



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

برنامه سازی پیشرفته و کارگاه

شروع برنامه نویسی جاوا

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

آرمان حسینی

بهار ۱۴۰۳

فهرست

۳	مقدمه
۳	نترسید و چیزهای مختلف رو امتحان کنید!
۴	اولین برنامه
۶	کدی که اجرا کردیم
۸	متغیرها
۱۱	لیترال‌ها
۱۳	ثابت‌ها
۱۴	شرط‌ها
۱۵	حلقه‌ها
۱۷	متدها
۱۹	بلوک‌ها
۲۱	ورودی
۲۳	Command-Line Arguments
۲۷	غیرفعال کردن Inline Completion در IntelliJ
۲۸	چه چیزی یاد گرفتیم؟

مقدمه

این داکيومنت برای آشنایی اولیه با syntax زبان جاوا طراحی شده. قراره کمی باهاش کدنویسی کنیم و مفاهیم پایه‌ای رو یاد بگیریم. ممکنه موقع کار با جاوا، یاد زبان‌هایی مثل C یا C++ بیفتید، و این طبیعی هست چون syntax این زبان‌ها شباهت زیادی به هم دارن. بسیاری از کارهایی که اینجا انجام می‌دیم، شبیه چیزی هست که ممکنه قبلاً در C تجربه کرده باشید.

لپ‌تاپ‌هاتون رو آماده کنید، ادیتورهایتون رو باز کنید و قدم به قدم با ما پیش بیاید تا اولین تجربه برنامه‌نویسی با جاوا رو کسب کنید. اگر جایی از این داکيومنت براتون گنگ بود، حتماً سرچ کنید، از دستیارهای آموزشی بپرسید یا از ChatGPT کمک بخواید تا براتون توضیح بده.

توجه کنید که این داکيومنت، برای دست به کد شدن شماست. بعضی از کلیدواژه‌های جاوا مثل public، class، static و غیره این‌جا توضیح داده نمی‌شن و طبیعیه که نسبت بهشون یه خورده گیج باشین. نگران نباشین! اون‌ها رو توی جلسات بعدی بهتون توضیح می‌دیم. تا اون موقع، اگر دوست داشتید می‌تونید از ما راجع بهشون بپرسین یا توی اینترنت در موردشون بخونین.

نترسید و چیزهای مختلف رو امتحان کنید!

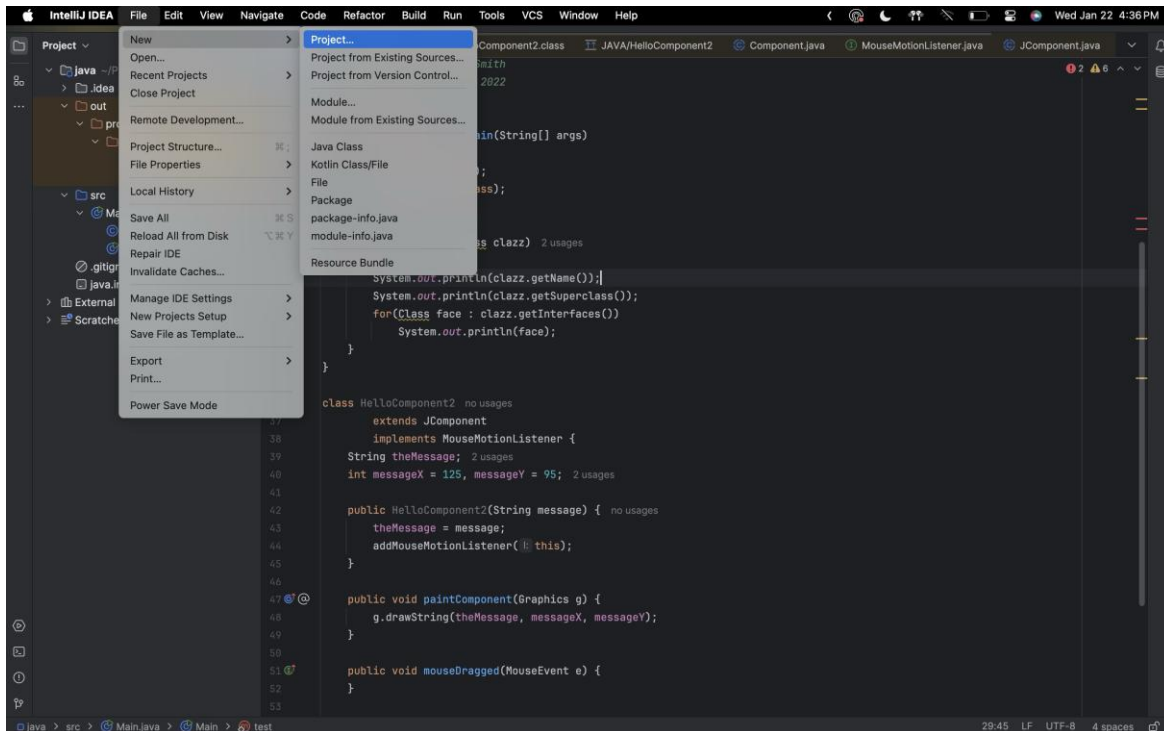
قبل از شروع داکيومنت، خوبه که این رو بدونید که شما قراره خراب‌کاری‌های زیادی توی کدهای اولتون بکنید! چیزهای مختلف رو امتحان می‌کنین، خطاهایی می‌خورید که هیچ ایده‌ای ندارین از کجا اومدن یا حتی اگر می‌دونین از کجا، نمی‌دونین چجوری باید از شرشون خلاص بشین.

نترسید! کدها رو بشکونید! این خطاها و مشکلات، بخشی از فرآیند یادگیری شما هستن. راجع بهشون اول توی اینترنت سرچ کنید، اگر نشد، از ChatGPT بخواین براتون توضیح بده و اگر حتی اون هم نشد، از تدریس‌یارتون بپرسید.

