کارگاه برنامه نویسی پیشرفته

دستور کار شماره یک

اهداف

- آشنایی با JDK، JRE و JVM
- اجرا کردن برنامه در کامندلاین[1]
- آشنایی با سینتکس[2]
- یادگیری شیوه‌ی کار با گیت[3] و گیت‌هاب[4]

[1] command line
[2] syntax
[3] git
[4] GitHub

فهرست مطالب

آشنایی با JDK، JRE و JVM

برای اجرای یک برنامه‌ی جاوا با کامندلاین، بهتر است با مفاهیم JDK، JRE و JVM آشنا شویم.

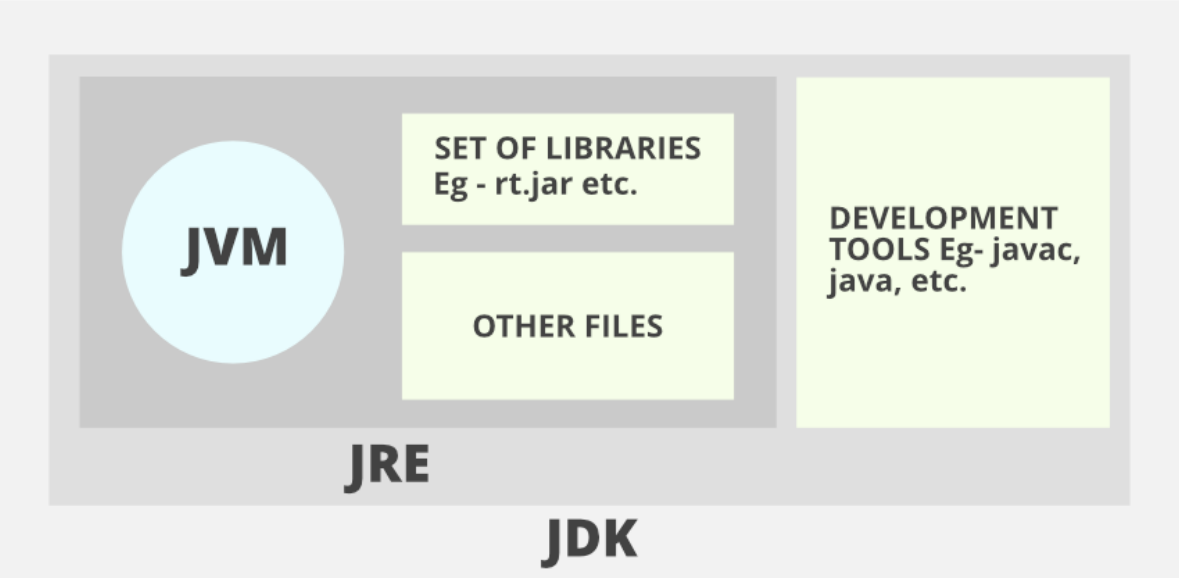
JDK (Java Development Kit): ابزاری است که امکان توسعه و اجرا کردن یک برنامه‌ی جاوا را فراهم می‌کند.

JRE (Java Runtime Environment): ابزاری است که تنها شرایط را برای اجرای یک برنامه‌ی جاوا فراهم می‌کند.

JVM (Java Virtual Machine): بخش مهمی در JRE و JDK است و در هر دو وجود دارد.

در اجرای هر برنامه‌ی جاوا توسط JDK و JRE، در اصل از JVM استفاده می‌شود. JVM مسئول اجرای یک برنامه‌ی جاوا به صورت خط به خط است.

تصویر زیر درک بهتری در مورد این سه ابزار به شما می‌دهد:



(شکل کلی JDK،JRE و JVM)

کامپایل و اجرای کد جاوا در ترمینال

یک ویرایشگر متن باز کنید و کد زیر را در آن بنویسید. سپس آن را با نام HelloWorld.java ذخیره کنید:

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

(اولین کد جاوا)

حال در پوشه‌ای که فایل کد را ذخیره کرده‌اید، CMD[1] یا ترمینال را باز کرده و دستور زیر را وارد کنید:

```
javac HelloWorld.java
```

با این کار یک فایل HelloWorld.class توسط کامپایلر[2] جاوا ایجاد می‌شود. (پایان مرحله‌ی کامپایل کد)

این فایل حاوی کدهایی است که توسط ماشین مجازی جاوا (JVM) شناخته می‌شود و می‌تواند آن‌ها را اجرا کند.

