به نام خدا

برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی · دانشکده مهندسی و علوم کامپیوتر

دکتر مجتبی وحیدی اصل

آشنایی با جاوا

قائم علی‌آبادی

فهرست مطالب

1. مقدمه و مروری بر جاوا و تاریخچه
2. فرایند تولید و اجرای برنامه و ابزارهای لازم
3. شناسه‌ها و کلمات رزرو شده
4. متغیرها و انواع داده (Primitive/Reference)
5. انواع اولیه: اندازه، مقدار پیش‌فرض و بازه
6. عملگرهای حسابی، تقدم و شرکت‌پذیری
7. عملگرهای رابطه‌ای و تساوی
8. نمادهای میان‌بری و عملگرهای ++/--
9. رشته‌ها و عملگر +
10. ثابت‌ها و دقت ممیز شناور
11. لیترال‌های عددی و نماد علمی
12. تبدیل نوع: ضمنی، صریح، سرریز و جدول تبدیل
13. نوع کاراکتر، Unicode/ASCII و تبدیل با اعداد

مقدمه

این جلسه شروع مسیر ما با زبان **جاوا**ست. قبل از اینکه وارد مباحث پیشرفته‌تر بشیم، خوبه اول کمی درباره تاریخچه جاوا بدونیم و با مفاهیم اولیه اون آشنا بشیم. هدف اینه که اول توی مباحث پایه‌ای قوی بشیم تا در ادامه راحت‌تر سراغ موضوعات جدی‌تر مثل شیءگرایی یا چندریسمانی بریم.

بنابراین در این جلسه بیشتر روی مقدمات تمرکز 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 خواهند 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

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 **Sun** توسط شرکت **Oracle** خریداری شد و از آن زمان تاکنون، اوراکل مالک اصلی و توسعه‌دهنده جاوا است.

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 **JavaScript**، **Delphi** و **PHP** 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.

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.

Timeline– History Of Java

1991 — Birth of Java as the Green Project

1995 — Official launch of Java; applets revolutionize web development

1997 — JCP (for open collaboration), JDK 1.1, and JVM standardization begins

1998 — Introduction of Java 2 (SE, EE, ME)

2004 — Java 5– adds generics, annotations, and enhanced for-loop

2006 — Java becomes open-source under the GNU General Public License

2009 — Oracle acquires Sun Microsystems (takes over Java)

ORACLE

2014 — Java 8 introduces lambdas, Stream API, and Date-Time API

2017 — Java 9 introduces modules (Project Jigsaw)

2021 — Java 17 LTS version released with performance improvements

2023 — Java 20 & 21 refine virtual threads and tools

unstop

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.

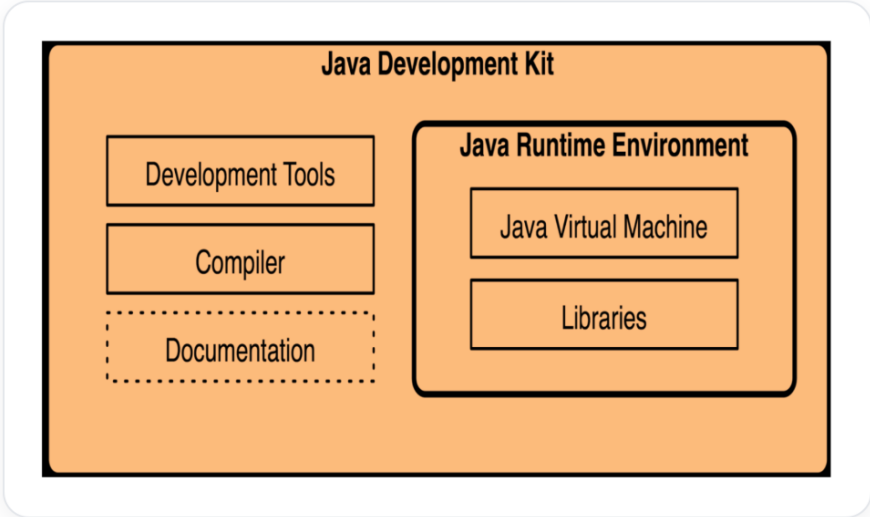
- نوشتن و ویرایش کد منبع (Source Code)
- کامپایل برنامه
- بارگذاری برنامه در حافظه
- درستی‌سنجی (Verification)
- اجرا توسط ماشین مجازی جاوا (JVM)

این فرایند به برنامه‌نویس کمک می‌کند کد نوشته‌شده 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.

