در حافظه

در این مرحله، فایل‌های **class** حاوی بایت کد توسط JVM بارگذاری شده و به حافظه اصلی (RAM) منتقل می‌شوند.

- این فرآیند با نام **Loading** شناخته می‌شود

- لودر کلاس، تمام فایل‌های موردنیاز را از دیسک یا کتابخانه جاوا به حافظه منتقل می‌کند

**یادآوری:** بارگذاری موفق در این مرحله پیش‌نیاز ادامه اجرای برنامه است.

## فاز چهارم: درستی‌سنجد

پس از بارگذاری، JVM بایت‌کدها را بررسی می‌کند تا از رعایت قوانین امنیتی و سازگاری آنها مطمئن شود. این مرحله با نام **Bytecode Verification** شناخته می‌شود.

- کنترل امنیتی بایت‌کدها
- اطمینان از عدم وجود دستورهای غیرمجاز (Runtime Errors)
- جلوگیری از بروز خطاها زمان اجرا

**نتیجه:** تنها بایت‌کدهای معتبر اجازه اجرا پیدا می‌کنند.

## فاز پنجم: اجرای برنامه

در این مرحله، JVM بایت کد معتبر را اجرا می‌کند. اجرای کد می‌تواند به دو صورت انجام گیرد: **تفسیر خطبهخط (Interpreter)** یا **کامپایل لحظه‌ای (JIT Compiler)**.

- تفسیر ساده: اجرای بایت کد به صورت خطبهخط
- JIT Compiler: تبدیل بخش‌های پرکاربرد بایت کد به زبان ماشین برای افزایش سرعت
- امکان اجرای یک برنامه در سیستم‌های مختلف بدون تغییر کد

**یادآوری:** ترکیب تفسیر و JIT باعث اجرای بهینه برنامه‌های جاوا می‌شود.

## نوشتن یک برنامه ساده در جاوا

برای شروع کار با زبان **جاوا**، یک برنامه ساده می‌نویسیم که تنها وظیفه‌اش چاپ یک رشته در خروجی باشد. این برنامه شامل یک کلاس و متدهای `main` است که نقطه آغاز اجرای هر برنامه جاوا محسوب می‌شود.

- برای تعریف کلاس از کلیدواژه `public class` استفاده کرده و سپس نام کلاس را مشخص می‌کنیم.
- تمامی دستورات کلاس باید در میان آکولاد `{ }`  نوشته شوند.
- کلاس باید شامل متدهای `main` باشد؛ این متدهای نقطه شروع اجرای برنامه است.
- ساختار متدهای اصلی به صورت `public static void main(String[] args)` تعریف می‌شود.
- درون متدهای `main` می‌توانیم با دستور `System.out.println()` متن دلخواه را چاپ کنیم.
- پس از هر دستور باید از `;` برای پایان دستور استفاده شود.

**نکته:** هر فایل جاوا باید تنها شامل یک کلاس عمومی (`public`) باشد و نام فایل دقیقاً با نام همان کلاس یکسان ذخیره گردد.

```
In [ ]: public class First {  
    public static void main(String[] args) {  
        System.out.println("Salam!!!");  
    }  
}
```

## شناسه‌ها در جاوا (Identifiers)

شناسه نامی است که برای معرفی متغیرها، کلاس‌ها، متدها و سایر اجزای برنامه استفاده می‌شود.

- می‌تواند شامل حروف، ارقام، خط زیرین (`_`) یا نماد (`$`) باشد.

- باید با یک حرف، خط زیرین یا `$` شروع شود (شروع با عدد مجاز نیست).

- نمی‌تواند یکی از **کلمات رزو شده** جاوا باشد.

- نمی‌تواند مقادیر `null` یا `true` یا `false` باشد.

- هیچ محدودیتی روی طول شناسه وجود ندارد.