برای اجرای برنامه دستور زیر را وارد کنید:

```
java HelloWorld
```

با وارد کردن دستور بالا شاهد خروجی «Hello World» خواهیم بود.

بدین ترتیب می‌توان یک برنامه‌ی جاوا را در ترمینال اجرا کرد.

[1] Command Prompt
[2] compiler

بررسی برخی از نکات پایه سینتکس جاوا

مدیریت شرط‌ها و تصمیم‌گیری

برای مدیریت شرط‌ها و تصمیم گیری، روش‌های زیر در جاوا موجود است:

if statement:

```
if (condition) {

}
```

(نمونه‌ی if statement در جاوا)

if-else statement:

```
if (condition) {

}
else {

}
```

(نمونه‌ی if-else statement در جاوا)

if-else if:

```
if (condition1) {

}
else if (condition2) {

}
```

(نمونه‌ی if-else if statement در جاوا)

توجه کنید که جای شرط‌ها فقط boolean قرار دهید.

switch-case statement:

برای تصمیم‌گیری بر اساس مقدار یک متغیر می‌توان از switch-case استفاده کرد:

```
switch (i) {
    case 1:
        System.out.println("i = 1");
        break;
    case 2:
        System.out.println("i = 2");
        break;
    case 3:
        System.out.println("i = 3");
        break;
    default:
        System.out.println("i > 3");
}
```

(نمونه‌ی switch-case در جاوا)

ternary statement:

```
int ternary = i==10 ? 10 : 0;
```

(نمونه‌ی ternary statement در جاوا)

می‌توان گفت این روش کوتاه شده‌ی if-else statement است.

حلقه‌ها

for loop:

```
for (int i = 0; i < 10; i++) {

}
```

(نمونه‌ی for loop در جاوا)

for-each loop:

برای مثال می‌خواهیم تمام اعضای یک آرایه را چاپ کنیم:

```
int[] arr = new int[] {
        1, 2, 3, 4, 5
};

for (int i : arr) {
    System.out.println(i);
}
```

(نمونه‌ی for-each loop در جاوا)

در آینده خواهیم دید این کار را می‌توان روی لیست‌ها، ست‌ها و... نیز انجام داد.

while-loop:

```
while (condition) {

}
```

(نمونه‌ی while-loop در جاوا)

do-while loop:

```
do {

} while (condition)
```

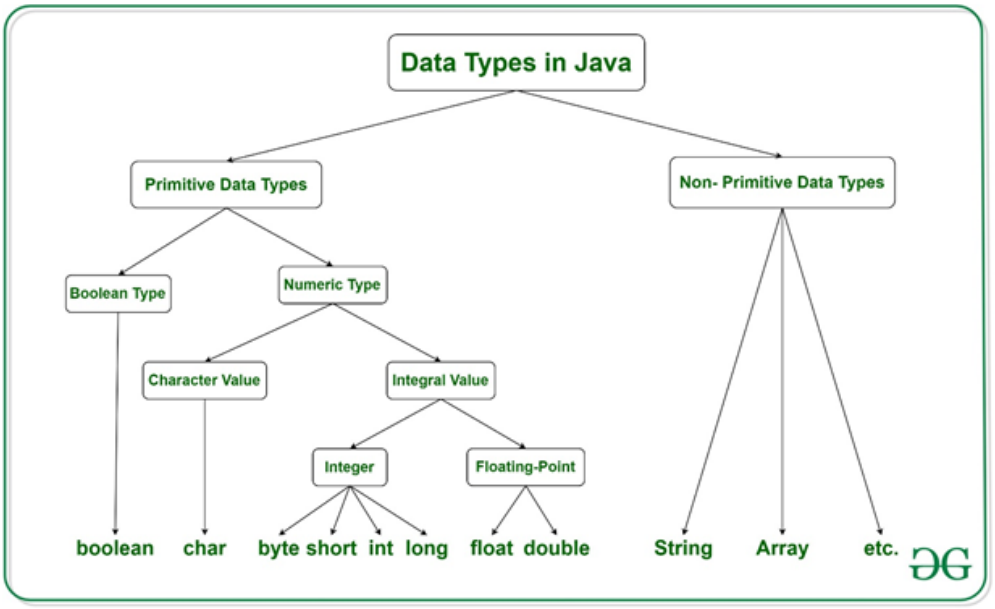
(نمونه‌ی do-while loop در جاوا)

دقت کنید که این نوع حلقه حداقل یک بار اجرا خواهد شد.