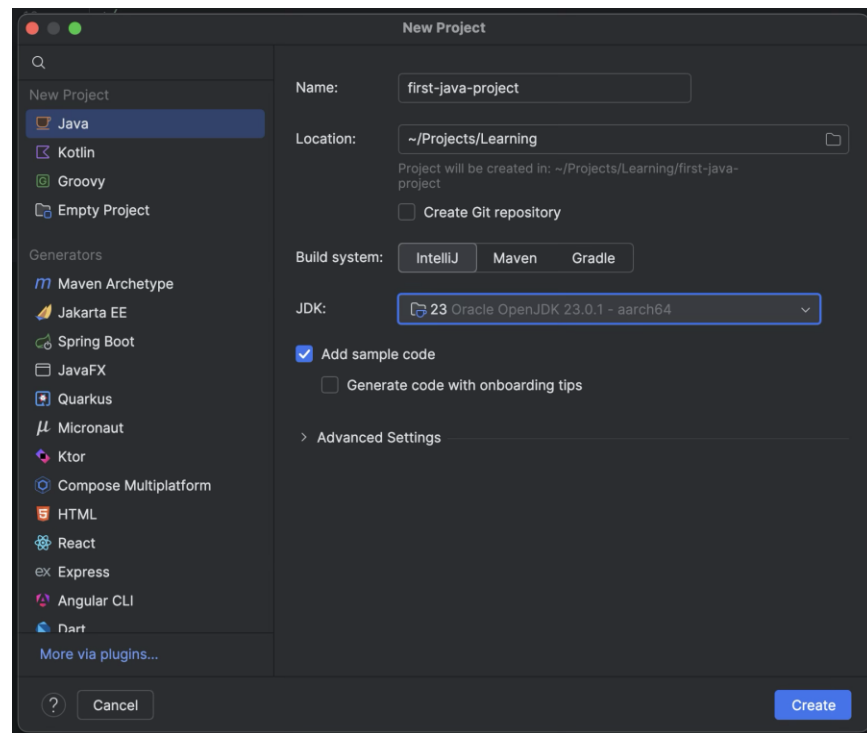
چیزهای جدید رو امتحان کنید، توی کدهای مختلف فضولی کنین و اگر هم چیز خوب و جدیدی پیدا کردین، با باقی به اشتراکش بذارین. فو‌قش اینه که به خطا می‌خورین و با ما رفعش می‌کنین.

اولین برنامه

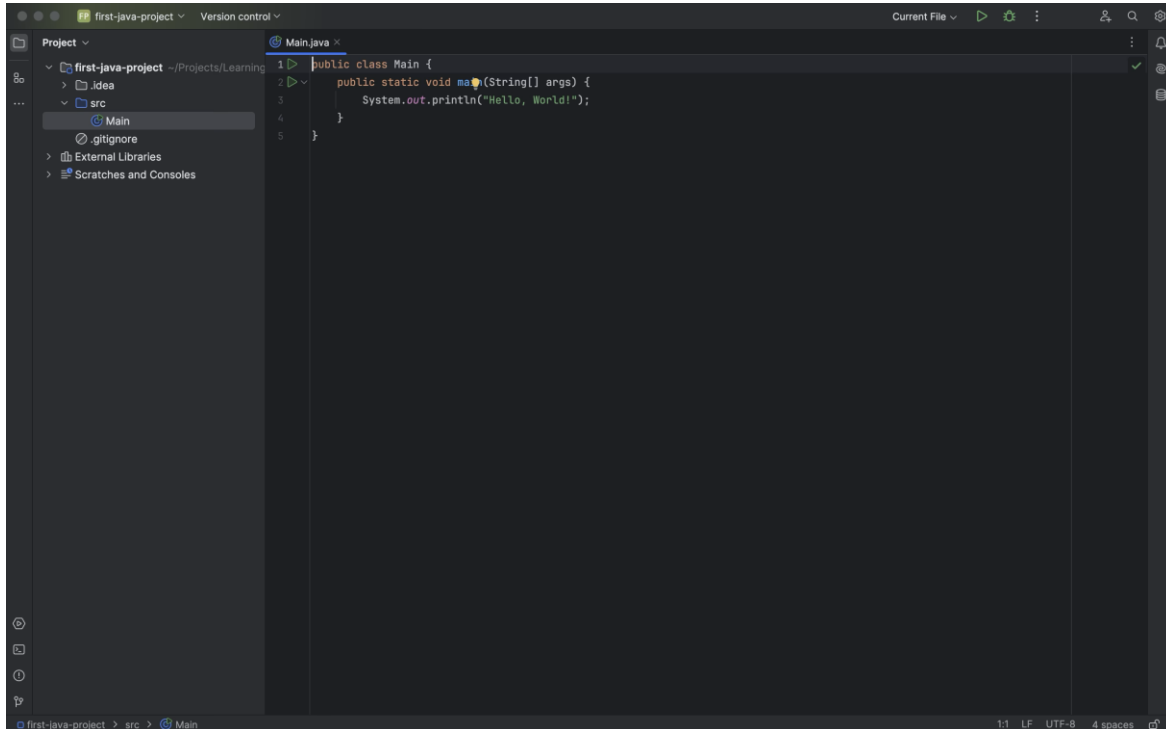
برای درست کردن اولین برنامه جاواتون، IntelliJ رو باز کنید و از مسیر زیر، اولین پروژه‌تون رو ایجاد کنید:



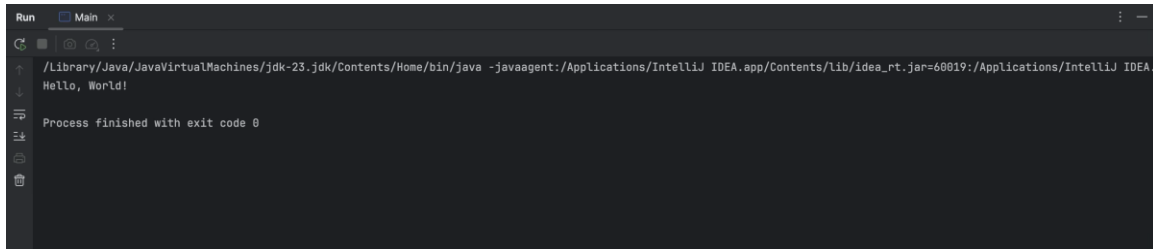
یه صفحه مثل تصویر زیر براتون باز می‌شه. اسم پروژه‌تون رو انتخاب کنید، مطمئن بشید مسیر ذخیره‌سازی درسته و حتماً چک کنید که همون JDKی که قبلاً دانلود کردید، انتخاب شده باشه. من خودم نسخه 23.0.1 رو نصب کردم و برای این پروژه هم از همین نسخه استفاده کردم:



با این کار، اولین پروژه جاوای شما درست می‌شه! همچنین صفحه‌ای رو باید ببینین:



این، اولین برنامه‌ی شماست. با زدن اون دکمه‌ی مثلثی شکل بالای صفحه، برنامه رو run کنید. اگر درست compile و run شد، باید خروجی‌ای مشابه این ببینید:



تبریک! اولین برنامه‌ی جاواتون رو اجرا کردین!