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



JDK یک 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 (مثلاً Notepad یا WordPad)
- انجام اصلاحات لازم و ذخیره فایل در حافظه
- حفظ پسوند `.java` برای فایل

فایل منبع صرفا یک فایل متنی ساده است و بدون کامپایل قابلیت اجرا ندارد.

```java
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 زمان کامپایل و زمان اجرا
- مدیریت پروژه و فایل‌ها

eclipse

NetBeans

Visual Studio Code

IJ

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 اجرا در JVM است.
- بایت‌کد مستقل از سیستم‌عامل بوده و روی هر دستگاهی با JVM قابل اجراست.

ماشین مجازی جاوا (JVM)

JVM بایت‌کد تولیدشده 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 **JDK** و **Java Platform**
- شبیه‌ساز سخت‌افزار و سیستم‌عامل برای اجرای بایت‌کد
- امکان اجرای یک برنامه روی سیستم‌های مختلف بدون تغییر
- برخلاف زبان‌های وابسته به سخت‌افزار، برنامه‌های جاوا قابل حمل هستند.

فاز سوم: بارگذاری برنامه در حافظه

در این مرحله، فایل‌های `.class` حاوی بایت‌کد 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.
- بارگذاری موفق در این مرحله پیش‌نیاز ادامه اجرای برنامه است.

فاز چهارم: درستی‌سنجی

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
- جلوگیری از بروز خطاهای زمان اجرا (Runtime Errors)

نتیجه: تنها بایت‌کدهای معتبر اجازه اجرا پیدا می‌کنند.

فاز پنجم: اجرای برنامه

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

- تفسیر ساده: اجرای بایت‌کد به صورت خط‌به‌خط
- JIT Compiler: تبدیل بخش‌های پرکاربرد بایت‌کد به زبان ماشین برای افزایش سرعت
- امکان اجرای یک برنامه در سیستم‌های مختلف بدون تغییر کد
- همچنین 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 متد `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 متد `main` باشد؛ این متد نقطه شروع اجرای برنامه است.
- ساختار متد اصلی به صورت `public static void main(String[] args)` تعریف می‌شود.
- درون متد `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.

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 رزرو شده** جاوا باشد.
- نمی‌تواند مقادیر false ، true یا null باشد.
- هیچ محدودیتی روی طول شناسه وجود ندارد.
- 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

نکته مهم: 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 `boolean`، `double`، `int`).
- 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.

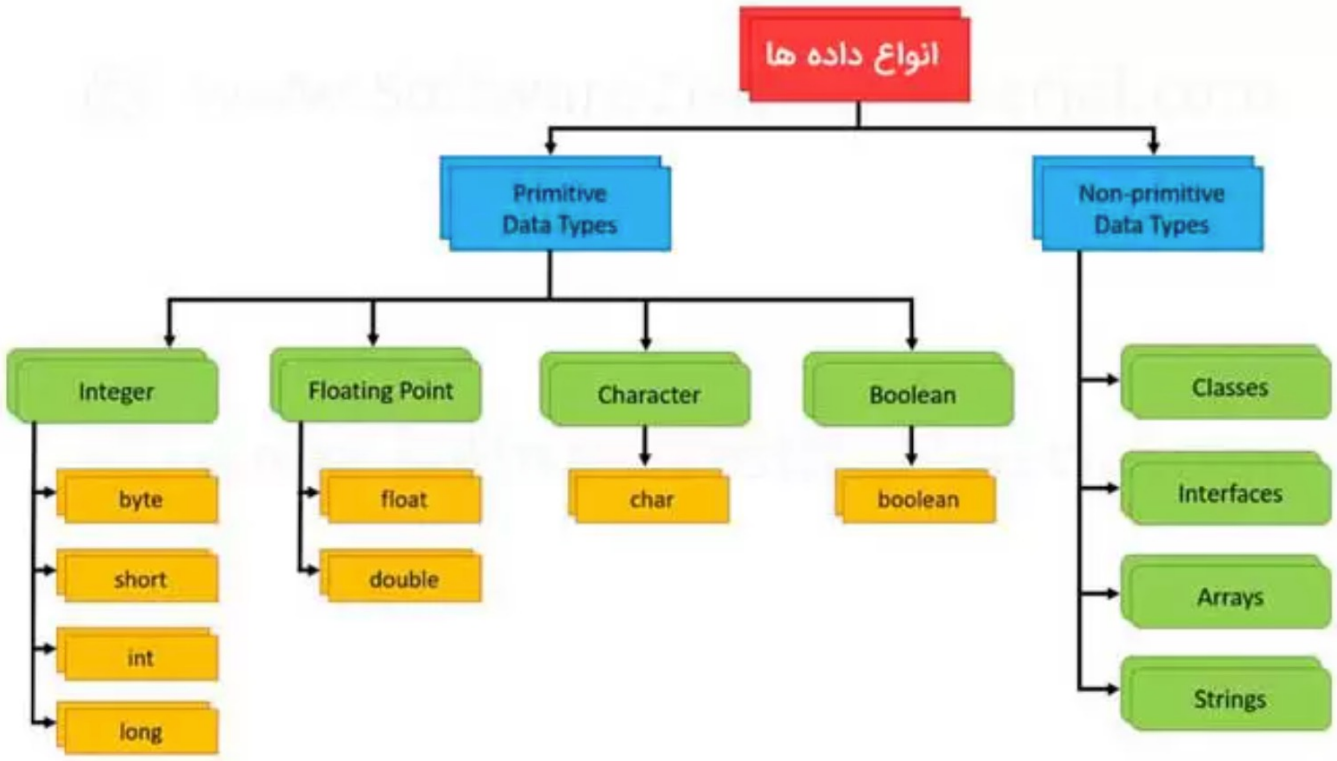
مثال:

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

انواع داده در جاوا

در جاوا، داده‌ها به دو دسته کلی تقسیم می‌شوند:

- **انواع اولیه (Primitive Types):** مانند `boolean`، `double`، `int` و غیره.
- **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
-128 to 127	bits 8	0	Signed integer	byte
-32768 to 32767	bits 16	0	Signed integer	short
-2^31 to 2^31-1	bits 32	0	Signed integer	int
-2^63 to 2^63-1	bits 64	0	Signed integer	long
~±3.4E38	32 bits	0.0	IEEE 754 floating point	float
~±1.7E308	64 bits	0.0	IEEE 754 floating point	double

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:

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

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:

- `5 / 2` ← مقدار صحیح **2**
- `5.0 / 2` ← مقدار اعشاری **2.5**
- `5 % 2` ← باقی‌مانده تقسیم **1**

عملگر باقی‌مانده (Modulo)

عملگر `%` برای محاسبه باقی‌مانده تقسیم به‌کار 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

عملگر `%` 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`

نکته: همیشه پرانتز بالاترین اولویت را دارد.

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)
```