یکی از تفاوت‌های مهم حلقه‌ی while و do-while این است که در حلقه while، در ابتدا کار و قبل از اجرای دستورات حلقه، شرط پایان بررسی می‌شود. اما در حلقه do-while، این کار بعد از یک بار اجرای دستورات آن انجام می‌گردد. (مانند if، شرط جلوی while نیز حتما باید از نوع boolean باشد)

آشنایی با انواع داده‌ها در جاوا

به عکس زیر توجه کنید:



(انواع داده‌ها در جاوا)

با تمام زیرشاخه‌های primitive data types در گذشته آشنا شده‌اید.

جاوا یک نوع جدید، به نام String دارد که طرز استفاده از آن مانند متغیرهای دیگر است، ولی نسبت به آن‌ها قابلیت‌های بیشتری دارد.

برای مقداردهی String چندین راه وجود دارد که چند نمونه‌ی آن در زیر آمده است:

```
String s1 = "Java";
char[] ch = {'s', 't', 'r', 'i', 'n', 'g', 's'};
String s2 = new String(ch);
String s3 = "Example";
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
```

(مقداردهی String در جاوا)

خروجی:

```
java
strings
example
```

(خروجی تکه کد بالا)

اما String قابلیت‌های بیشتری نیز در اختیارمان قرار می‌دهد. برای مثال به تکه کد زیر توجه کنید:

```
String text = “Hello World!”;

System.out.println(“The length of text string is “ + text.length());
System.out.println(text.toUpperCase());
System.out.println(text.toLowerCase());

text = “locate where does ‘locate’ occurs!”;
System.out.println(text.indexOf(“locate”));
```

(متدهای مختلف کلاس String)

خروجی این نمونه کد:

```
The length of text string is 12
HELLO WORLD!
hello world!
0
```

(خروجی تکه کد بالا)

با ساختار این قابلیت‌ها در آینده آشنا می‌شوید.

معرفی enhanced switch

قبل‌تر با ساختار switch-case آشنا شدید. در آن مشابه ساختار if-esle if ladder، برای تصمیم‌گیری، بر اساس مقدار یک متغیر یا داده عمل می‌کردیم که در بعضی مواقع کار را برای ما آسان‌تر می‌کند.

به تکه کد زیر توجه کنید، در این مورد ما نیازی به break در هر case نداریم:

```java
String month = scanner.next();
switch (month) {
    case "April" -> System.out.println(month + ": it's first case");
    case "August" -> System.out.println(month + ": it's second case");
    case "June" -> System.out.println(month + ": it's third case");
    default -> System.out.println(month + ": None of cases");

}
```

(نمونه‌ی استفاده از enhanced switch)

چند مثال از ورودی و خروجی این کد:

```
April: it's first case
```

(خروجی تکه کد بالا برای ورودی April)

```
None of cases
```

(خروجی تکه کد بالا برای ورودی December)

نام گذاری متغیرها

در برنامه‌نویسی جاوا، نام‌گذاری مشهور و استاندارد متغیرها و متدها بر اساس نام‌گذاری camelCase است. در این روش، همه‌ی حروف به جز حروف اول کلمات دوم به بعد کوچک هستند. نمونه‌ی نام‌گذاری یک متغیر به این روش:

welcomeToAdvancedProgrammingWorkshop

همچنین در نام‌گذاری کلاس‌ها از متد PascalCase استفاده می‌شود. در این روش بر خلاف روش بالا، حروف اول همه‌ی کلمات بزرگ و باقی حروف کوچک هستند. نمونه‌ی نام‌گذاری یک کلاس به این روش:

AdvancedProgrammingLab

چند نکته:

- سعی کنید نام متغیرها معنی‌دار باشد و با توجه به کاربرد آن متغیر نام‌گذاری شده باشد.
- سعی کنید نام متدها معنی‌دار باشد و با توجه به عملکرد متد نام‌گذاری شده باشد. (در آینده با متدها آشنا می‌شوید)
- سعی کنید نام کلاس‌ها مرتبط با وظیفه‌ی آن‌ها نام‌گذاری شده باشید. (در آینده با کلاس‌ها آشنا می‌شوید)


گرفتن ورودی از کاربر در جاوا

ساده‌ترین راه برای گرفتن ورودی از کاربر در جاوا، استفاده از کلاس Scanner است. این کلاس در پکیج java.util قرار دارد. (راه‌های دیگری هم وجود دارد که ما از آن‌ها استفاده نمی‌کنیم)