کدی که اجرا کردیم

بیاین ببینیم این کد که اجرا شد، چی بود. منطقاً باید چنین کدی روی صفحه‌ی ادیتورتون باشه:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

توی این کد، یک کلاس به اسم Main می‌بینید که داخلش یه تابع به اسم main قرار داره. بعداً بهترتون یاد می‌دیم که کلاس چی هست، public چه معنی‌ای داره و چطور می‌تونید کلاس‌های خودتون رو بسازید. ولی فعلاً، تمام توابع و کدهایی که می‌نویسید، توی همین کلاس Main قرار می‌گیره و نیازی نیست الان درگیر یادگیری مباحث مربوط به کلاس‌ها بشید.

```
public class Main {
    // Your code here
}
```

داخل این class، یه تابع به اسم main می‌بینید. توی جاوا، ما به تابع‌هامون می‌گیم Method (حواستون باشه که Function اصطلاح دقیقی برای اون‌ها نیست). این متد شبیه همون تابع main توی C هست که قبلاً باهاش آشنا شدید و همون جاییه که برنامه‌تون از اون شروع به اجرا می‌کنه:

```
public static void main(String[] args) {
    // Your program starts from here
}
```

شاید براتون سوال شده باشه که public static void و String[] args دقیقاً چی هستن. کلیدواژه‌های public و static جزو مفاهیمی هستن که وقتی وارد

مباحث شی‌گرایی بشیم بیشتر در موردشون صحبت می‌کنیم، پس فعلاً لازم نیست نگران‌شون باشید.

اما اون void احتمالاً از تجربه‌های قبلی‌تون در برنامه‌نویسی براتون آشنا باشه. وقتی void توی یه متد استفاده می‌شه، یعنی اون متد هیچ مقداری رو به عنوان خروجی برنمی‌گردونه. اینجا هم همین مفهوم صدق می‌کنه؛ یعنی متد main چیزی رو return نمی‌کنه.

توی main، شما کد زیر رو می‌بینین:

```
System.out.println("Hello, World!");
```

این تکه کد برای چاپ "Hello, World!" نوشته شده. حالا اگه این خط رو پاک کنین و دوباره بنویسینش، متوجه می‌شین که دستور println می‌تونه علاوه بر رشته‌ها، انواع دیگه‌ای مثل int و boolean رو هم چاپ کنه. یعنی به راحتی می‌تونین با همون دستور، مقادیر مختلف رو به خروجی چاپ کنین:



این یکی از ویژگی‌های خوب IntelliJ هست که به شما این امکان رو می‌ده که نوع و نام پارامترهای یک تابع و همچنین خروجی اون رو ببینید. این قابلیت می‌تونه خیلی در نوشتن کد به شما کمک کنه. حالا که صحبت از int و boolean و سایر نوع‌های داده شد، بیایید اولین متغیرهای خودمون رو توی جاوا تعریف کنیم.

متغیرها

کد روی صفحه رو به کد زیر تغییر بدین:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;

        System.out.println(a);
    }
}
```

توی این کد، متغیر a رو از جنس int تعریف کردیم و مقدار اولیه ۱۰ رو بهش دادیم و بعد اون رو چاپ کردیم. به int، boolean، float و انواع مشابهشون، data type می‌گیم. data type های اصلی جاوا که بهشون Primitive data types هم گفته می‌شه، به همراه رنجی که پوشش می‌دن، توی جدول زیر آورده شده. حواستون باشه که اینجا هم مثل C، نباید از رنجی که برای هر تایپ تعریف شده خارج بشید.

Type	Definition	Approximate range or precision
boolean	Logical value	true or false
char	16-bit, Unicode character	64K characters
byte	8-bit, signed integer	-128 to 127
short	16-bit, signed integer	-32,768 to 32,767
int	32-bit, signed integer	-2.1e9 to 2.1e9
long	64-bit, signed integer	-9.2e18 to 9.2e18
float	32-bit, IEEE 754, floating-point value	6-7 significant decimal places
double	64-bit, IEEE 754	15 significant decimal places

شما می‌تونید به شکلی مشابه، متغیرهایی از جنس هر کدوم از این type ها ایجاد و چاپ کنید:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        double b = 3.14;
        boolean c = true;
        char d = 'a';

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
    }
}
```

خروجی این کد، همون‌طور که انتظار داریم، به شکل زیر هستش:


```

/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Cor
10
3.14
true
a
Process finished with exit code 0

```

شما توی java، به type برای متغیرها تون به اسم String هم دارین، که البته جزو primitive data typeها نیست. مثلا Hello, World ای که توی اولین مثال print کردین، نمونه‌ای از همین type بود. می‌تونید به شکل زیر به String تعریف کنید:

```

public class Main {
    public static void main(String[] args) {
        String str = "Hello, World!";

        System.out.println(str);
    }
}

```

علاوه بر این، جاوا به دستور جدا برای print کردن، به سبک printf ای که توی C داشتین، داره:

```

String name = "Arman";
int age = 27;

System.out.printf("Name: %s\nAge: %d\n", name, age);

```

شما می‌تونید تمام عملیات‌های ریاضی که توی زبان‌های دیگه مثل C روی متغیرها انجام می‌دادین رو این‌جا هم انجام بدین، طبیعتا جاوا هم مثل همه زبان‌های دیگه، اولویت ریاضی بین اون‌ها رو رعایت می‌کنه:

Precedence	Operator	Operand type	Description	Precedence	Operator	Operand type	Description
1	++, --	Arithmetic	Increment and decrement	6	==, !=	Object	Equality and inequality of reference
1	+, -	Arithmetic	Unary plus and minus	7	&	Integral	Bitwise AND
1	~	Integral	Bitwise complement	7	&	Boolean	Boolean AND
1	!	Boolean	Logical complement	8	^	Integral	Bitwise XOR
1	(type)	Any	Cast	8	^	Boolean	Boolean XOR
2	*, /, %	Arithmetic	Multiplication, division, remainder	9		Integral	Bitwise OR
3	+, -	Arithmetic	Addition and subtraction	9		Boolean	Boolean OR
3	+	String	String concatenation	10	&&	Boolean	Conditional AND
4	<<	Integral	Left shift	11		Boolean	Conditional OR
4	>>	Integral	Right shift with sign extension	12	?:	N/A	Conditional ternary operator
4	>>>	Integral	Right shift with no extension	13	=	Any	Assignment
5	<, <=, >, >=	Arithmetic	Numeric comparison				
5	instanceof	Object	Type comparison				
6	==, !=	Primitive	Equality and inequality of value				

