به نام خدا

دستور کار کارگاه برنامه‌نویسی پیشرفته

جلسه دوم

مفاهیم کلاس، شئ، مستندسازی در جاوا و تکمیل خودکار

مقدمه

در این جلسه قرار است تا با مفاهیم کلاس و شئ، تفاوت این دو و نحوه برنامه نویسی شئ‌گرا با جاوا آشنا شویم. در برنامه‌نویسی شئ‌گرا، مفاهیم همانند زندگی واقعی هستند. یک شئ که می‌تواند هر چیزی باشد (مثلا یک دانشجو)، تعدادی ویژگی مخصوص به خود دارد (مثلا یک دانشجو، با ویژگی‌های نام، نام خانوادگی، شماره دانشجویی و موجودی حساب کاربری مشخص می‌شود). همچنین یک شئ می‌تواند عملیاتی انجام دهد (مثلا یک دانشجو می‌تواند در یک کارگاه ثبت‌نام کند، یا موجودی حساب خود را افزایش دهد). همان‌طور که می‌بینید، دانشجوهای مختلف ویژگی‌های مختلفی دارند و در آنِ واحد می‌توانند عملیات مختلفی مستقل از یکدیگر انجام دهند؛ ولی همه آنها در ویژگی‌ها و عملیاتی که می‌توانند انجام دهند، مشترکند. یک کلاس در برنامه‌نویسی شئ‌گرا، ویژگی‌ها و عملیات کلی مشترک بین شئ‌ها را تعریف می‌کند. برای اینکه بهتر متوجه این موضوع و تفاوت‌های آن شوید، تمرین‌های این جلسه را به دقت انجام دهید.

نکات آموزشی

در برنامه‌نویسی شئ‌گرا همانند برنامه‌نویسی ساخت‌یافته از مفاهیم متغیرها، تابع‌ها (که در اینجا متد[1] گفته می‌شوند)، دستورات شرطی و حلقه‌ها، انتساب‌ها و عملگرهای ریاضی و منطقی استفاده می‌شود. این مفاهیم را در درس مبانی برنامه‌نویسی یاد گرفته‌ایم و در دستور کار اول نیز یادآوری شده بود.

برای اینکه کدهایمان خوانا باشند، به نحوه تعریف کلاس‌ها، شئ‌ها، متدها، متغیرها، چینش متدها و متغیرها در یک کلاس و نحوه کامنت‌گذاری دقت کنید. تمام این موارد را با مثال خواهیم دید.

مراحل انجام کار

به صورت مرحله به مرحله برنامه‌ای می‌نویسیم که کارگاه برنامه‌نویسی پیشرفته را شبیه‌سازی کند. همان‌طور که می‌دانید یک کارگاه از تعدادی دانشجو تشکیل شده است.

[1] Method

انجام دهید: با استفاده از کدهای زیر کلاس دانشجو را تعریف کنید:

```
/**
 * The Student class represents a student in a student
administration system.
 * It holds the student details relevant in our context.
 *
 * @author Ehsan
 * @version 0.0
 */
public class Student {
    fields

    constructors

    methods
}
```

در تعریف هر کلاس دو قسمت وجود دارد:

۱) Header: اطلاعات مربوط به کلاسی که تعریف می‌کنید را به شکل کامنت در بیرون از کلاس و در ابتدای فایل قرار دهید. این اطلاعات به شما کمک می‌کند تا برنامه‌ای خوانا داشته باشید و توسعه‌دهندگان دیگر نیز با خواندن این توضیحات، دیدی کلی از کلاسی که نوشتید به دست بیاورند. برای کامنت‌گذاری مانند زبان C، از /* */ برای چند خط و // برای یک خط استفاده می‌کنیم. در صورتی که پیش از تعریف هر کلاس یا هر متد توضیحات مورد نظر را در بین /** و */ قرار دهید، اطلاعاتی را برای JavaDoc مربوط به یک کلاس تهیه کرده‌اید که می‌توانید توضیحات را در قالب یک فایل مرتب html مستندسازی کنید. در این جلسه به طور مفصل درباره JavaDoc صحبت خواهیم کرد. تعدادی از کلمات کلیدی که با @ شروع شده‌اند، برای تولیدکردن JavaDoc کاربرد دارد. مثلا عبارت @author Ehsan، در مستندسازی که به صورت خودکار تولید خواهد شد، نویسنده این کلاس را Ehsan ثبت می‌کند.