برای استفاده از این کلاس مراحل زیر را انجام می‌دهیم:

۱- ابتدا آن را import می‌کنیم:

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

(نحوه‌ی import کردن کلاس Scanner)

۲- سپس لازم است خط زیر را در متد main برنامه بنویسیم: (در آینده خواهید دید که هرکدام از این عملیات چه معنایی دارد)

```
Scanner scanner = new Scanner(System.in);
```

(نحوه‌ی ساختن یک Scanner)

توجه: حتماً برای استفاده از Scanner، در طول اجرای برنامه آن را فقط یک بار بسازید (دلایل آن را در آینده خواهید دید)

حال Scanner ما آماده‌ی گرفتن ورودی است.

به مثال‌های زیر توجه کنید:

```
int anInt = scanner.nextInt();
double aDouble = scanner.nextDouble();
String aString = scanner.next();
String anotherString = scanner.nextLine();
long aLong = scanner.nextLong();
```

(متدهای مختلف کلاس Scanner)

با توجه به نوع ورودی، از یکی از دستورات بالا (و یا دیگر دستورات کلاس Scanner) استفاده می‌کنیم.

با اینکه خروجی scanner.next() و scanner.nextLine() هر دو String است اما این دو با هم تفاوت‌هایی دارند. متد next() تا زمانی به خواندن ورودی ادامه می‌دهد که به white space برخورد کند. اما nextLine() تمام خط را با spaceهای بین کلمات در String ذخیره می‌کند.

به طور مثال، اگر کاربر hello world! وارد کند، مقدار aString برابر با hello می‌شود.

فلسفه‌ی گیت

گیت مشهورترین Version Control System است. در این قسمت، ابتدا با کاربرد سیستم‌های کنترل ورژن آشنا می‌شویم.

معرفی Version Control System

فرض کنید شما عضو یک تیم برنامه‌نویسی هستید و به همراه اعضای تیم، تصمیم می‌گیرید که نسخه‌ی جدیدی از پروژه‌تان را ارائه کنید. برای هماهنگی بین اعضای تیم و ادغام کردن تسک[1]هایی که هر عضو انجام داده چه راهکاری دارید؟ ورژن جدید و قبلی را در چه حافظه‌ای و به چه صورتی ذخیره می‌کنید؟

یک پاسخ ساده به این سؤالات بدین شرح است که برای هر ورژن، یک دایرکتوری[2] مجزا وجود داشته باشد. هم‌چنین هر یک از اعضا در سیستم خود، تغییراتی که باید را اعمال کند و سپس این تغییرات به صورت دستی باهم ترکیب شوند.

به کارگیری این روش بسیار خسته‌کننده و زمان‌بر است. سیستم‌های کنترل ورژن برای سهولت این کار ایجاد شده‌اند. با استفاده از این سیستم‌ها، تمام تغییرات، مرحله به مرحله، در یک مخزن (ریپازیتوری[3]) نگهداری می‌شوند و اعضای تیم می‌توانند با دسترسی به این مخزن، در ایجاد تغییرات هماهنگی داشته باشند.

[1] task
[2] directory
[3] repository

دسته‌بندی Version Control System

سیستم‌های کنترل ورژن به دو دسته‌ی centralized و distributed تقسیم می‌شوند. گیت یک سیستم distributed است. در مورد تفاوت این دو دسته در این لینک مطالعه کنید.

حال که با ضرورت استفاده از سیستم‌های کنترل ورژن آشنا شدیم، جزئیات بیشتری را در مورد نحوه‌ی کار این سیستم‌ها بررسی می‌کنیم.

Areaها در گیت

در توضیحات قبل گفته شد که با استفاده از یک سیستم کنترل ورژن می‌توان نسخه‌های مختلف پروژه را در یک مخزن نگهداری کرد. حال نحوه‌ی کار سیستم گیت را یاد می‌گیریم و با areaها در گیت بیشتر آشنا می‌شویم.

