کارگاه برنامه نویسی پیشرفته

دستور کار شماره سه

اهداف

- آشنایی با Object Composition
- آشنایی با رشته‌ها در جاوا
- یادگیری جاواداک[1]

[1] Javadoc

فهرست مطالب

آشنایی با Object Composition

در برخی موارد برنامه‌نویسی پیش می‌آید که رابطه‌ی بین دو کلاس به قدری زیاد است که باید یک شئ از یک کلاس در کلاس دیگر به عنوان فیلد وجود داشته باشد. در واقع object composition زمانی به وجود می‌آید که کلاس A به کلاس B برای کارایی مطلوب نیازمند بوده و کلاس B بدون کلاس A وجود و معنی نداشته باشد. مانند قلب در بدن انسان.

مزایای استفاده از object composition

۱. امکان استفاده‌ی مجدد از کدهای قبلی
۲. جبران عدم امکان ارث‌بری[1] از چند کلاس در جاوا
۳. امکان دیباگینگ[2] و تست راحت‌تر برنامه
۴. امکان ایجاد راحت‌تر تغییر در کارایی یک کلاس
۵. عدم وجود محدودیت در نام‌گذاری متدها[3] در مقایسه با ارث‌بری

در مثال زیر کلاس University دارای چندین College می‌باشد. همانند توضیحات، دانشکده‌ی بدون دانشگاه معنا ندارد:

```
import java.util.*;

// class College
class College {
    public String name;
    public int capacity;

    College(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
    }
```

[1] inheritance
[2] debugging
[3] method

```
    public String getName() {
        return name;
    }

    public int getCapacity() {
        return capacity;
    }
}

// University has more than one college.
class University {
    // reference to refer to list of college.
    private final List<College> colleges;

    University(List<College> colleges) {
        this.colleges = colleges;
    }

    // Getting total number of colleges
    public List<College> getTotalCollegesInUniversity() {
        return colleges;
    }
}

class CompositionExample {
    public static void main(String[] args) {
        // Creating the Objects of College class.
        College c1
                = new College("Computer Engineering College", 120);
        College c2
                = new College("Mechanic Engineering College", 150);
        College c3 = new College("Electronics Engineering College", 200);

        // Creating list which contains the no. of colleges.
        List<College> college = new ArrayList<College>();

        college.add(c1);
        college.add(c2);
        college.add(c3);

        University university = new University(college);
        List<College> colleges = university.getTotalCollegesInUniversity();

        for (College cg : colleges)
            System.out.println("Name : " + cg.name + " and " + " Capacity : "
+ cg.getCapacity());
    }
}
```

(مثالی از object composition)

آشنایی با رشته‌ها در جاوا

در جاوا، رشته شئ‌ای می‌باشد که به دنباله‌ای از کاراکترها اشاره می‌کند که به دنبال هم قرار دارند و می‌توان با "" یا استفاده از سازنده[1] کلاس String رشته‌ی دلخواه را ساخت.

رشته‌ها داده‌های تغییر ناپذیر[2] هستند و هنگام تغییر آن‌ها رشته‌ی دیگری ساخته می‌شود و رشته‌ی قبلی از بین می‌رود و به متغیر ما رشته‌ی جدید ساخته شده نسبت داده می‌شود.

در جاوا برای کلاس String متدهای زیادی ساخته شده است که برای استفاده از آن‌ها نیازی به ایمپورت[3] کردن کتابخانه‌ای نمی‌باشد. (جزو کلاس‌های built-in است)

متدهای پرکاربرد کلاس استرینگ

- str.length()

این متد طول رشته داده شده را برمی‌گرداند.

- str.concat(String str2)

این متد رشته‌ی داخل پرانتز را به انتهای رشته‌ی اولیه متصل می‌کند و رشته‌ی نهایی را برمی‌گرداند. دقت کنید که این متد در رشته‌ی اولیه تغییری ایجاد نمی‌کند.

- str.equals(String str2)

این متد دو رشته را مقایسه می‌کند و اگر هر دو محتویات یکسانی داشته باشند true برمی‌گرداند در غیر این صورت مقدار false برمی‌گرداند.