**نکته:** انتخاب نام‌های خوانا و معنادار برای شناسه‌ها باعث درک بهتر کد می‌شود.

## کلمات رزرو شده در جاوا

نمونه‌ای از کلمات کلیدی جاوا:

short	long	int	float	double	char	byte	boolean
if	while	static	final	abstract	protected	private	public
new	class	continue	break	case	switch	for	else
import	throw	catch	try	super	this	void	return
			enum	implements	extends	interface	package

**یادآوری:** همه این کلمات باید با حروف کوچک نوشته شوند.

## متغیرها در جاوا (Variables)

متغیر محلی در حافظه است که داده‌ای را در خود نگه می‌دارد. می‌توان آن را مانند یک لیوان در نظر گرفت که مقداری داخل آن ریخته می‌شود.

- هر متغیر باید **نوع داده‌ای مشخص** داشته باشد (مانند `int`, `double`, `boolean`).
- اندازه متغیرها از پیش مشخص است و وابسته به سخت‌افزار یا سیستم عامل نیست.

- پیش از استفاده، باید مقداردهی شود.
- برای مقداردهی از عملگر `=` استفاده می‌کنیم.

مثال:

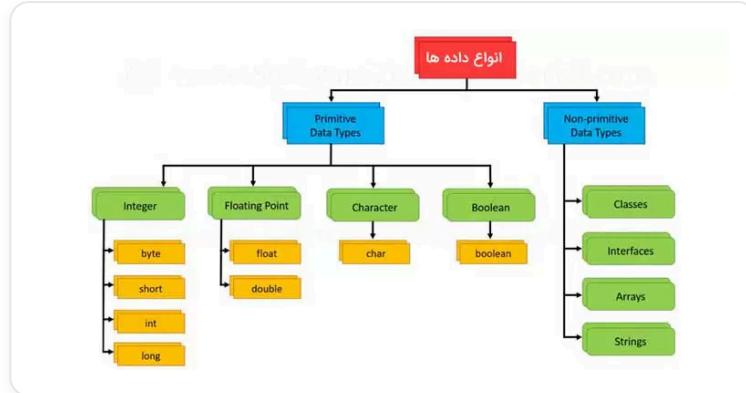
```
int x;  
x = 2;
```

**نکته:** نام متغیر باید گویا باشد تا هدف استفاده از آن در کد مشخص شود.

## انواع داده در جاوا

در جاوا، داده‌ها به دو دسته کلی تقسیم می‌شوند:

- **انواع اولیه (Primitive Types):** مانند `int`, `double`, `boolean` وغیره.
- **انواع ارجاعی (Reference Types):** شامل اشیاء (Objects)، آرایه‌ها و کلاس‌ها.



## انواع اصلی (Primitive) در جاوا

جاوا ۸ نوع داده اولیه دارد که هر کدام اندازه و مقدار پیشفرض خاصی دارند:

Range	Size	Default	محتوا	Type
NA	bit 1	false	true/false	boolean
u0000 to \uFFFF\	bits 16	u0000\	Unicode character	char
to 127 128-	bits 8	0	Signed integer	byte
to 32767 32768-	bits 16	0	Signed integer	short
to $2^{31}-1$ $31^2-$	bits 32	0	Signed integer	int
to $2^{63}-1$ $63^2-$	bits 64	0	Signed integer	long
$\pm 3.4E38\sim$	bits 32	0.0	IEEE 754 floating point	float

Range	Size	Default	محتوا	Type
$\pm 1.7E308\sim$	bits 64	0.0	IEEE 754 floating point	double

**نکته:** انواع اولیه به صورت مستقیم در حافظه ذخیره می‌شوند و سریع‌تر از انواع ارجاعی هستند.

## عملگرهای حسابی در جاوا

عملگرهای حسابی برای انجام محاسبات ریاضی روی مقادیر استفاده می‌شوند:

حاصل	مثال	مفهوم	نماد
35	$1 + 34$	جمع	+
33.9	$0.1 - 34.0$	تفريق	-
9000	$30 * 300$	ضرب	*
0.5	$2.0 / 1.0$	تقسیم	/
2	$3 \% 20$	باقي‌مانده	%