عملگرهای تساوی و رابطه‌ای

عملگر ریاضی استاندارد	عملگر در جاوا	نمونه در جاوا	معنی شرط
=	==	x == y	x برابر با y است
≠	!=	x != y	x نابرابر با y است
>	>	x > y	x بزرگ‌تر از y است
<	<	x < y	x کوچک‌تر از y است
≥	>=	x >= y	x بزرگ‌تر یا مساوی y است
≤	<=	x <= y	x کوچک‌تر یا مساوی y است

شرکت‌پذیری

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

- `x = y = z = 17` به صورت `x = (y = (z = 17))` اجرا می‌شود (راست به چپ).
- `72 / 2 / 3` به صورت `(72 / 2) / 3` اجرا می‌شود (چپ به راست).

نمادهای میان‌بری

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

```
int i = 10;
int newNum = 10 * i++;
System.out.println(newNum);



int i = 10;
int newNum = 10 * (++i);
System.out.println(newNum);
```

عملگر + روی رشته‌ها

در جاوا، عملگر + برای **رشته‌ها (String)** سربارگذاری شده و امکان **به‌هم‌چسباندن رشته‌ها** را فراهم می‌کند.

اگر یکی از عملوندها رشته باشد، سایر عملوندها هم به رشته تبدیل می‌شوند.

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

برنامه محاسبه مساحت دایره

مثال زیر برنامه‌ای است که مساحت دایره را با استفاده از شعاع ورودی محاسبه و چاپ می‌کند:

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

```
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:

متد	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
`nextLine()`	خواندن یک خط کامل از ورودی

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

برنامه‌نویسی 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.

این تمرین‌ها کمک می‌کنند تا با نحوهٔ دریافت داده از کاربر و انجام محاسبات پایه در جاوا آشنا شوید.