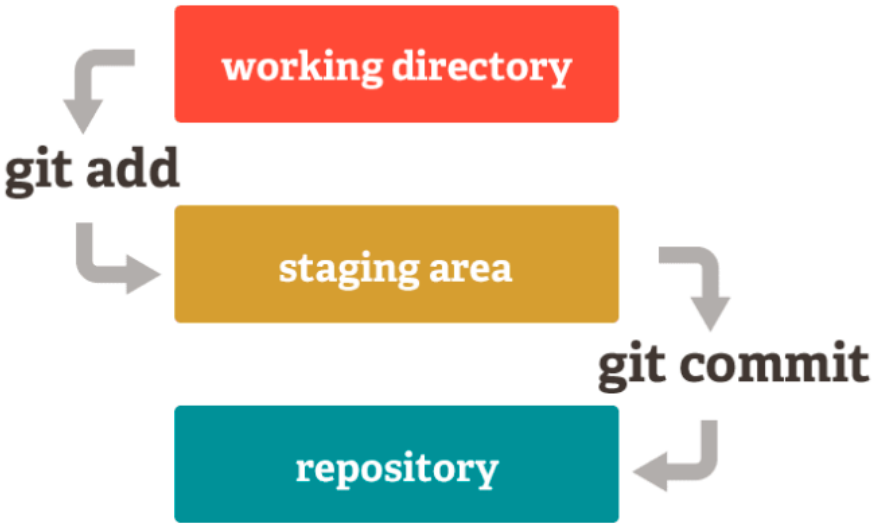
برای ساختن یک مخزن جهت نگهداری ورژن‌ها، از دستور init استفاده می‌کنیم. این دستور یک فایل مخفی git. در دایرکتوری پروژه ایجاد می‌کند:

```
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ git init
Initialized empty Git repository in /home/sarv2000/uni/TA-AP/Spring 2021/snake/.
git/
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ ls -a
.  ..  .DS_Store  .git  .idea  snake.iml  src
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ ls
snake.iml  src
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$
```

(نحوه‌ی اضافه کردن گیت به یک دایرکتوری)

در مرحله‌ی بعد، باید فایل‌های پروژه را به مخزن منتقل کنیم.

برای درک بهتر، ابتدا به تصویر زیر توجه کنید:



(انواع Area ها در گیت)

همان‌طور که مشاهده می‌کنید، سه ناحیه در کار با گیت به کار گرفته می‌شوند. Working area همان دایرکتوری پروژه‌ی ماست. repository، همان‌طور که توضیح داده شد، یک مخزن است که نسخه‌های پروژه را نگهداری می‌کند. ما برای کنترل نسخه‌ها باید به نحوی پروژه را پس از هر تغییر، از working directory به ریپازیتوری منتقل کنیم. این کار با استفاده از دستور commit صورت می‌گیرد. در ادامه در مورد این دستور توضیح خواهیم داد.

اما بین working directory و مخزن، فضای دیگری به نام staging area یا index وجود دارد. این فضا برای آن به کار گرفته می شود که گیت بتواند تغییراتی که باید وارد ریپازیتوری شوند را تشخیص دهد. گاهی نیازی نیست تمام تغییرات working area را به ریپازیتوری منتقل کنیم.

مثلا فرض کنید شما در حال اضافه کردن یک فیچر[1] به پروژه هستید و در این میان متوجه می‌شوید که بخش‌هایی از ورژن قبلی باگ[2] خورده و باید فورا اصلاح شوند (پیش از آنکه فیچر جدید اضافه شود). مثلا برای دیباگ[3] کردن شما ۲ فایل را در working directory تغییر می‌دهید و می‌خواهید ورژن دیباگ شده را وارد ریپازیتوری کنید. از طرفی پیش از این ۳ فایل دیگر نیز برای اضافه کردن فیچر جدید تغییر کرده‌اند. اگر staging area وجود نداشته باشد و بخواهیم فایل‌ها را مستقیما از working directory به ریپازیتوری منتقل کنیم، تمام این ۵ فایل وارد ریپازیتوری می‌شوند. در صورتی که ورژن دیباگ شده فقط تغییرات ۲ فایل دیباگ را نیاز دارد.