**یادآوری:** نتیجه عملگر تقسیم بسته به نوع داده می‌تواند صحیح (integer) یا اعشاری (double) باشد.

## کاربرد عملگر تقسیم و باقیمانده

هنگام استفاده از عملگر `/` یا `%` نتیجه به نوع داده ورودی وابسته است:

`2 / 2` → مقدار صحیح •

`2.5 / 2` → مقدار اعشاری •

`1 % 2` → باقیمانده تقسیم •

## عملگر باقیمانده (Modulo)

عملگر `%` برای محاسبه باقیمانده تقسیم به کار می‌رود و در بسیاری از مسائل کاربرد دارد:

- تشخیص زوج یا فرد بودن عدد
- یافتن روز هفته با توجه به روز فعلی و تعداد روزهای بعد

☞ مثال: اگر امروز پنجشنبه (روز ششم) باشد و ۱۰ روز بعد را بخواهیم:  
 $(6 + 10) \% 7 = 2$ : روز دوم یعنی یکشنبه

**نکته:** عملگر باقیمانده در مسائل زمان‌بندی و چرخه‌ای بسیار پرکاربرد است.

## تقدم عملگرها

در جاوا، برخی عملگرها نسبت به بقیه تقدم بیشتری دارند:

عملگر	توضیح	اولویت
% , *, /	ضرب، تقسیم، باقیمانده	اول انجام می‌شوند
- , +	جمع و تفریق	بعد از آنها انجام می‌شوند

مثال:  $1 + 2 * 3$  معادل است با  $1 + (2 * 3) = 7$  ✎

## عبارات حسابی در جاوا

عبارات حسابی ترکیبی از متغیرها، ثابت‌ها و عملگرهای ریاضی هستند. جاوا آن‌ها را مطابق تقدم عملگرها محاسبه می‌کند. ✎ مثال:

$$(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x + 9 * (4 / x + 9 + x) / y$$

**یادآوری:** استفاده از پرانتز به وضوح و درک بهتر اولویت محاسبات کمک می‌کند.

## عملگرهای تساوی و رابطه‌ای

معنی شرط	نمونه در جاوا	عملگر در جاوا	عملگر ریاضی استاندارد
x برابر با y است	$x == y$	$==$	$=$
x نابرابر با y است	$x != y$	$!=$	$\neq$
x بزرگتر از y است	$x > y$	$<$	$<$
x کوچکتر از y است	$x < y$	$>$	$>$
x بزرگتر یا مساوی y است	$x >= y$	$=<$	$\leq$
x کوچکتر یا مساوی y است	$x <= y$	$=>$	$\geq$

## شرکت‌پذیری

وقتی دو عملگر دارای اولویت یکسان باشند، ترتیب اجرای آن‌ها بر اساس **شرکت‌پذیری** تعیین می‌شود.

اجرا می‌شود (راست به چپ).  $x = (y = (z = 17))$  به صورت  $x = y = z = 17$  •

اجرا می‌شود (چپ به راست).  $(72 / 2) / 3$  به صورت  $72 / 2 / 3$  •

## نماذهای میانبری

معادل با	مثال	عملگر
$i = i + 8$	$i += 8$	$=+$
$f = f - 8.0$	$f -= 8.0$	$=-$
$i = i * 8$	$i *= 8$	$=*$
$i = i / 8$	$i /= 8$	$=/$
$i = i \% 8$	$i \%= 8$	$=\%$

## عملگرهای ++ و --

- **`++var`**: یک واحد به مقدار اضافه می‌کند، سپس مقدار جدید برگردانده می‌شود.
- **`var++`**: مقدار اصلی برگردانده می‌شود، سپس یک واحد اضافه می‌کند.
- **`--var`**: یک واحد کم می‌کند، سپس مقدار جدید برگردانده می‌شود.
- **`var--`**: مقدار اصلی برگردانده می‌شود، سپس یک واحد کم می‌کند.