[1] constructor
[2] immutable
[3] import

- str.indexOf(int chUnicode), str.indexOf(char c), str.indexOf(String str2)

این متد به دنبال ایندکس[1] مربوط به اولین مکان وقوع کاراکتر مربوط به یونیکد مورد نظر، کاراکتر داده شده یا رشته‌ی داخل پرانتز در رشته‌ی داده شده می‌گردد و در صورت یافتن آن، ایندکس مربوطه را برمی‌گرداند و اگر هم در رشته‌ی داده شده وجود نداشته باشد، ۱- برمی گرداند.

- str.toUpperCase(), str.toLowerCase()

این متدها تمام کاراکترهای رشته‌ی اولیه را به حروف بزرگ‌تر (toUpperCase) یا به حروف کوچک‌تر (toLowerCase) تبدیل می‌کنند و رشته‌ی نهایی را برمی‌گردانند. دقت کنید که این دو متد در رشته‌ی اولیه تغییری ایجاد نمی‌کنند.

- str.toCharArray()

این متد رشته را به آرایه‌ای از کاراکترها تبدیل می‌کند و آرایه‌ی نهایی را برمی‌گرداند.

- str.split(String str2, int limit)

این متد بر اساس ورودی داده شده آرایه‌ای از رشته‌ها را برمی‌گرداند که بین هر عضو آرایه در رشته‌ی اولیه رشته‌ی داخل پرانتز وجود دارد و با استفاده از پارامتر limit می‌توان تعداد اعضای آرایه را محدود کرد.

- str.valueOf(parameter)

این متد رشته‌ای شامل ورودی را برمی‌گرداند. (ورودی می‌تواند هر نوع داده‌ی primitive باشد)

- str.replace(char oldChar, char newChar)

این متد تمامی کاراکترهای قدیمی را با کاراکترهای جدید داده شده جایگزین می‌کند و رشته‌ی نهایی را برمی‌گرداند ولی در رشته‌ی اولیه تغییری ایجاد نمی‌کند.

- str.compareTo(String str2)

این متد دو رشته‌ی داده شده را مقایسه می‌کند و نتیجه را برمی‌گرداند. در صورت برابری دو رشته ۰ و در صورت بزرگ‌تر بودن رشته‌ی اولیه عددی مثبت و کوچک‌تر بودن رشته‌ی اولیه نیز عددی منفی برمی‌گرداند.

[1] index

- str.charAt(int index)

این متد کاراکتر موجود در ایندکس داده شده را برمی‌گرداند.

- str.subString(int beginIndex, int lastIndex)

این متد رشته‌ی با شروع از ایندکس[1] اولیه و ختم به ایندکس ما قبل دومی در ورودی‌ها را از رشته‌ی اولیه برمی‌گرداند و همچنین در صورت عدم استفاده از ایندکس دوم، تا انتهای رشته‌ی اولیه در رشته‌ی نهایی وجود خواهد داشت.

همچنین برای اطلاعات بیشتر می‌توانید به متدهای استرینگ جاوا مراجعه کنید.

نکته: دقت داشته باشید که متد equals() به محتویات دو رشته دقت می‌کند ولی عملگر == به محل دو رشته در حافظه دقت می‌کند، به عنوان مثال در کد زیر جواب false چاپ می‌شود:

```
public class Main {
    public static void main(String[] args) {
        String str1 = "Hello, World";
        String str2 = new String("Hello, World");
        System.out.println(str1 == str2);
    }
}
```

(روشی از ساخت رشته)

در حالی که اگر رشته‌ها را به صورت زیر بسازیم جواب true چاپ می‌شود:

```
public class Main {
    public static void main(String[] args) {
        String str1 = "Hello, World";
        String str2 = "Hello, World";
        System.out.println(str1 == str2);
    }
}
```

(روش دیگری از ساخت رشته)

در ادامه به علت این موضوع می‌پردازیم. اما اول باید با مفهومی آشنا شویم.