۲)تعریف کلاس: برای تعریف کلاس از کلمه کلیدی class استفاده می‌کنیم. کلمه سمت چپ آن یعنی public سطح دسترسی کلاس را مشخص می‌کند، فعلا کلاس‌هایمان را public تعریف می‌کنیم؛ به این معنا که همه کلاس‌های دیگر قادر هستند از آن استفاده کنند. کلمه سمت راست، اسم کلاس است. اسامی کلاس را به شکل CamelCase[۲] با حرف اول بزرگ تعریف می‌کنیم. محدوده

[۲] یعنی کلمات به هم چسبیده هستند و حرف اول در کلمات میانی بزرگ نوشته می‌شوند. مثلا goodStudent.

کلاس هم با {} مشخص می‌شود. همان طور که می‌بینید بعد از فیلدها، constructor و متدها به ترتیب تعریف می‌شوند.

۱. فیلدها: ویژگی‌های مشترک بین دانشجوها را در این‌جا تعریف می‌کنیم. مثل نام، نام خانوادگی، شماره دانشجویی و نمره. فیلدها را این طور تعریف می‌کنیم: اول سطح دسترسی آن را با private مشخص می‌کنیم، در این صورت این متغیر فقط در محدوده کلاس Student قابل دسترسی است. بعد از آن نوع (type) آن مثلا int و در نهایت اسم آن را مشخص می‌کنیم. نام را به شکل camelCase با حرف اول کوچک تعریف می‌کنیم. برای

```
public class Student {

    // the student's first name
    private String firstName;

    // the student's last name
    private String lastName;

    // the student ID
    private String id;

    //the grade
    private int grade;

}
```

خوانایی بیشتر کد همه فیلدها را همراه با کامنت توضیحات آن فیلد تعریف می‌کنیم.

۲. Constructor: اولین رویه‌ای[۳] است که بعد از ساختن یک شئ فراخوانی می‌شود و فیلدهای لازم را مقداردهی می‌کند و عملیات تعیین‌شده لازم دیگر را انجام می‌دهد.

با استفاده از @param در کامنت قبل از constructor، شرح تک تک پارامترهای ورودی به آن را مشخص می‌کنیم. constructor را با سطح دسترسی public و بدون نوع خروجی و همنام با نام کلاس تعریف می‌کنیم. در ابتدای تعریف هر شئ از جنس Student، نام، نام خانوادگی و شماره دانشجویی دریافت می‌شود و فیلد مربوطه به هرکدام مقداردهی می‌شود. همچنین در این مثال، مقدار نمره به طور پیش‌فرض برای هی شئ از این جنس، صفر قرار می‌گیرد.

[۳] Procedure

متدها: عملیاتی که شئ‌های از جنس Student قادر به انجام آن هستند در این قسمت تعریف

```
/**
 * Create a new student with a given name and ID number.
 *
 * @param fName first name of student
 * @param lname last name of student
 * @param sID student ID
 */
public Student(String fName, String lname, String sID){
        firstName = fName;
        lastName = lname;
        id = sID;
        grade = 0;
}
```

می‌شوند.

برای تعریف هر متد مانند تعریف تابع در C عمل می‌کنیم؛ با این تفاوت که در تعریف آن ابتدا سطح دسترسی آن متد را مشخص می‌کنیم. مثلا public. همچنین برای متدهایی که خروجی دارند با @return اطلاعات مربوط به خروجی متد را برای تولید مستندات ثبت می‌کنیم.

اگر نیاز است که از بیرون از کلاس به آن فیلدها دسترسی داشته باشیم، متدهای setter و getter را برای آنها تعریف می‌کنیم. مزیت این روش در آن است که اگر بخواهیم در مقدار دهی به یک فیلد محدودیتی ایجاد کنیم، می‌توانیم آن محدودیت را در متد set مربوط به آن پیاده‌سازی کنیم. مثلا می‌خواهیم شماره دانشجویی یک رشته ۷ رقمی باشد نه بیشتر یا کمتر. اگر فقط فیلد مربوط را public تعریف کنیم، یک شئ دیگر از بیرون می‌تواند به هر شکلی به آن مقداردهی کند که مورد نظر ما نیست!