## تفاوت عملگرهای پیشوندی و پسوندی

در استفاده از `i++` و `++i`، خروجی متفاوت است:

```
In [ ]: int i = 10;
int newNum = 10 * i++;
System.out.println(newNum);
```

```
In [ ]: int i = 10;
int newNum = 10 * (++i);
System.out.println(newNum);
```

## عملگر + روی رشته‌ها

در جاوا، عملگر `+` برای [رشته‌ها \(String\)](#) سربارگذاری شده و امکان [به هم‌چسباندن رشته‌ها](#) را فراهم می‌کند. اگر یکی از عملوندها رشته باشد، سایر عملوندها هم به رشته تبدیل می‌شوند.

```
In [ ]: int x = 0, y = 1, z = 2;
String myString = "x, y, z ";
System.out.println(myString + x + y + z);
```

## برنامه محاسبه مساحت دایره

ابتدا متغیر `radius` تعریف می‌شود و حافظه برای آن اختصاص می‌یابد.

```
In [ ]: public class ComputeArea {
    public static void main(String[] args) {
        double radius; // radius is created in memory
        double area; // area is created in memory
        radius = 20; // radius is now 20
        area = radius * radius * 3.14159; // area is now 1256.636
        System.out.println("The area for the circle of radius " + radius + " is " + area);
    }
}
```

## خواندن ورودی از کنسول (از کاربر)

- ابتدا باید کتابخانه `java.util.Scanner` را در ابتدای برنامه `import` کنیم.
- سپس یک شیء از نوع `Scanner` ایجاد می‌کنیم:  
`;Scanner input = new Scanner(System.in)`
- از متد‌های زیر برای خواندن مقادیری از انواع مختلف استفاده می‌کنیم:

متدها	کاربرد
next()	خواندن یک رشته
nextByte()	خواندن یک مقدار از نوع byte
nextInt()	خواندن یک مقدار از نوع int
nextShort()	خواندن یک مقدار از نوع short
nextLong()	خواندن یک مقدار از نوع long
nextFloat()	خواندن یک مقدار از نوع float
nextDouble()	خواندن یک مقدار از نوع double
nextBoolean()	خواندن یک مقدار از نوع boolean

```
In [ ]: System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

## برنامه نویسی با ورودی کاربر

در این بخش، قصد داریم با استفاده از کلاس **Scanner** از کاربر ورودی بگیریم و آن را پردازش کنیم. برای این کار، دو برنامه ساده اما مهم می‌نویسیم:

1. **محاسبه مساحت دایره:** شعاع دایره توسط کاربر وارد می‌شود و برنامه مساحت دایره را محاسبه کرده و نمایش می‌دهد.
2. **تبدیل ثانیه به دقیقه و ثانیه:** تعداد ثانیه‌ای که کاربر وارد می‌کند به دقیقه و ثانیه باقیمانده تبدیل و نمایش داده می‌شود.

**نکته:** این تمرین‌ها کمک می‌کنند تا با نحوه دریافت داده از کاربر و انجام محاسبات پایه در جاوا آشنا شوید.

```
In [ ]: import java.util.Scanner; // Scanner is in the java.util package
```

```
public class ComputeAreaWithConsoleInput {  
    public static void main(String[] args) {  
        // Create a Scanner object  
        Scanner input = new Scanner(System.in);  
  
        // Prompt the user to enter a radius  
        System.out.print("Enter a number for radius: ");  
        double radius = input.nextDouble();  
  
        // Compute area  
        double area = radius * radius * 3.14159;  
  
        // Display result  
        System.out.println("The area for the circle of radius " + radius + " is " + area);  
    }  
}
```

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