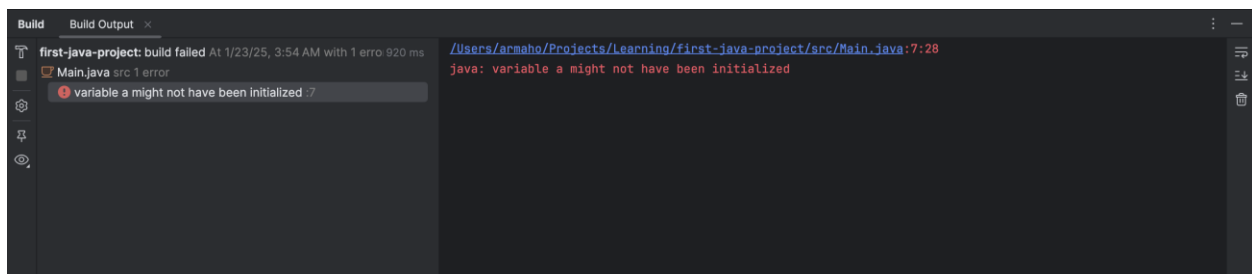
می‌تونید متغیرها رو، موقع ساختشون مقداردهی نکنید و صرفاً به نوشتن typeشون اکتفا کنید:

```
public static void main(String[] args) {
    int a;
}
```

ولی حتماً باید قبل از استفاده از اون‌ها، بهشون یه مقدار بدین. وگرنه جاوا نمی‌ذاره که برنامه‌تون رو compile کنین. به عنوان مثال، اگر تلاش کنید تا کد زیر رو اجرا کنین:

```
int a;
System.out.println(a);
```

جاوا خطای زیر رو بهتون می‌ده:



وقتی یه متغیر رو موقع تعریف کردنش مقداردهی می‌کنید، می‌تونید از کلیدواژه var استفاده کنید تا کامپایلر جاوا خودش نوع متغیر رو براساس مقداری که بهش داده میشه، تشخیص بده. اینطوری دیگه نیازی به مشخص کردن دستی نوع متغیر نیست.

```
var a = 10;
var b = 20;

System.out.println(a + b);
```

توی کد بالا، چون مقادیر ۱۰ و ۲۰ رو به a و b دادین، کامپایلر جاوا متوجه می‌شه که این دو متغیر از نوع int هستن و باهاشون مثل int برخورد می‌کنه. نکته اینه که var یه data type جدید نیست، فقط یک روش برای راحت‌تر کردن کار شماست که دیگه لازم نیست برای همه متغیرها، typeشون رو بنویسید. کامپایلر خودش نوع متغیرها رو براساس مقدار اولیه‌ای که می‌گیرن، تشخیص می‌ده.

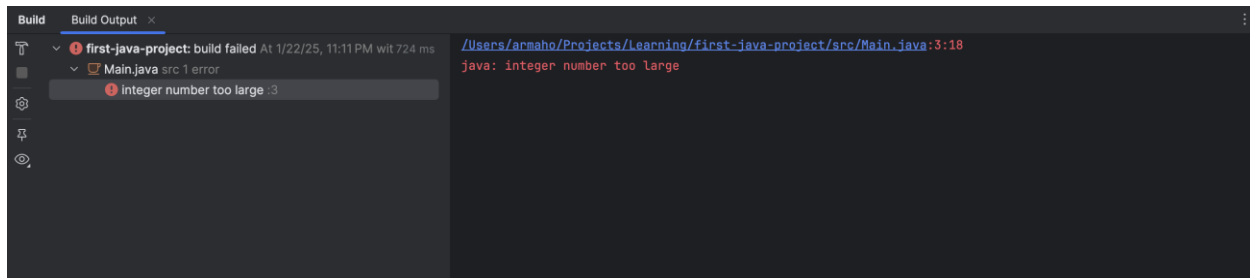
لیترال‌ها

کد زیر رو توی IDE اتون اجرا کنین:

```
public static void main(String[] args) {
    long a = 999999999999;

    System.out.println(a);
}
```

با تلاش برای اجرای اون، به خطای زیر می‌خورین:



این خطا، یه خورده عجیبه. با این که شما برای این عدد خیلی بزرگ، از یه متغیر long استفاده کردین که باید اون رو توی خودش جا بده، ولی باز هم به خطای number too large می‌خورین. چرا؟

خطی که a رو توش تعریف کردین با خط زیر جایگزین کنید:

```
long a = 999999999999L;
```

مشکل حل شد، درسته؟ خطای قبلی به خاطر نوع متغیر a نبود. متغیر long می‌تونه اعداد خیلی بزرگتری رو هم نگه داره. مشکل این بود که خود جاوا نمی‌دونست باید با عدد ۹۹۹۹۹۹۹۹۹۹۹ چطور برخورد کنه. آیا باید اون رو به عنوان int، long یا حتی double در نظر بگیره؟ توی کد دوم، با گذاشتن L در انتهای عدد، شما به جاوا گفتین که این عدد از نوع long هست و باید باهاش مثل یک long برخورد کنه.

به اعداد و کاراکترهایی که توی برنامه‌تون hard code می‌کنین، literal می‌گن. کد زیر، شامل ۳ تا literal:

```
int r = 10;
char next_line = '\n';
System.out.printf("Circle area: %f%c", 3.14 * r * r, next_line);
```

این literal‌ها به ترتیب ۱۰، '\n' و ۳/۱۴ هستن. همونطور که قبلاً گفتیم، literal‌ها هم مثل متغیرها type دارن. مثلاً عدد ۱۲۳ از نوع int هست و 123L از نوع long. همینطور وقتی عددی مثل ۳/۱۴ رو به

صورت اعشاری تعریف می کنید، به طور پیش فرض جاوا نوع اون رو double در نظر می گیره و اگه بخواید که نوعش float باشه باید از 3.14F استفاده کنید. همچنین می تونید از نمایش علمی برای اعداد اعشاری هم استفاده کنید:

```
double d = 8.31;  
double e = 3.00e+8;  
float f = 8.31F;  
float g = 3.00e+8F;
```

می تونین وقتی دارین با اعداد خیلی بزرگ سرو کله می زنین، با استفاده از _ ارقام اون ها رو جدا کنین تا راحت تر خونده بشن:

```
int a = 999999999; // Very hard to read!  
int b = 999_999_999; // Easy :)  
  
System.out.printf("a = %d, b = %d\n", a, b);
```

توی کد بالا، مقادیر a و b یکسان هستن، ولی یکی شون رو خیلی راحت تر می شه خوند!

ثابت‌ها

توی کد زیر، به constant به اسم MAX_AGE رو توی جاوا تعریف کردیم:

```
public class Main {  
    public static final int MAX_AGE = 99;  
  
    public static void main(String[] args) {  
        int age = 102;  
  
        if (age > MAX_AGE) {  
            System.out.println("You're too old!");  
        }  
    }  
}
```

این جا هم لازم نیست با public static ای که قبل MAX_AGE اومده کاری داشته باشیم، توی داکویمنت‌های بعدی حسابی با اون‌ها آشنا می‌شیم و باهاشون کار می‌کنیم. کلیدواژه final به شما این امکان رو می‌ده که یک constant تعریف کنید. constant‌ها، بعد از تعریف و مقداردهی‌شون، هیچ وقت مقدارشون عوض نمی‌شه.