در جاوا برای استفاده‌ی بهینه‌تر از حافظه از مکانی در هیپ[2] به نام String Constant Pool استفاده می‌شود که هر رشته‌ای که در حین اجرای برنامه ساخته شود، رشته‌ی معادل آن در این مکان ذخیره می‌شود. اما علت جواب متفاوت دو مثال بالا تنها با این مطلب قابل فهم نیست، بلکه باید به نحوه‌ی تشکیل و محل اشاره کردن پوینتر رشته‌ها در جاوا دقت کرد.

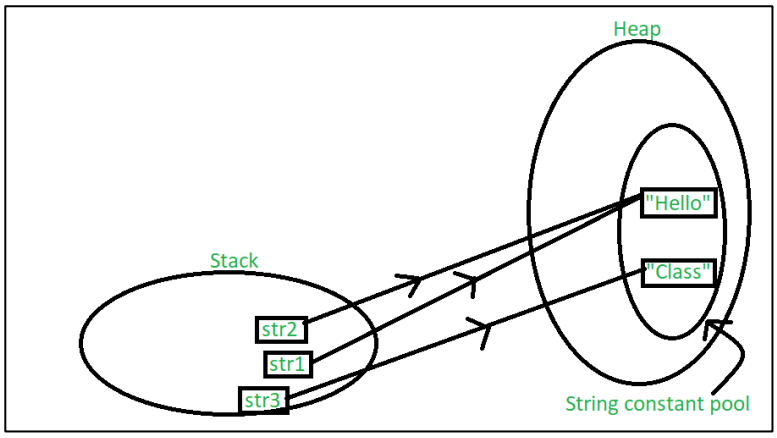
[1] index
[2] heap

بررسی نحوه ساختن رشته در جاوا

برای ساخت رشته در جاوا، چندین روش وجود دارد:

۱. استفاده از ""

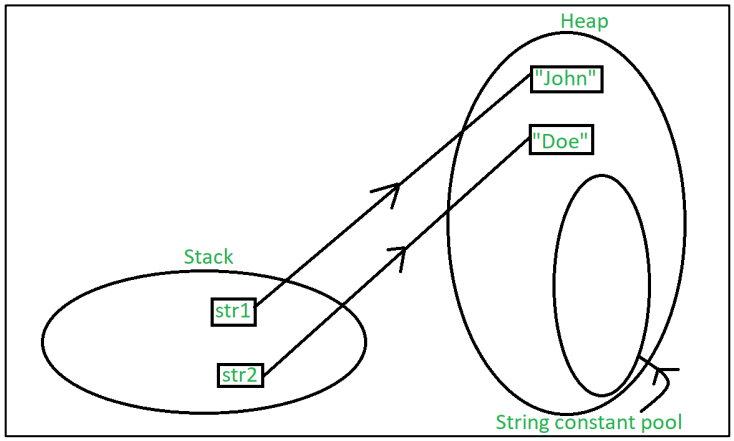
در صورت استفاده از این طریق متغیر رشته‌ی ساخته شده به مکانی در pool مربوط به رشته‌ها اشاره می‌کند که برای هر دو مقدار یکسان، یک خانه در نظر می‌گیرد. برای همین در مثال دوم بخش قبلی عملگر == که محل متغیرها را بررسی می‌کند مقدار true برمی‌گرداند:



(شماتیک ساخت رشته با روش اول)

۲. استفاده از سازنده‌ی کلاس استرینگ و کلمه‌ی new

در این حالت JVM محل مجزایی برای هر رشته در هیپ در نظر می‌گیرد و متغیر آن به این مکان اشاره می‌کند. همچنین مقدار رشته‌ی تشکیل شده در pool ذخیره نمی‌شود. در نتیجه در مثال اول عملگر == مقدار false را برمی‌گرداند:



(شماتیک ساخت رشته با روش دوم)

حال با توجه به مفاهیم گفته شده جواب کد زیر را حدس بزنید و حدس خود را در محیط برنامه‌نویسی خود امتحان کنید:

```
public class Main {
    public static void main(String[] args) {
        String str1 = new String("Hello, World");
        String str2 = new String("Hello, World");
        System.out.println(str1 == str2);
    }
}
```