انجام دهید: باتوجه به توضیحات قسمت قبل کلاس Student را کامل کنید.

```
/**
 * get the first name of student
 * @return firstName field
 */
public String getFirstName() {
        return firstName;
}

/**
 * @param firstName set first name of a student
 */
public void setFirstName(String fName) {
        firstName = fName;
}

/**
 * Print the student’s last name and ID number to the
output terminal.
 */
public void print() {
        System.out.println(lastName + ", student ID: "
                        + id + ", grade: " + grade);
}
```

سپس کد زیر را در یک کلاس جدید تعریف کنید. پروژه خود را اجرا کنید و خروجی خود را تحلیل کرده به مدرس کارگاه ارائه دهید.

```java
public class Run {
    public static void main(String[] args) {
        Student std1 = new Student("Ehsan","Edalat", "9031066");
        Student std2 = new Student("Seyed", "Ahmadpanah", "9031806");
        Student std3 = new Student("Ahmad", "Asadi", "9031054");

        std1.print();
        std1.setGrade(15);
        std1.print();

        std2.print();
        std2.setGrade(11);
        std2.print();

        std3.print();
        std3.setFirstName("HamidReza");
        std3.print();
    }
}
```

همان‌طور که می‌بینید برای تعریف یک شئ (مانند خط سوم) این طور عمل می‌کنیم:

۱. ابتدا نوع شئ (نام کلاس) را تعیین می‌کنیم مثلا Student
۲. یک نام برای شئ انتخاب می‌کنیم. نام باید بامفهوم باشد و مرتبط با کلاسی باشد که از آن شئ را می‌سازیم. نام را به شکل camelCase با حرف اول کوچک می‌نویسیم.
۳. با استفاده از کلمه کلیدی new و به دنبال آن نام کلاس شئ ساخته می‌شود.
۴. درون پرانتز باید پارامترهایی که برای Constructor آن کلاس تعریف کرده‌ایم را مقداردهی کنیم.

انجام دهید: کلاس Lab را مانند کد زیر پیاده‌سازی کنید. کد زیر پیاده‌سازی متدها و constructor و کامنت‌های مرتبط با آنها را در خود ندارد. این موارد باید پیاده‌سازی شوند. در کلاس Run که پیش از این پیاده‌سازی کرده بودید، یک شئ از جنس Lab بسازید و تعدادی دانشجو به آن enroll کرده و در نهایت متد print را فراخوانی کنید. متد print باید شامل اطلاعات دانشجوهای Lab و میانگین نمره‌های آن‌ها باشد.

توجه! همان طور که می‌بینید در متد enrollStudent پارامتر ورودی از نوع Student است. یک متد از یک کلاس می‌تواند شئ ای از یک کلاس دیگر را به عنوان پارامتر ورودی دریافت کند.

فکر کنید: در مورد نحوه ارسال یک شئ به یک متد فکر کنید. این ارسال از جنس call-by-value است یا از جنس call-by-reference؟ با یک مثال درستی حدس خود را نشان دهید و به مدرس

```
public class Lab {
    private Student[] students;
    private int avg;
    private String day;
    private int capacity;
    private int currentSize;
    public Lab(int cap, String d) {}

    public void enrollStudent(Student std) {
        if (currentSize < capacity) {
            students[currentSize] = std;
            currentSize++;
        } else {
            System.out.println("Lab is full!!!");
        }
    }
    public void print() {}
    public Student[] getStudents() {}
    public void setStudents(Student[] students) {}
    public int getAvg() {}
    public void calculateAvg() {}
    public String getDay() {}
    public void setDay(String day) {}
    public int getCapacity() {}
    public void setCapacity(int capacity) {}
}
```

کارگاه نتیجه را گزارش دهید.