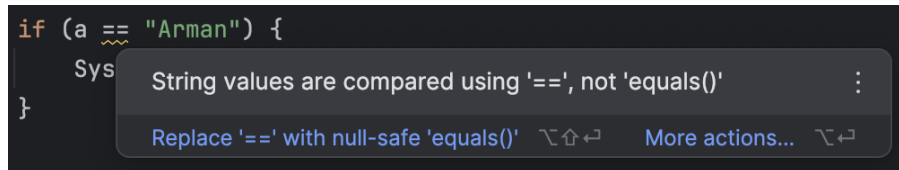
شرط‌ها

شرط‌های java، خیلی شبیه C هستن، می‌تونین با سینتکس زیر از اون‌ها استفاده کنین:

```
int radius = 10;

if (radius <= 20) {
    System.out.println("Radius is too small");
} else if (radius >= 50 && radius <= 100) {
    System.out.println("Radius is too big");
} else {
    System.out.println("Other stuff");
}
```

فقط حواستون باشه که String‌ها رو با == مقایسه نکنید. اگر این کار رو بکنید، خود IntelliJ بهتون یه warning می‌ده و ازتون می‌خواد که از متد equals استفاده کنین:



```
if (a == "Arman") {
    Sys
}
```

String values are compared using '==', not 'equals()'

Replace '==' with null-safe 'equals()' More actions...

شیوهٔ درست مقایسهٔ String‌ها توی جاوا به این شکل هست:

```
if (a.equals("Arman")) {
    // Do something
}
```

علت این اتفاق، این هستش که همون‌طور که قبلاً گفتیم، String یک primitive type نیست و a reference type، توی جلسات بعدی هر دوی این‌ها رو تعریف می‌کنیم و تفاوت‌هاشون رو با هم می‌گیم.

by the way، همیشه سعی کنین به warning‌ها، hint‌ها و کلاً ابزارهایی که IntelliJ بهتون می‌ده توجه کنید، می‌تونید کدتون رو باهاشون بهتر کنید و ازشون چیز میز یاد بگیرید.

حلقه‌ها

توی جاوا، دو نوع اصلی حلقه داریم، `for` ها و `while` ها. اگر اون‌ها رو از C یا زبان‌های دیگه یادتونه، تقریباً همونه!

توی کد زیر به `for` صدتایی می‌زنیم:

```
for (int i = 0; i < 100; i++) {
    System.out.println(i);
    int j = i;
    // do any other work needed
}
```

همون‌طور که می‌بینید، `for` syntax حلقه‌ی `for` توی جاوا خیلی شبیه به زبان C ه، و هرکاری اون‌جا می‌کردین رو می‌تونین این‌جا هم بکنین. می‌تونین به جای `i++` هر عبارت مشابه دیگه که تغییری در `i` ایجاد می‌کنه رو بذارین. هر دوی `for` های زیر، درستن. اون‌ها رو اجرا کنید و خروجی‌شون رو بررسی کنین:

```
for (int i = 0; i < 100; i += 2) {
    System.out.println(i);
}
```

```
for (int i = 0; i < 100;) {
    System.out.println(i);
}
```

توی حلقه‌ی دوم، به کل `i` رو بعد از هر `iteration` تغییری نمی‌دیم. این باعث می‌شه که تا هر وقت که برنامه اجرا بشه، 0 چاپ کنیم. حتی حلقه‌ی زیر هم درست هستش:

```
for (; true;) {
    System.out.println("running forever!");
}
```

تنها چیزی که گذاشتنش توی حلقه‌ی `for` اجباری هست، شرط تکرار حلقه‌ست. مثلاً همون `true` ای که توی کد بالا گذاشتیم. هر دو بخش دیگه رو می‌تونین کنار بذارین.

حلقه‌ی `while` هم، مشابه زبان C هستش. به شکل زیر می‌تونیم یکی درست کنیم:

```
int count = 10;
while (count > 0) {
    System.out.println("Counting down: " + count);
    // maybe do other useful things
    // and decrement our count
    count = count - 1;
}
System.out.println("Done");
```

توی java، ما do while هم داریم. این نوع حلقه، شبیه a while، با این تفاوت که اول کدهای داخل while اجرا می‌شه، بعد از اون شرطش چک می‌شه، و این فرآیند تا زمانی که شرط درست باشه ادامه پیدا می‌کنه. پس کد داخل do while، حداقل یک بار اجرا می‌شه، حتی اگر شرط while غلط باشه.

```
int i = 0;
do {
    System.out.println(i);
    i++;
} while (i < 5);
```

شاید یه خورده در مورد do while گیج شده باشین. نگران نباشین، در آینده کاربردها و شکل استفاده ازش رو می‌بینین.

علاوه بر همه این‌ها، برای این که روی اعضای یک آرایه، لیست و چیزهای مشابه iterate کنیم هم یک نوع خاص از for داریم، که توی داکيومنت‌های بعدی باهاش آشنا می‌شین.

مثل C، این‌جا هم می‌تونید از break یا continue توی حلقه‌هاتون استفاده کنین:

```
for (int i = 0; i < 10; i++) {
    if (i == 5)
        continue;
    System.out.println(i);
}
```

```
while(true) {
    if (watchForErrors())
        break;
    // No errors yet so do some work...
}
// The "break" will cause execution to
// resume here, after the while loop
```

شاید حواستون نبوده باشه، ولی if‌های دو کد بالا آکولاد ندارن! توی جاوا، اگر کد داخل یک if، for، while، else یا هر چیز بلوک‌دار دیگه‌ای تک خطی باشه، لازم نیست که براش آکولاد بذارین.

متدها

می‌تونیم به شکل زیر توی java، Method ای مثل checkEmail تعریف کنیم:

```
public class Main {
    // this method will return true if the email contains the character '@'
    // and false otherwise
    public static boolean checkEmail(String email) {
        int emailLength = email.length();

        for (int i = 0; i < emailLength; i++) {
            if (email.charAt(i) == '@') { // email.charAt(i) will return the
// i-th character of the email
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        String email = "armanhosseini@gmail.com";

        if (!checkEmail(email)) {
            System.out.printf("%s is not a valid email address\n", email);
        } else {
            System.out.println("Ok");
        }
    }
}
```

وظیفه این متد اینه که بررسی کنه آیا ایمیل ورودی کاراکتر @ رو داره یا نه. احتمالاً تا این جا می‌دونید که قراره در آینده توضیح بدیم که چرا از public static استفاده می‌کنیم. فعلاً باید این رو بدونید که برای هر چیزی که داخل کلاس Main تعریف می‌کنید، باید اون رو قبل از استفاده بنویسید.