(مثالی از ساخت رشته)

همچنین در جاوا کلاس‌هایی مشابه کلاس String هستند با این تفاوت که تغییرپذیر هستند. مانند StringBuilder و StringBuffer. به عنوان مثال در کد زیر عبارت Hello, World دو مرتبه چاپ می‌شود در حالی که در کلاس String رشته‌های اولیه تغییری نخواهند کرد:

```
public class Main {
    public static void main(String[] args) {
        StringBuffer str1 = new StringBuffer("Hello");
        StringBuffer str2 = new StringBuffer("Hello");
        str1.append(", World");
        str2.append(", World");
        System.out.println(str1);
        System.out.println(str2);
    }
}
```

(ساخت رشته با استفاده از StringBuffer)

جاواداک

چرا جاواداک؟

زبان جاوا اصولاً برای پروژه‌هایی با ابعاد بزرگ استفاده می‌شود که به وسیله‌ی تیم‌های برنامه‌نویسی توسعه می‌یابد. 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[1] انجام می‌شود.

از طرفی، هنگامی که برنامه‌نویسان برنامه‌های خود را به صورت کتابخانه در اختیار دیگران قرار می‌دهند، لازم است چگونگی فراخوانی متدهای استفاده‌شده در آن برای استفاده‌کنندگان به نحوی مشخص شود که بدون نیاز به اطلاع از جزئیات و نحوه‌ی پیاده‌سازی، بتوان به سادگی از آن‌ها در کاربردهای مختلف استفاده کرد. کتابخانه‌های جاوا همراه با یک فایل مستند ارائه می‌شوند که در آن روش استفاده از کلاس‌های موجود در کتابخانه، توضیح واسط‌های[2] 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 درسایت‌هایی مانند سایت‌های زیر یافت می‌شوند:

- https://www.oracle.com
- https://www.tutorialspoint.com/java

یکی از مهم‌ترین ابزارهای نگارش مستند در جاوا، جاواداک است. این ابزار که در JDK موجود است، برای ساخت مستند کاربرد دارد.

[1] documentation
[2] interface

استفاده از جاواداک

روش استفاده از این ابزار به این صورت است که ابتدا در کد خود با استفاده از یک دستور زبان خاص توضیحات را وارد کرده، سپس با اجرای جاواداک مستندات را در قالب یک فایل html تولید می‌کنید. این دستورات به صورت کامنت[1] لابلای کد نوشته می‌شوند و توسط کامپایلر[2] بررسی نمی‌شوند. نوع سوم از کامنت‌گذاری که در جدول زیر آماده است، برای نوشتن این دستورات به کار می‌رود:

Sr.No.	Comment & Description
Single-Line	// text The compiler ignores everything from "//" to the end of line.
Multi-line	/* A comment is written here */ The compiler ignores everything from "/*" to "*/"
Documentation	/** 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.

(انواع مختلف کامنت‌گذاری در جاواداک)

[1] comment
[2] compiler

تگ‌ها

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، @param جهت معرفی آرگومان‌های[1] یک متد است که اصولاً بیان آن در مستندسازی هر متد الزامیست. در جدول زیر تگ‌های شناخته شده برای تولید مستند همراه با توضیحات و کاربرد هر کدام آورده شده است:

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 superclass.
{@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

(تگ‌های مختلف جاواداک)

[1] arguments

@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

(ادامه‌ی تگ‌های جاواداک)