اشکال‌زدایی

1. دانشجویی متد print را به شکل زیر پیاده‌سازی کرده است. اشکال کد در کجاست؟

```
public void print() {
    for (int i = 0; i < students.size(); i++) {
        System.out.println("std fname: " + students[i].getFirstName()
                           + " std id:" + students[i].getId()
                           + " std grade:"+ students[i].getGrade());
    }
        System.out.println("Lab AVG:" + avg);
}
```

2. با هماهنگی با مدرس کارگاه، کد پیاده‌سازی شده یکی دیگر از افراد کلاس را با توجه به نکات زیر ارزیابی کنید:

- کامنت‌گذاری مناسب و به‌جا برای فیلدها، متدها و constructorها و رعایت نکات مربوط به JavaDoc
- پیاده‌سازی درست کلاس‌ها و متدهای هر یک و آزمون درستی عملکرد آنها
- نحوه درست نام‌گذاری برای کلاس‌ها، متدها و فیلدها

انجام دهید:

می‌خواهیم شبیه سازی کلاس کارگاه را به دانشکده تعمیم دهیم. جزئیاتی که در مورد این شبیه‌سازی در دانشکده ما وجود دارد را ابتدا بر روی کاغذ بیاورید و در مورد آن با مدرس کارگاه مشورت کنید. سپس کلاس‌هایی که به آنها رسیده‌اید را پیاده‌سازی کنید. برای بررسی درستی کدهای خود کلاس Run مناسب پیاده‌سازی کنید. در انتها کد خود را به مدرس کارگاه ارائه دهید.

آشنایی با مستندات جاوا

زبان برنامه‌نویسی جاوا معمولا برای پروژه‌هایی با ابعاد بزرگ استفاده می‌شود که به وسیله‌ی تیم‌های برنامه‌نویسی توسعه می‌یابند. از این رو لازم است روشی برای انتقال اطلاعات و نحوه‌ی استفاده از کلاس‌ها و متدهای نوشته‌شده توسط هر برنامه‌نویس به دیگران وجود داشته باشد. این عمل توسط مستندسازی کدها انجام می‌شود. از طرفی، هنگامی که برنامه‌نویسان برنامه‌های خود را به صورت کتابخانه در اختیار دیگران قرار می‌دهند، لازم است چگونگی فراخوانی توابع و متدهای استفاده‌شده در آن برای

استفاده‌کنندگان به نحوی مشخص شود که بدون نیاز به اطلاع از جزئیات و نحوه پیاده‌سازی، بتوان به سادگی از آن‌ها در کاربردهای مختلف استفاده کرد. یکی از اهداف دستور این جلسه آشنایی با روش استفاده از این مستندات و تولید آن‌ها برای برنامه‌هایی است که در این درس پیاده‌سازی می‌شوند.

کتابخانه‌های جاوا همراه با یک فایل مستند ارائه می‌شوند که در آن روش استفاده از کلاس‌های موجود در کتابخانه، توضیح واسط (interface)های موجود، روش فراخوانی متدها، ورودی و خروجی هر متد و شرح کلی عملکرد مربوط به آن توضیح داده شده است. این مستندات برای کتابخانه‌های معروف جاوا در اینترنت موجود است و در سایت‌هایی مانند: https://www.oracle.com و https://www.tutorialspoint.com/java/ [۴] یافت می‌شود (نسخه‌ای از مستندات رسمی موجود در سایت اوراکل از http://ceit.aut.ac.ir/~ghaffarian/files/jdk-۸u۱۶۱-docs-all.zip قابل دانلود است.

یکی از مهم‌ترین ابزارهای نگارش مستند در جاوا، JavaDoc است. این ابزار که در JDK موجود است، برای ساخت مستند کاربرد دارد. روش استفاده از این ابزار به این صورت است که ابتدا در کد خود با استفاده از یک دستور زبان [۵] خاص توضیحات را وارد کرده، سپس با اجرای JavaDoc مستندات را در قالب یک فایل html تولید می‌کنید. این دستورات به صورت کامنت لابلای کد نوشته می‌شوند و توسط کامپایلر بررسی نمی‌شوند. نوع سوم از کامنت‌گذاری که در جدول ۱ آماده است، برای نوشتن این دستورات به کار می‌رود.

جدول ۱ - انواع کامنت گذاری در جاوا

Sr.No.	Comment & Description
۱	**/* text */** The compiler ignores everything from /* to */.
۲	**//text**

[۴] منبع رسمی زبان برنامه نویسی جاوا این سایت است ولی به دلیل تحریم‌ها فعلا امکان دسترسی به آن‌ها از آدرس‌های ایران نیست.