همونطور که می‌بینید، تعریف متدها توی جاوا مشابه Function های زبان C هست و از این نظر شباهت زیادی داره. پس هر کاری که توی C می‌کردید، اینجا هم همونطور عمل می‌کنید! مثال بعدی، عدد n-ام فیبوناچی رو به شکلی بازگشتی چاپ می‌کنه:

```
public class Main {
    public static int fib(int n) {
        if (n == 0)
            return 1;
        if (n == 1)
            return 1;

        return fib(n - 1) + fib(n - 2);
    }
}
```

```
public static void main(String[] args) {  
    int n = 10;  
  
    System.out.println(fib(n));  
}  
}
```

حواستون باشه که فعلا، method ها تون رو بیرون از کلاس Main تعریف نکنید.

بلوک‌ها

به اصطلاح، به هر چیزی که بین دو آکولاد {} می‌نویسید، یک block می‌گن. به عنوان مثال، کد زیر ۳ block تعریف می‌کنه:

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
    }
}
```

شاید براتون جالب باشه که برای تعریف یک block، حتی نیازی استفاده از if، for، یا امثال اون‌ها نیست و می‌تونید بدون استفاده ازشون یه block برای خودتون تعریف کنید:

```
public class Main {
    public static void main(String[] args) {
        float pi = 3.14F;

        {
            int r = 12;
            System.out.println(pi * r * r);
        }
    }
}
```

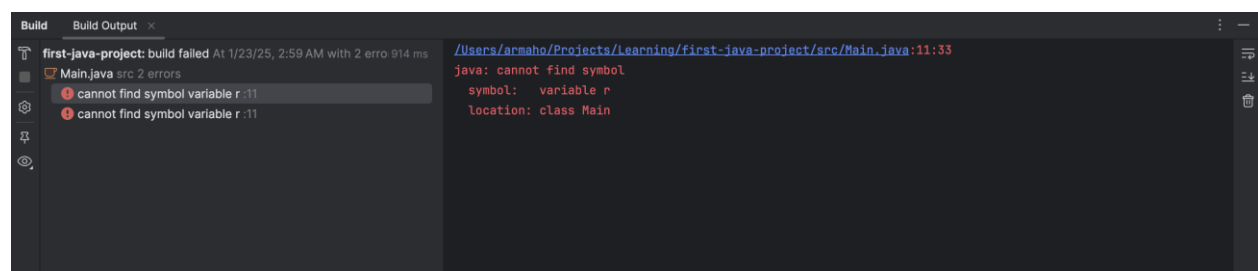
مثل زبان‌های دیگه، متغیرهایی که توی یه block تعریف می‌شن متعلق به همون block هستن یا به عبارتی، scope اون block هست. به این متغیرها، متغیرهای محلی یا local variables می‌گن. به همین خاطره که اگر کد زیر رو اجرا کنید:

```
float pi = 3.14F;

{
    int r = 12;
}

System.out.println(pi * r * r);
```

خطای زیر رو می‌گیرید:



این اتفاق، برای شما به این معنی که نمی‌تونید از متغیرهای محلی که داخل کدهای `method`، `for`، `if` یا هر چیز `block` دار دیگه‌ای، بیرون اون استفاده کنید.

ورودی

ورودی گرفتن توی java، یه مقدار با زبان‌های دیگه فرق داره. برای استفاده از اون، اول باید یه object از جنس Scanner با استفاده از System.in درست کنید. اگر جمله قبلی براتون بی‌مفهوم بود، نگراناش نباشید! وقتی شی‌گرایی رو یاد گرفتین بهتر باهاش ارتباط می‌گیرین. برای الآن، کد زیر رو بالای متد mainتون پیست کنید:

```
var scn = new Scanner(System.in);
```

دقت کنید که برای استفاده از اون، باید java.util.Scanner رو import کرده باشین. import کردن، برای استفاده از کدیه که توی فایل فعلی شما وجود نداره. کافیه خط زیر رو به ابتدای برنامه‌تون اضافه کنید:

```
import java.util.Scanner;
```

حالا می‌تونید متدهای مختلف scn رو برای خوندن ورودی کاربر استفاده کنین. مثلاً با استفاده از nextInt()، می‌تونید عددی که کاربر به عنوان ورودی می‌ده رو بخونین:

```
import java.util.Scanner;

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

        System.out.print("Enter your age: ");
        int age = scn.nextInt();

        if (age >= 18) {
            System.out.println("You're an adult now.");
        }
    }
}
```

این کد رو امتحان کنید تا ورودی و خروجی IntelliJ هم یک نگاه انداخته باشین.

برای خوندن یک String از ورودی کاربر، از nextLine() استفاده کنید:

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

        System.out.print("gmail: ");
        String email = scn.nextLine();

        if (!email.contains("@gmail.com")) {
            System.out.println("provide a valid gmail address");
        }
    }
}
```

آبجکت Scanner، methodهای مشابه دیگری هم بهتون می‌ده که می‌تونید به شکلی مشابه ازشون استفاده کنید:

Method	Description
<code>nextBoolean()</code>	Reads a <code>boolean</code> value from the user
<code>nextByte()</code>	Reads a <code>byte</code> value from the user
<code>nextDouble()</code>	Reads a <code>double</code> value from the user
<code>nextFloat()</code>	Reads a <code>float</code> value from the user
<code>nextInt()</code>	Reads a <code>int</code> value from the user
<code>nextLine()</code>	Reads a <code>String</code> value from the user
<code>nextLong()</code>	Reads a <code>long</code> value from the user
<code>nextShort()</code>	Reads a <code>short</code> value from the user

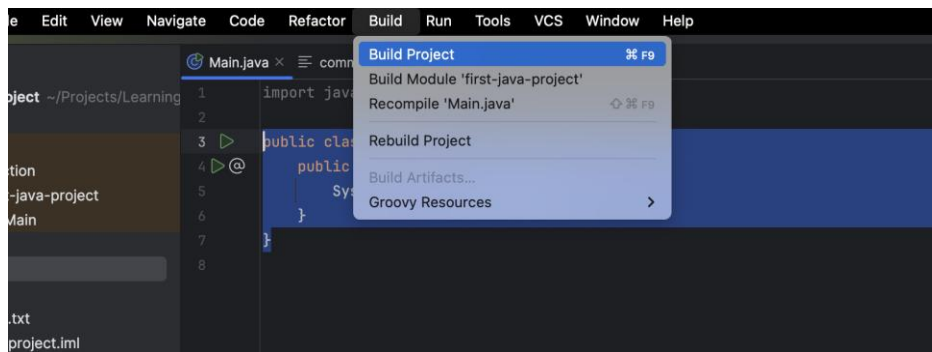
Command-Line Arguments

شاید حواستون نبوده باشه، ولی ما هنوز نگفتیم که `String[] args` توی ورودی‌های متد `main` چه کار می‌کنه. `args`، ورودی‌هایی هست که از طریق Command-Line به برنامه شما داده شدن. در ادامه، با هم کاربرد اون‌ها رو بررسی می‌کنیم.