بنابراین staging area طراحی شده و برای استفاده از آن، ابتدا با استفاده از دستور add فایل‌های مورد نظر را به staging area انتقال می‌دهیم و سپس با دستور commit، فایل‌ها را به مخزن منتقل می‌کنیم:

```
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ git add snake.iml
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ git commit -m "commit message"
[master (root-commit) 2320d30] commit message
 1 file changed, 11 insertions(+)
 create mode 100644 snake.iml
sarv2000@sarv2000:~/uni/TA-AP/Spring 2021/snake$ git log
commit 2320d308077e344024cb8dc43122a7095fa025c3 (HEAD -> master)
Author: sarvenaz-srv <sarvghad79@gmail.com>
Date:   Wed Aug 18 11:24:51 2021 +0430

    commit message
```

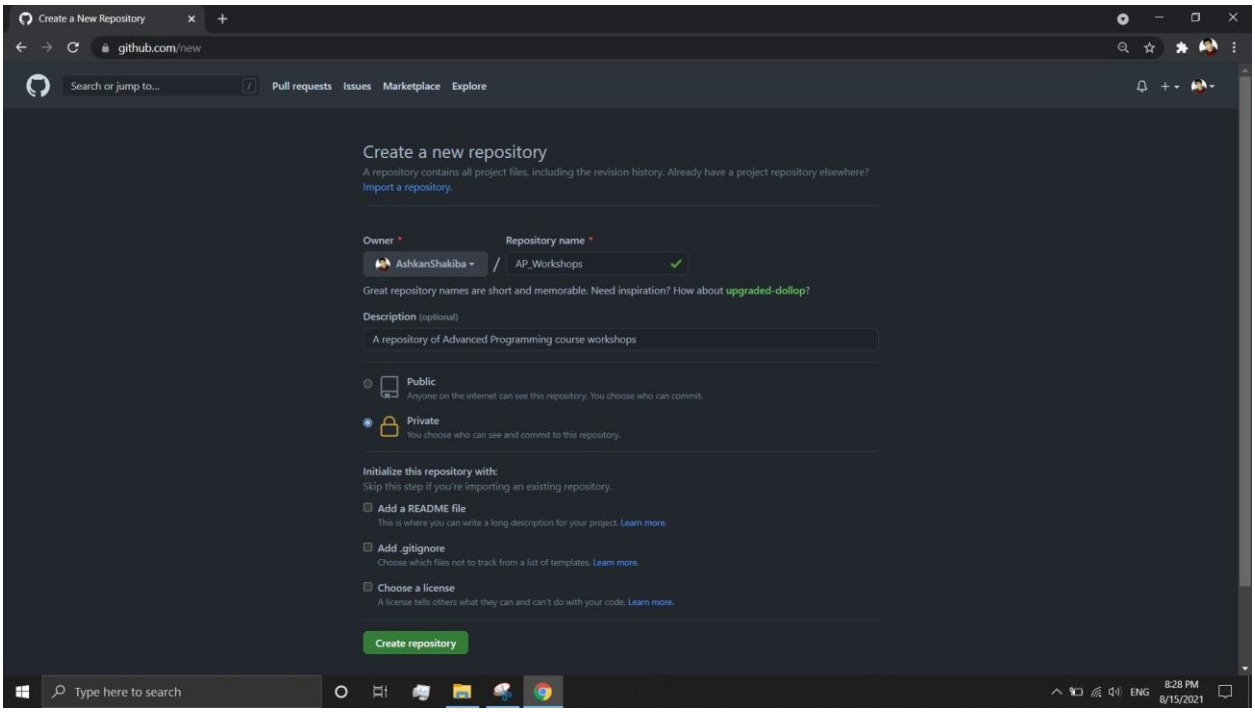
(انتقال فایل‌ها از working directory به مخزن)

در دستور کار قبلی نحوه‌ی ایجاد حساب کاربری در گیت‌هاب را به طور دقیق بررسی کردیم. حال به ادامه‌ی یادگیری کار کردن با گیت‌هاب می‌پردازیم.

[1] feature
[2] bug
[3] debug

ساخت مخزن در گیت‌هاب

پس از انتخاب گزینه‌ی New در بخش your repositories در صفحه‌ی اول وب‌سایت گیت‌هاب، در صفحه‌ی باز شده در بخش Repository name، نامی مناسب برای آن انتخاب کنید: (توجه کنید که نمی‌توانید از space استفاده کنید و به جای آن می‌توانید از کاراکترهایی مثل - یا _ استفاده کنید)



(صفحه‌ی ساخت مخزن در وب‌سایت گیت‌هاب)

سپس در بخش Description، می‌توانید توضیحی مختصر درباره‌ی پروژه‌تان اضافه کنید.