[۵] Grammar

	The compiler ignores everything from // to the end of the line.
۳	**/** documentation */** This is a documentation comment and in general it's called **doc comment**. The **JDK javadoc** tool uses *doc comments* when preparing automatically generated documentation.

این دستورات قبل از قطعه کدی که قصد توضیح آن را داریم (مثال: قبل از تعریف کلاس، قبل از تعریف متدها و فیلدها) نوشته می‌شود. برای نمونه، اگر قصد توضیح عملکرد یک کلاس خاص را داریم، در بالای کد (مانند کد ۱) توضیحات را می‌نویسیم.

```java
/**
 * The HelloWorld program implements an application that
 * simply displays "Hello World!" to the standard output.
 *
 * @author  Sepehr Sabour
 * @version 1.0
 * @since   2018-01-15
 */
public class HelloWorld {

  public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

کد ۱- نمونه یک متد مستندسازی شده

همچنین می توانید با استفاده از تگ‌های مخصوص زبان html نیز توضیحات خود را تکمیل کنید. در این صورت می‌توانید قالب فایل تولیدشده را نیز بهبود دهید. نمونه‌ای از یک فایل مستندات مربوط به کلاس Scanner را در فایل پیوست‌شده مشاهده کنید.

تگ‌ها در مستندات برای بیان آن‌چه از پیش تعریف شده است، استفاده می‌شوند و روش یکسانی را برای معرفی این موارد فراهم می‌آورند. برای مثال، تگ @param جهت معرفی پارامترهای یک متد است که اصولا بیان آن در مستندسازی هر متد الزامی است. در جدول ۲ تگ‌های شناخته‌شده برای تولید مستند همراه با توضیحات و کاربرد هر کدام آورده شده است.

جدول ۲-تگ‌های استفاده شده در JavaDoc

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.	{@code text}
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@inheritDoc}	Inherits a comment from the **nearest**inheritable class or implementable interface.	Inherits a comment from the immediate surperclass.
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
{@linkplain}	Identical to {@link}, except the link's label is displayed in plain text than code font.	{@linkplain package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.	@param parameter-name description
@return	Adds a "Returns" section with the description text.	@return description

@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField component.	@serialField field-name field-type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
{@value}	When {@value} is used in the doc comment of a static field, it displays the value of that constant.	{@value package.class#field}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

پس از نوشتن مستندات و گذاشتن تگ‌های مورد نظر، با استفاده از دستور javadoc مستند مربوطه را تولید می‌کنیم.