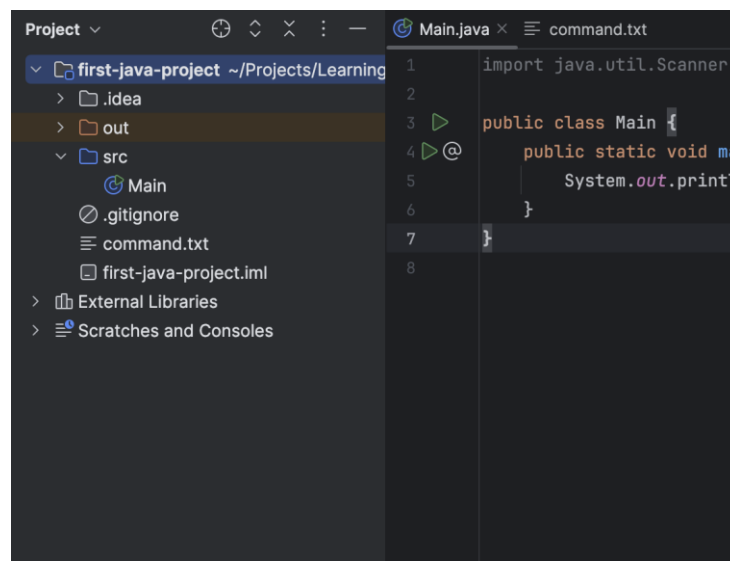
کد جاواتون رو به کد زیر تغییر بدین:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Given arg: " + args[0]);
    }
}
```

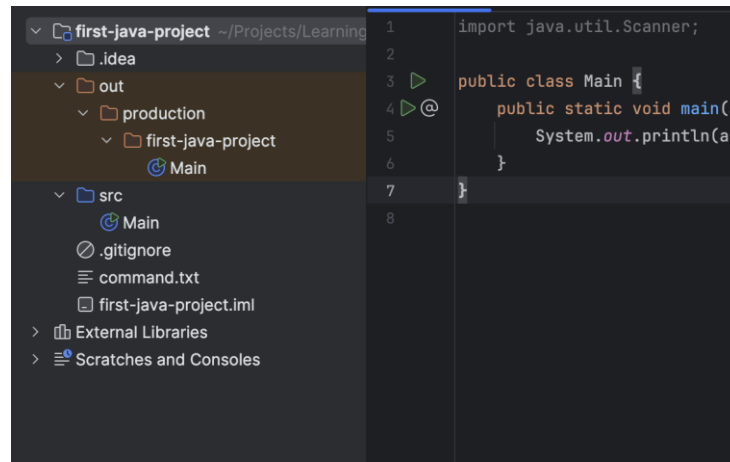
از منوی بالای IntelliJ، `Build > Build Project`، رو انتخاب کنین تا کدتون کامپایل بشه.



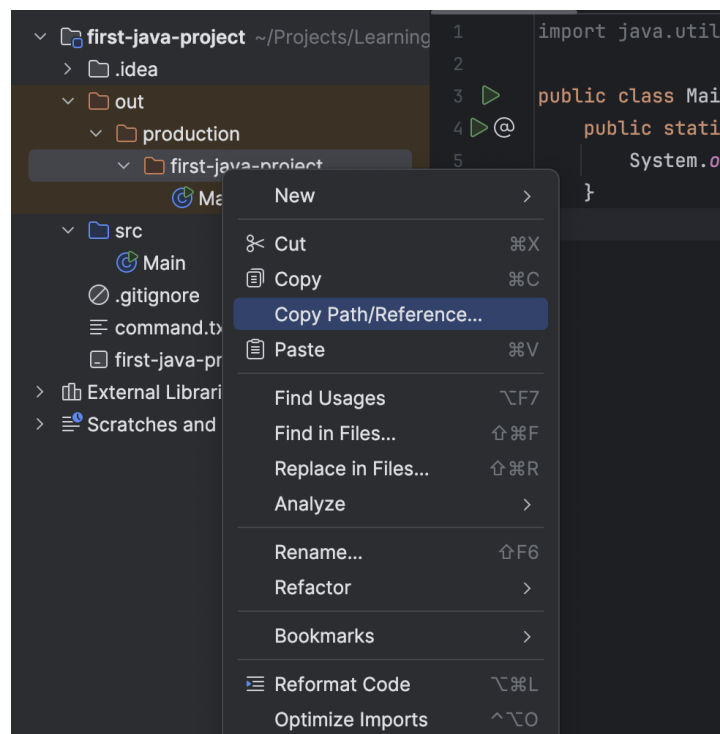
بعد از این کار، باید کنار کدهاتون یه دایرکتوری به اسم `out` درست شده باشه که شامل کد کامپایل‌شده‌تون هست:



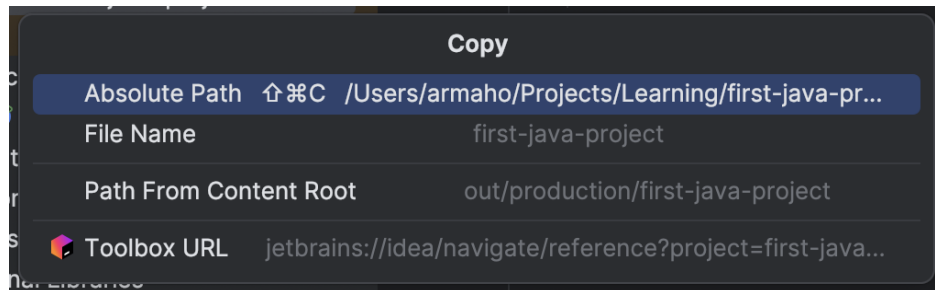
روی out کلیک کنید، و روی production هم کلیک کنید تا به دایرکتوری به اسم خود پروژه‌تون ببینید.
اونم باز کنید:



اون فایل Main که توی out می‌بینید، کد کامپایل‌شده‌تونه. اگر روش دبل‌کلیک کنید، IntelliJ براتون decompile می‌کنه و کدی مشابه کد خودتون بهتون نشون می‌ده. روی دایرکتوری {your-project-name}/production/out راست کلیک کنید و دکمه Copy Path/Reference... رو بزنین:



همچنین صفحه‌ای براتون باز می‌شه. Absolute Path رو بزنید.



حالا که مسیر خروجی بیلد رو کپی کردین، می‌تونین با استفاده از command-line برنامه‌تون رو اجرا کنید.

اگر Windows دارین، cmd و اگر Linux/macOS دارین، terminal رو باز کنید. اول از همه چک کنید که دستور زیر براتون کار کنه:

```
java --version
```

باید خروجی‌ای مشابه این ببینید:

```
java 23.0.1 2024-10-15
Java(TM) SE Runtime Environment (build 23.0.1+11-39)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.1+11-39, mixed mode, sharing)
```

اگر این‌طور نبود، یا راجع به مشکل سرچ کنید، یا نزدیک‌ترین تدریس‌یار رو خبر کنید تا با هم درستش کنیم. در غیر این صورت، با استفاده از دستور cd به دایرکتوری خروجی بیلدتون (که بالاتر کپی‌ش کردین) برین:

```
cd /Users/armaho/Projects/Learning/first-java-project/out/production/first-java-project
```

حواستون باشه که به جای مسیری که من به cd دادم، مسیری که خودتون دادین رو بذارین.

اگر همه چیز درست انجام شده باشه، می‌تونید با استفاده از دستور زیر برنامه‌تون رو اجرا کنید. به اون ThisIsAnArgument که نوشتیم، command-line argument می‌گن و یکی از راه‌های ورودی دادن به یه برنامه‌ست:

```
java Main ThisIsAnArgument
```

می‌بینید که خروجی‌تون به همچین شکلی هستش:

```
Given arg: ThisIsAnArgument
```

اگر خروجی‌تون درست باشه، شما موفق شدین که یک argument رو از command-line بخونید، و اون رو خروجی بدین. ممکن بود بیش از یک argument داشته باشین:

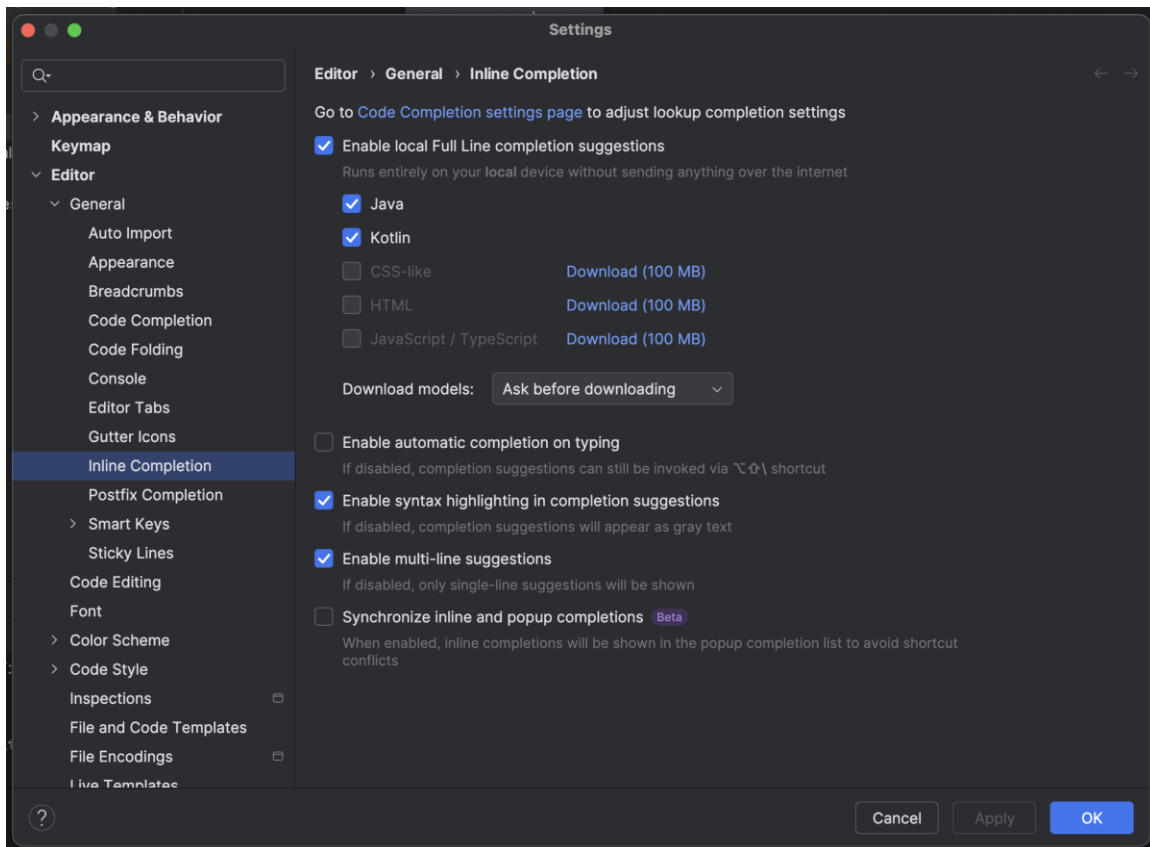
```
java Main arg1 arg2 arg3
```

جاوا، تمام این ورودی‌ها رو از طریق آرایهٔ `args` که ورودی متد `main` هست در اختیارتون می‌ذاره.

غیرفعال کردن Inline Completion در IntelliJ

حالا که با زبان جاوا آشنا شدین، حتما کدهای جلسات ابتدایی کارگاه و تمرین‌هاتون رو بدون استفاده از Inline Completion بزنین تا خود IntelliJ کدهاتون رو کامل نکنه. این کار، باعث می‌شه که مبانی زبان جاوا رو بهتر یاد بگیرید. نگران نباشین! یکی-دو جلسه دیگه دوباره اون رو روشن می‌کنید، چون برنامه‌نویس‌ها هر روز ازش استفاده می‌کنن تا سریع‌تر کد بزنین.

برای خاموش کردن اون، به تنظیمات IntelliJ برین. بعدش به مسیر `Editor > General > Inline Completion` برین تا صفحه زیر رو ببینین:



تیک `Enable automatic completion on typing` رو بردارین و دکمه `Apply` رو بزنین.

چه چیزی یاد گرفتیم؟

توی این داکيومنت، شما فهمیدین که:

- چگونه پروژۀ جدید جاوا تعریف کنیم و اجراش کنیم.
- چجوری ورودی بگیریم و خروجی بدیم.
- انواع متغیرهای پایه توی جاوا چی هستن و هر کدوم چجوری تعریف و استفاده می‌شن.
- حلقه‌ها توی جاوا چطوری کار می‌کنن.
- چجوری می‌تونیم با استفاده از متدها، کدتون رو به تکه‌های کوچک‌تر تقسیم کنیم.
- بلوک‌ها چی هستن.

اگر توی هر کدوم از این مفاهیم مشکلی دارین، حتماً با نزدیک‌ترین تدریس‌یار مطرحش کنیم. حتی اگر سوالی به نظرتون خیلی واضح می‌رسه، می‌تونیم از تدریس‌یارها بخوایم که جوابش رو بهتون بدن و لازم نیست هوشمندانه بودن سوالتون رو معیاری برای پرسیدن یا نپرسیدنش بدونید.