در زیر تکه‌کدی آمده است که مثالی از نوشتن جاواداک است:

```java
/**
* <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  ap-ce-aut
* @version 1.0
* @since   2021-08-24
*/

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 parameter 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 the addNum method.
     * @param args Unused.
     * @return Nothing.
     */

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

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

(تگ‌های مختلف جاواداک)

انجام دهید: پیاده سازی PhoneBook در جاوا

در این سوال قصد داریم پیاده‌سازی یک دفترچه‌ی تلفن را انجام دهیم.

توضیح کلاس‌ها به صورت زیر است:

- **Address**

این کلاس جهت ذخیره‌سازی آدرس مخاطبین استفاده می‌شود و دارای فیلدهای زیر است:

- zipCode

  کد پستی مخاطب را به صورت یک رشته ذخیره می‌کند (می‌تواند خالی باشد).

- country

  کشور مربوط به مخاطب را به صورت یک رشته ذخیره می‌کند (می‌تواند خالی باشد).

- city

هر مخاطب می‌تواند یک ایمیل داشته باشد (یا ایمیلی نداشته باشد) که به صورت یک رشته ذخیره می‌شود.

توضیحات متدهای این کلاس نیز به صورت زیر است:

- String toString()

  تمامی اطلاعات آدرس را با فرمت مناسب در یک رشته باز می‌گرداند.

- **PhoneNumber**

این کلاس به منظور ذخیره‌سازی شماره‌ی تلفن است. توضیحات فیلدهای آن به صورت زیر است:

- countryCode

  کد کشور شماره‌ی مربوطه است. برای مثال برای ایران، این مقدار برابر با «98+» است.

- number

شماره‌ی تلفن مربوطه است، که در یک رشته با **طول دقیقاً ۱۲** ذخیره می‌شود. برای مثال این مقدار می‌تواند برابر با «9121234567» باشد.

توضیحات متدهای این کلاس این به شرح زیر است:

- String toString()

تمامی اطلاعات شماره‌ی تلفن را با فرمت مناسب در یک رشته بر می‌گرداند.

- **Contact**

این کلاس درواقع یک شئ از مخاطب می‌باشد و شامل تمام اطلاعات مورد نیاز برای یک مخاطب است. توضیح فیلدهای آن به صورت زیر است:

- group

هر مخاطب می‌تواند عضو یک گروه خاص باشد (یا عضو هیچ گروهی نباشد) که به صورت یک رشته ذخیره می‌شود.

- email

هر مخاطب می‌تواند یک ایمیل داشته باشد (یا هیچ ایمیلی نداشته باشد) که به صورت یک رشته ذخیره می‌شود.

- firstName

یک رشته که نام مخاطب در آن ذخیره می‌شود.

- lastName

یک رشته که نام خانوادگی مخاطب در آن ذخیره می‌شود.

- phoneNumber

یک شئ از کلاس PhoneNumber که اطلاعات مربوط به شماره تلفن مخاطب را ذخیره می‌کند.

- address