```java
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);
  }
}
```

```java
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 :خروجی
System.out.println(1.0 - 0.9);
// 0.09999999999999998 :خروجی
```

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

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:

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

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

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

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

نماد علمی (Scientific notation):

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

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
```

سرریز داده‌ها (Overflow)

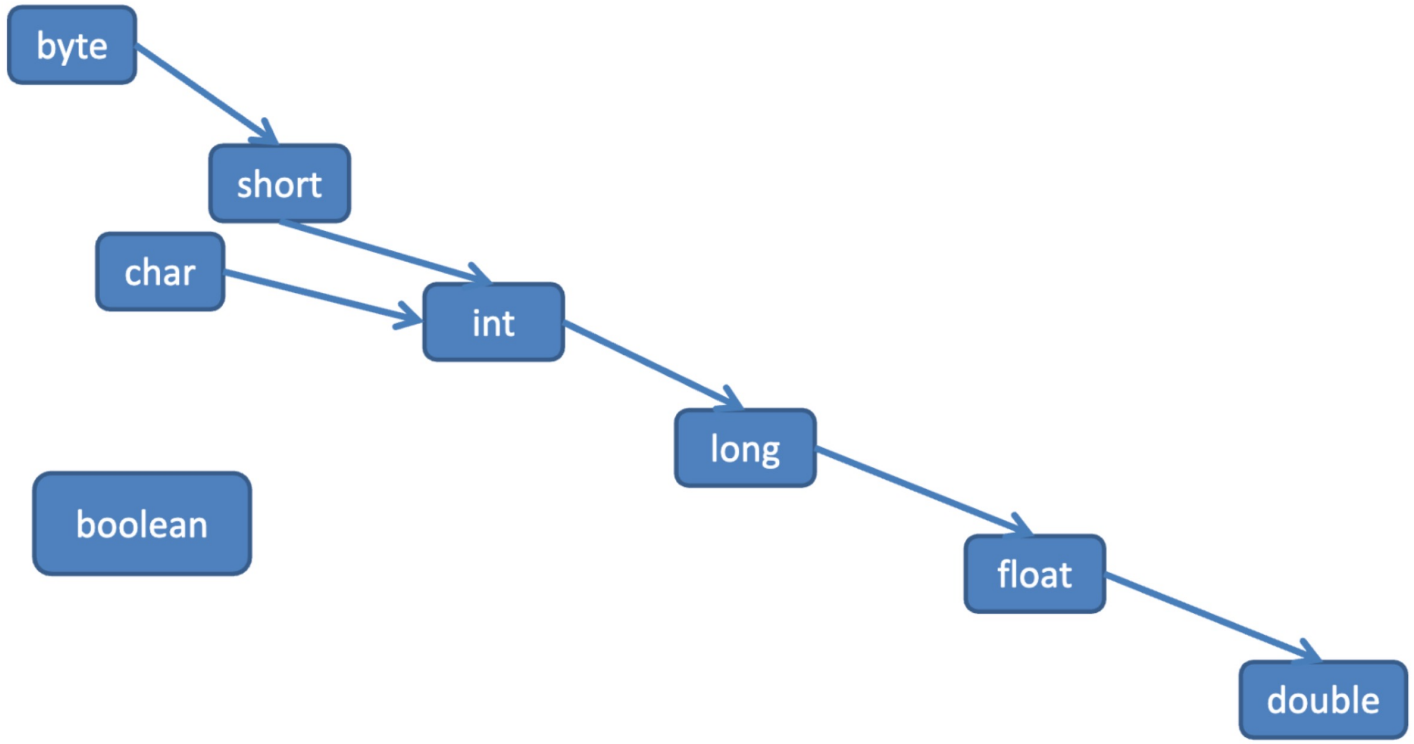
نمی‌توان مقداری بزرگ‌تر از ظرفیت یک نوع داده در آن قرار داد. در این حالت سرریز رخ می‌دهد و بخشی از مقدار از دست می‌رود.