```
public class DisplayTime {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        // Prompt the user for input  
        System.out.print("Enter an integer for seconds: ");  
        int seconds = input.nextInt();  
  
        int minutes = seconds / 60; // Find minutes in seconds  
        int remainingSeconds = seconds % 60; // Seconds remaining  
        System.out.println(seconds + " seconds is " + minutes + " minutes and " + remainingSeconds + " seconds");  
    }  
}
```

## ثابت‌ها و دقت ممیز شناور در جاوا

- برای تعریف ثابت‌ها از کلیدواژه `final` استفاده می‌کنیم:

```
final double PI = 3.14159;  
final int SIZE = 3;
```

- محاسبات اعشاری تقریبی هستند و همیشه دقیق ذخیره نمی‌شوند.

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);  
// 0.5000000000000001
```

به صورت دقیق ذخیره می‌شوند (int, long) اعداد صحیح خروجی:

## لیترال‌های عددی در جاوا

یک لیترال مقداری ثابت است که به‌طور مستقیم در برنامه نوشته می‌شود.

### لیترال‌های صحیح:

```
int i = 41;  
long x = 230990;  
double d = 5.0;
```

## لیترال‌های اعشاری:

- هستند `double` به طور پیش‌فرض از نوع.
- استفاده کنیم `f` یا `F` باشد از پسوند برای:

```
float f = 100.2f;
```

- استفاده می‌کنیم `D` یا `d` به‌طور صریح از `double` برای مشخص کردن:

```
double d2 = 100.2d;
```

## نماد علمی (Scientific notation):

```
double n1 = 1.23456e+2; // 123.456  
double n2 = 1.23456e-2; // 0.0123456
```

## تبدیل نوع داده‌ای (Type Conversion)

اگر دو عملوند از انواع مختلف باشند، جاوا به طور خودکار نوع مناسب را انتخاب می‌کند:

1. اگر یکی `double` باشد → دیگری هم به `double` تبدیل می‌شود.

2. اگر یکی `float` باشد → دیگری هم به `float` تبدیل می‌شود.

3. اگر یکی `long` باشد → دیگری هم به `long` تبدیل می‌شود.

4. در غیر این صورت هر دو به `int` تبدیل می‌شوند.

## مثال:

```
byte i = 100;  
long k = i * 3 + 4;           // ارتقا به long  
double d = i * 3.1 + k / 2;  // ارتقا به double
```