```
> javadoc  <MySourceFileName> .java -d <Destination Directory>

> javadoc -sourcepath  <Source Directory>  -d  <Destination Directory>
```

روش ساده‌تر استفاده از IDE برای تولید JavaDoc است. در IntelliJ IDEA، از منوی Tools، با انتخاب گزینه Generate JavaDoc می‌توان به سادگی فایل html مستندات پروژه را ایجاد کرد.

نمونه‌ای از مستند تولیدشده برای کد را در پیوست دستور کار قابل مشاهده است.

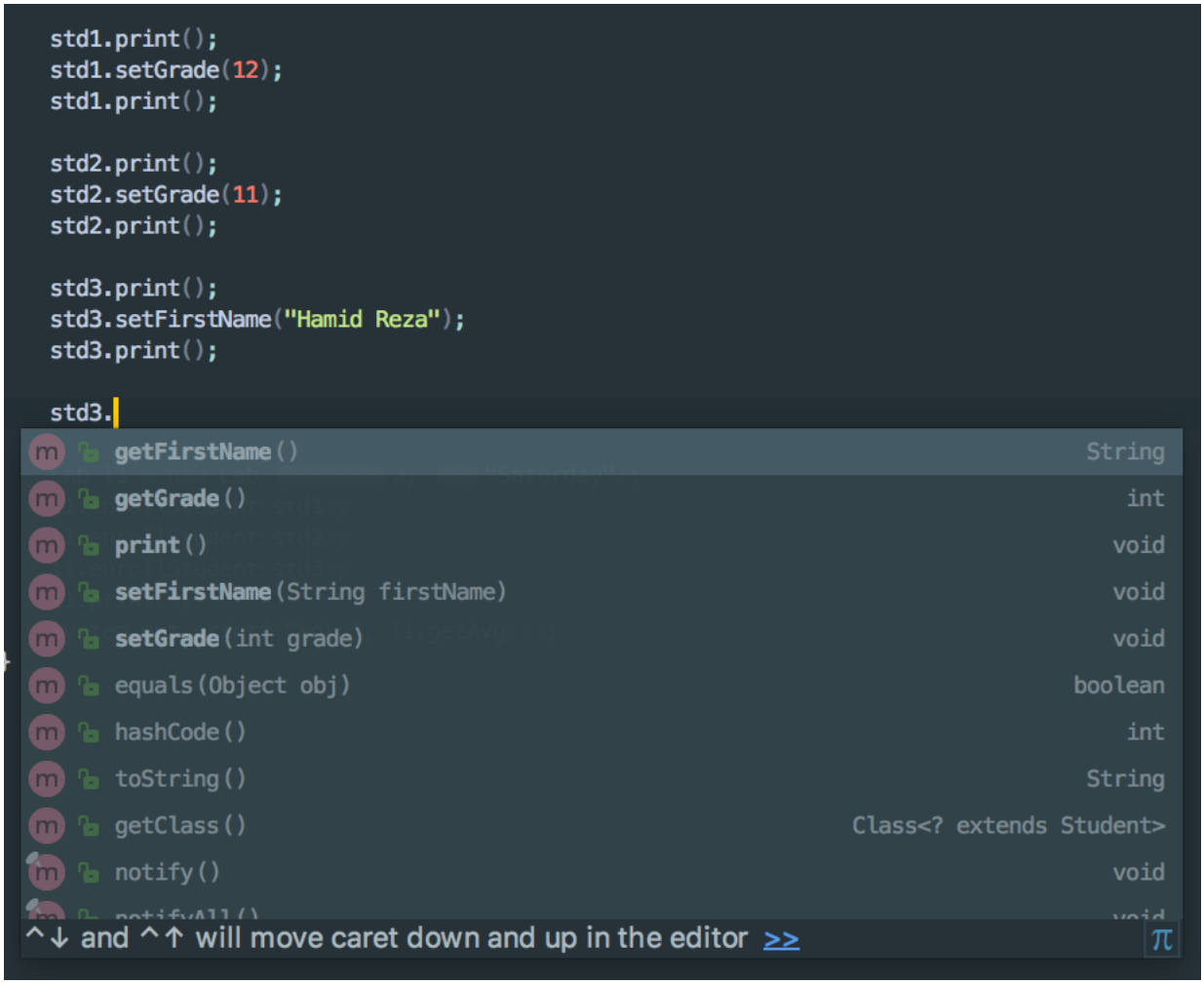
```java
import java.io.*;

/**
 * <h1>Add Two Numbers!</h1>
 * The AddNum program implements an application that
 * simply adds two given integer numbers and Prints
 * the output on the screen.
 * <p>
 * <b>Note:</b> Giving proper comments in your program makes it more
 * user friendly and it is assumed as a high quality code.
 *
 * @author  Sabour Sepehr
 * @version 1.0
 * @since   2018-01-17
 */
public class AddNum {
  /**
   * This method is used to add two integers. This is
   * a the simplest form of a class method, just to
   * show the usage of various javadoc Tags.
   * @param numA This is the first paramter to addNum method
   * @param numB  This is the second parameter to addNum method
   * @return int This returns sum of numA and numB.
   */
  public int addNum(int numA, int numB) {
    return numA + numB;
  }

  /**
   * This is the main method which makes use of addNum method.
   * @param args Unused.
   * @exception IOException On input error.
   * @see IOException
   */

  public static void main(String args[]) throws IOException {
    AddNum obj = new AddNum();
    int sum = obj.addNum(10, 20);