یک شئ از کلاس Address که اطلاعات مربوط به آدرس مخاطب را ذخیره می‌کند.

- **PhoneBook**

در این کلاس تمامی مخاطبین ذخیره می‌شوند. توضیح فیلدهای آن به صورت زیر است:

- contacts

یک آرایه از تمامی مخاطبین است.

توضیح متدهای این کلاس نیز به صورت زیر است:

- boolean addContact(Contact contact)

یک مخاطب را به لیست اضافه می‌کند. اگر مخاطب در آرایه موجود بود، false برمی‌گرداند و کاری انجام نمی‌دهد، در غیر این‌صورت مخاطب را به آرایه اضافه می‌کند و true برمی‌گرداند.

- boolean deleteContact(String firstName, String lastName)

براساس نام داده شده به متد، مخاطب را پیدا می‌کند و آن را حذف می‌کند و true برمی‌گرداند. اگر مخاطب در آرایه موجود نبود، false برمی‌گرداند.

- Contact findContact(String firstName, String lastName)

براساس نام و نام خانوادگی داده شده به متد، مخاطب را پیدا می‌کند و آن را برمی‌گرداند. اگر مخاطب در آرایه موجود نبود، null برمی‌گرداند.

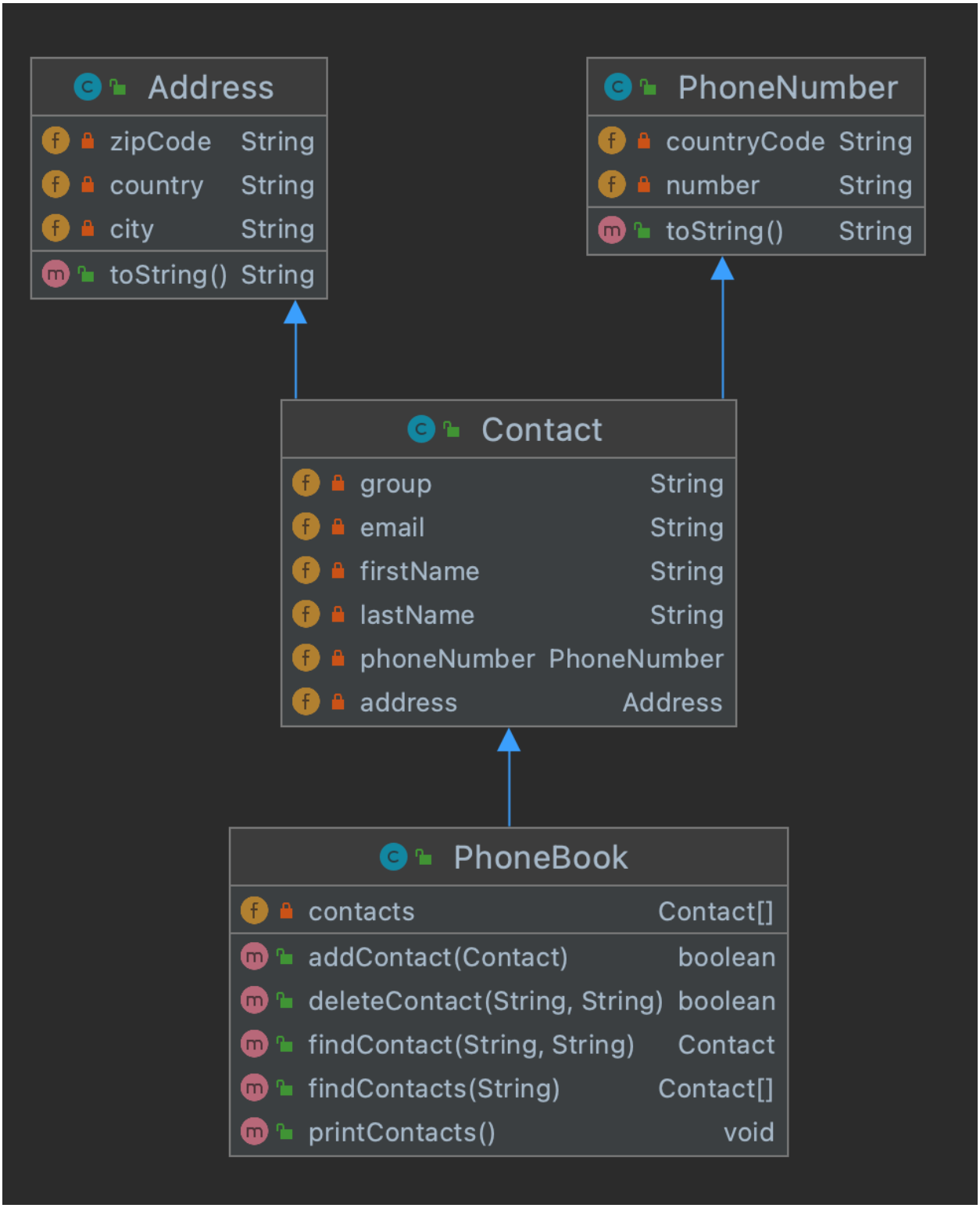
- Contacts[] findContacts(String group)

تمامی مخاطبین عضو گروه داده شده را تحت عنوان یک آرایه برمی‌گرداند. اگر هیچ مخاطبی با این گروه وجود نداشت، null برمی‌گرداند.

- void printContacts()

لیست مخاطبین را در قالبی مناسب نمایش می‌دهد.

در شکل زیر می‌توانید یک دیاگرام از تمام کلاس‌های مورد نیاز را ببینید: (پیاده‌سازی متدهای گفته شده الزامیست، طبیعی‌ست که در صورت نیاز می‌توانید متدهای دیگری نیز علاوه بر این متدها پیاده‌سازی کنید. درضمن هرجا که نیاز بود، متدهای getter و setter را نیز اضافه کنید)



(دیاگرام کلاس‌های پروژه)

- **Main**

در متد main این کلاس برنامه‌ای بنویسید که ورودی‌های زیر را بگیرد و خروجی مورد انتظار را تولید کند.

ورودی:

1- contacts -a <contact firstName> <contact lastName>

2- contacts -r <contact firstName> <contact lastName>

3- show -g <group name>

4- show -c <contact firstName> <contact lastName>

5- show

6- exit

۱. به‌منظور اضافه‌کردن مخاطب (در ادامه یک مثال از این فرایند خواهیم دید)

۲. به‌منظور حذف یک مخاطب (در صورتی که کاربر وجود داشت، Ok و در غیر این صورت Not found چاپ شود)

۳. به‌منظور نمایش دادن اطلاعات مخاطبان در یک گروه

۴. به‌منظور نمایش اطلاعات یک مخاطب

۵. نمایش نام تمام مخاطبان

۶. خروج از برنامه

خروجی:

دست شما برای تعیین فرمت خروجی برنامه باز است و می‌توانید هرگونه که خواستید خروجی‌ها را نمایش دهید.

صرفاً خروجی باید تمیز و خوانا باشد.

به مثال‌های خروجی زیر دقت کنید. مثال برای دستور show:

```
Contact {
    group: aut
    email: -
    firstName: Professor
    lastName: Bakhshi
    phoneNumber: (+98) 9123456789
    address: 1234567890 - IRN - Tehran
}
```

(نمونه‌ای از خروجی دستور show)

دقت کنید که در صورت خالی بودن هر یک از فیلدها، کاراکتر مناسبی چاپ کنید. (مانند مثال بالا، برای فیلد email)

مثال برای دستور اضافه کردن مخاطب:

```
Input: contacts -a Professor Bakhshi
Output: "Please enter contact's group: "
Input: AUT
Output: "Please enter contact's email: "
Input:
Output: "Please enter contact's country code: "
Input: +98
Output: "Please enter contact's phone number: "
Input: 9123456789
Output: "Please enter contact's zip code: "
Input: 1234567890
Output: "Please enter contact's country: "
Input: IRN
Output: "Please enter contact's city: "
Input: Tehran
Output: "Contact saved!"
```

(نمونه‌ای از خروجی دستور اضافه کردن مخاطب)

نحوه‌ی تحویل

قبل از پیاده‌سازی این تمرین، لازم است که مخزنی جدید در گیت‌هاب با نام AP-Workshop3 برای خودتان بسازید. دقت کنید که مخزنی که می‌سازید، حتماً از نوع private باشد که باقی افراد به آن دسترسی نداشته باشند.

برای انجام این تمرین، لازم است پس از پیاده‌سازی هر کلاس، تغییرات را در کامیتی جدا اعمال کرده و پس از تکمیل تمرین، همه‌ی کامیت‌ها را به مخزن گیت‌هابتان پوش کنید. توضیحات هر کامیت به صورت زیر است:

۱. ابتدا کلاس Address را کامل کنید و با پیغامی مناسب، تغییرات را کامیت کنید.

۲. بعد از آن، کلاس PhoneNumber را پیاده‌سازی کنید و با پیغامی مناسب، تغییرات را کامیت کنید.

۳. سپس کلاس Contact را پیاده‌سازی کنید و با پیغامی مناسب، تغییرات را کامیت کنید.

۴. پس از آن نیز کلاس PhoneBook را پیاده‌سازی کرده و با پیغامی مناسب، تغییرات را کامیت کنید.

۵. در آخر نیز کلاس Main را پیاده‌سازی کنید و تغییرات آن را کامیت کرده و در نهایت تمامی کامیت‌ها را به مخزن گیت‌هابتان پوش کنید.