## سربیز داده‌ها (Overflow)



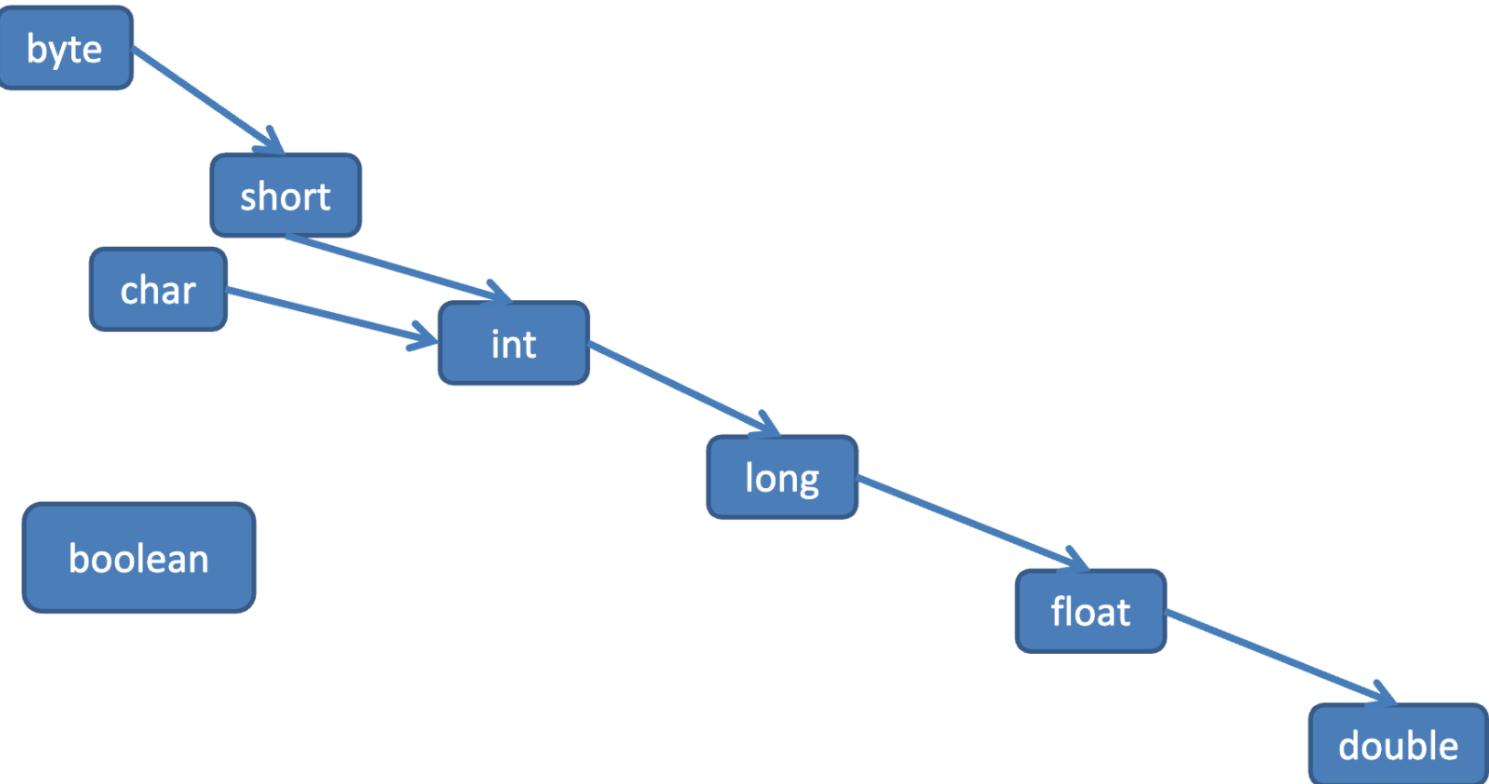
نمی‌توان مقداری بزرگ‌تر از ظرفیت یک نوع داده در آن قرار داد. در این حالت سربیز رخ می‌دهد و بخشی از مقدار از دست می‌رود.

**:مثال:**

```
int x = 1000;  
byte y = (byte)x; // بخشی از داده از دست می‌رود
```

## تبديل نوع ضمني

- تبدیل‌هایی که در جهت پیکان‌ها انجام شوند، توسط جاوا و خودکار انجام می‌شوند.
- پیکان‌ها خاصیت تعدد دارند.
- هر نوع تبدیل دیگر برخلاف جهت پیکان‌ها به `cast` مستقیم نیاز دارد.
- نوع **boolean** قابل تبدیل نمی‌باشد.
- نوع **char** یک نوع خاص به حساب می‌آید.



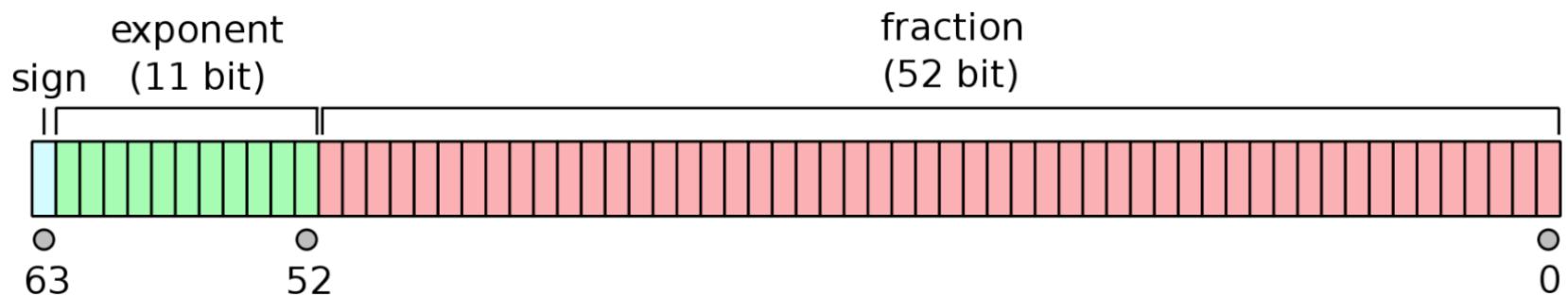
تبديل نوع عددی (numeric type conversion)