    System.out.println("Sum of 10 and 20 is :" + sum);
  }
}
```

کد ۱ - نمونه یک کلاس مستندسازی شده

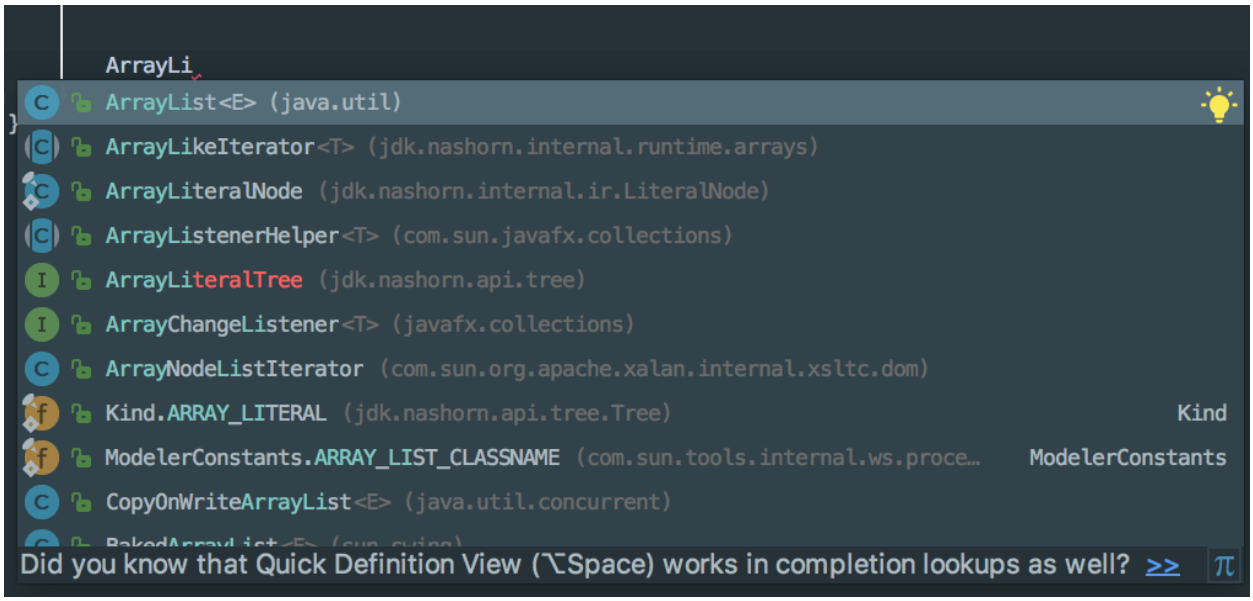
انجام دهید: برای تمرین بیشتر روش مستندسازی در جاوا، تمرین این جلسه را توسط روش گفته شده مستندسازی کنید. لازم به یادآوری است که مستندسازی تمرین‌ها و پروژه‌های این درس بعد از این جلسه الزامی است.

آشنایی با تکمیل خودکار در محیط توسعه یکپارچه

یکی از ویژگی‌هایی که IDEهای معروف مانند IntelliJ در اختیار توسعه‌دهندگان قرار می‌دهد، تکمیل خودکار کد است؛ به این ترتیب که شما بخشی از متد، فیلد، نوع و موارد دیگری را که می‌خواهید، نوشته و با فشردن دکمه‌های Ctrl + Space، IDE پیشنهادهایی برای تکمیل آن به شما ارائه می‌کند (پیشتر shortcutهای کاربردی محیط IntelliJ در مودل قرار گرفته است).



شکل ۱ – پیشنهادهای ارائه‌شده توسط IDE برای تکمیل کد



شکل ۲- پیشنهادهای ارائه‌شده توسط IDE برای تکمیل کد

با استفاده از این ویژگی، شما می‌توانید اطلاعاتی از کتابخانه‌هایی که برای شما آشنا نیستند، به دست آورید و مدت زمان نوشتن کد را کاهش دهید. در صورتی که شما نمی‌دانید چگونه می‌توانید با HashMap کار کنید، اگر یک instance از آن بسازید، با فشردن Ctrl + Space می‌توانید متدهایی که بیشتر کاربرد دارند را مشاهده کنید و به این ترتیب از متد مورد نظر استفاده کنید.

شکل ۳ - پیشنهادهای ارائه‌شده توسط IDE برای تکمیل کد

انجام دهید

با استفاده از مستند گیت موجود در مودل، پروژه این جلسه را در گیت قرار دهید.

لازم به ذکر است، از این جلسه به بعد، قرار دادن تمرین‌ها و پروژه‌ها در گیت **اجباری** است. شما باید به استاد کارگاه خود دسترسی Maintainer بدهید تا ایشان درستی فعالیت‌های شما را بررسی کند.

(**راهنمایی برای استاد:** تمام مستند گیت در این جلسه باید تدریس شود.)