مثال:

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

تبدیل نوع ضمنی

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



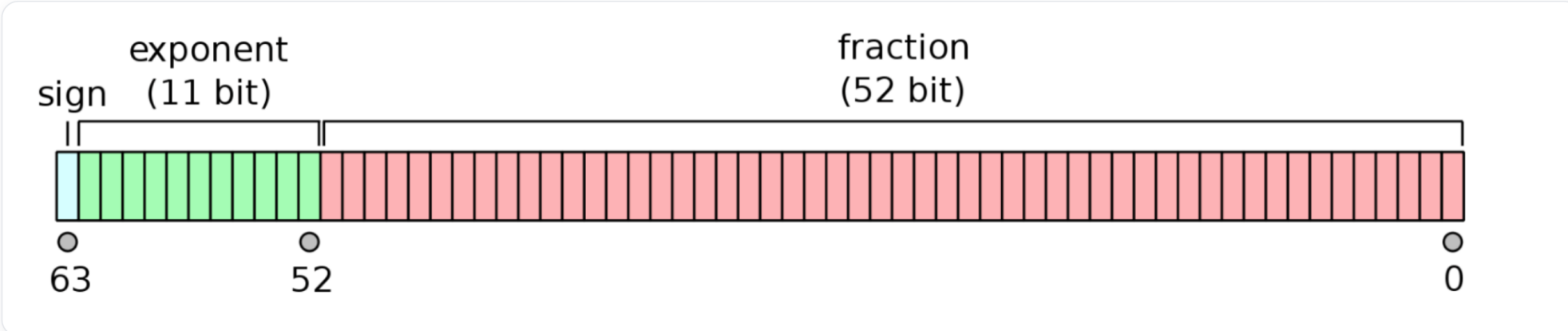
تبدیل نوع عددی (numeric type conversion)

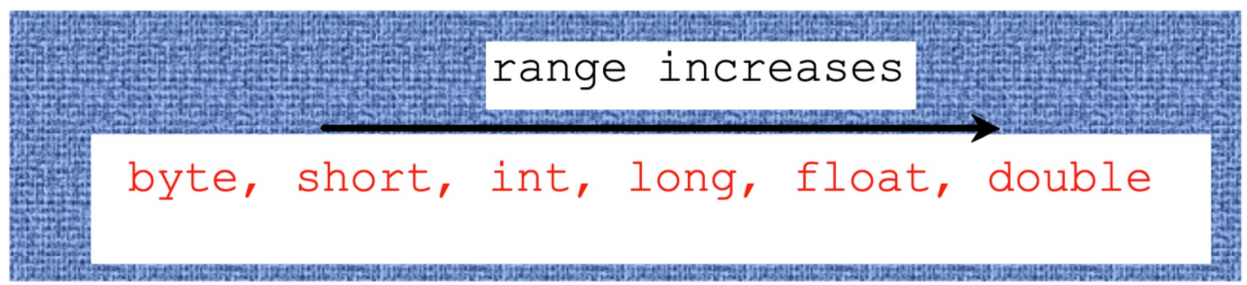
Implicit casting (Type widening)

```
double d = 3;
```

Explicit casting (Type narrowing)

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





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**: نیاز به cast مستقیم دارد.
- **Y***: تبدیل ممکن است باعث از دست رفتن دقت 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 تقریبی از اعداد 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
char ch = 'a';
System.out.println(++ch); // b
```

نکته: عملگرهای افزایشی و کاهشی می‌توانند روی متغیرهای char اعمال شوند تا کاراکتر بعدی یا قبلی Unicode را تولید کنند.

توی بلاک زیر کد بزنید و تست کنید!

فرمت Unicode

- کاراکترهای جاوا از Unicode استفاده می‌کنند.
- Unicode یک فرمت ۱۶ بیتی است که برای نمایش متون به زبان‌های مختلف استفاده می‌شود.
- کدهای Unicode از دوبایت (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;
```

کاراکترهای ASCII

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

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	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		

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.

- تفاوت JDK و JRE در چیست و هرکدام چه کاربردی دارند؟
- در فرآیند اجرای یک برنامه جاوا، مرحلهٔ Bytecode Verification چه نقشی دارد؟
- اگر دستور `int i = 10; int newNum = 10 * i++;` اجرا شود، خروجی چیست و چرا؟
- تفاوت بین عملگرهای `==` و `=` در جاوا چیست؟
- عملگر % چه کاربردهایی دارد؟ یک مثال واقعی بزنید.
- چرا جاوا را یک زبان قابل حمل (Platform Independent) می‌نامند؟

پایان

اگه سوال و پیشنهادی داشتید می‌تونید از هر دو طریق زیر با من در ارتباط باشید. :)

Email: me@ghaem.me
Telegram: @Ghaem