Implicit casting (Type widening)

```
double d = 3;
```

### Explicit casting (Type narrowing)

```
int i = (int)3.0;  
int i = (int)3.9; // Fraction part is truncated
```



range increases

byte, short, int, long, float, double

## جدول تبدیل نوع

double	float	long	int	char	short	byte	boolean	Convert From \ To
N	N	N	N	N	N	N	-	boolean
Y	Y	Y	Y	C	Y	-	N	byte
Y	Y	Y	Y	C	-	C	N	short
Y	Y	Y	Y	-	C	C	N	char
Y	*Y	Y	-	C	C	C	N	int
*Y	*Y	-	C	C	C	C	N	long
Y	-	C	C	C	C	C	N	float
-	C	C	C	C	C	C	N	double

- **N:** تبدیل نمی‌تواند انجام شود.
- **Y:** تبدیل خودکار انجام می‌شود.
- **C:** مستقیم دارد `cast` نیاز به.
- **Y\*:** تبدیل ممکن است باعث از دست رفتن دقیقت شود.

## مثال‌هایی از تبدیل

```
int i = 123456789; // a big integer
float f = i;        // f stores an approximation of i
System.out.println(f); // output: 1.23456792E8
```

```
i = (int) f; System.out.println(i); // output: 123456792
```

- انواع اعشاری تقریب عددها هستند.
- همیشه نمی‌توانند به اندازه انواع صحیح، تعداد زیادی رقم معنی‌دار ذخیره کنند.

## نوع داده کاراکتر

```
char letter = 'A';           // ASCII
char numChar = '4';          // ASCII
char letter = '\u0041';       // Unicode
char numChar = '\u0034';      // Unicode
```

اعمال شوند تا کاراکتر بعدی یا قبلی `char ch = 'a'; System.out.println(++ch); //` نکته: عملگرها افزایشی و کاهشی می‌توانند روی متغیرهای `b` و `to` بلاک زیر کد بزنید و تست کنید. را تولید کنند!

In [ ]:

## Unicode فرمت

- کاراکترهای جawa از Unicode استفاده می‌کنند.
- Unicode یک فرمت ۱۶ بیتی است که برای نمایش متون به زبان‌های مختلف استفاده می‌شود.
- کدهای Unicode از دو بایت (۱۶ بیت) تشکیل شده‌اند که پیش از کد، نماد `u\` قرار می‌گیرد.
- این ۱۶ بیت می‌توانند حدود 65535 کاراکتر را نمایش دهند.
- مجموعه کاراکترهای ASCII زیرمجموعه‌ای از Unicode است.

## فرمت Unicode - مثال

برنامه‌ای بنویسید که دو کاراکتر چینی و سه کاراکتر یونانی نمایش دهد.



## تبدیل (cast) بین نوع کاراکتر و انواع عددی

```
int i = 'a'; // Same as int i = (int)'a';
char c = 97; // Same as char c = (char)97;
```

## کarakterهای ASCII

مجموعه کarakterهای ASCII زیرمجموعه‌ای از Unicode بین \u0000 تا \u007f هستند.

**TABLE B.1** ASCII Character Set in the Decimal Index

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eof	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	\$	%	&	,
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	-	del		