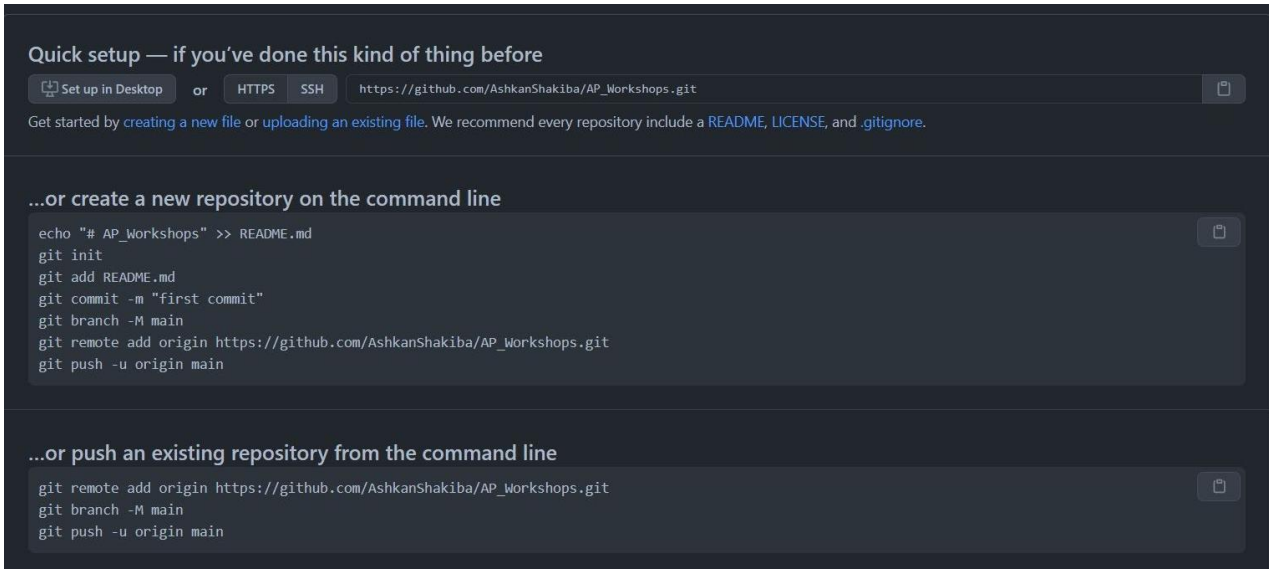
در بخش بعدی نیز می‌توانید انتخاب کنید که چه کسانی به مخزن دسترسی داشته باشند؛ در صورتی که گزینه‌ی Public را انتخاب کنید همه می‌توانند به آن دسترسی داشته باشند در حالی‌که با انتخاب Private، فقط خودتان و دیگر افرادی که شما به آن‌ها دسترسی بدهید به مخزن دسترسی خواهند داشت.

در نهایت بر روی Create repository کلیک کنید.

صفحه‌ی مربوط به مخزن برای شما باز می‌شود و پس از این نیز از طریق منوی صفحه اصلی گیت‌هاب و یا آدرس

https://github.com/your_username/repository_name

می‌توانید به آن دسترسی داشته باشید:



(صفحه‌ی مخزن ساخته شده)

همچنین در نوار Quick setup، گزینه HTTPS را انتخاب کنید و آدرسی که در فیلد روبروی آن وجود دارد را کپی کنید، چرا که برای معرفی remote به گیت، به این آدرس نیاز خواهیم داشت.

ساخت مخزن گیت

فولدری برای مخزن‌تان بسازید و در آن ترمینال را باز کنید.

می‌توانید از CMD و یا دیگر ترمینال‌ها استفاده کنید، اما پیشنهاد می‌شود از Git Bash استفاده کنید. (برای استفاده از Git Bash می‌توانید در فولدر مورد نظر کلیک راست کرده و گزینه Git Bash Here را انتخاب کنید)

سپس با دستور git init از گیت می‌خواهیم که این فولدر را به عنوان یک مخزن گیت بشناسد و مدیریت کند: (با این کار فولدر .git نیز به شکل پنهان به دایرکتوری پروژه‌تان اضافه می‌شود که وظیفه‌ی نگهداری اطلاعات و داده‌های مخزن را بر عهده دارد)

(ساخت دایرکتوری گیت)

سپس با دستور زیر یک remote به مخزن اضافه می‌کنیم:

git remote add <remote-name> <remote-address>

به جای remote-address از آدرسی که در گیت‌هاب کپی کردیم استفاده می‌کنیم و همچنین برای remote-name، می‌توانید از هر نام دلخواهی استفاده کنید، اما رایج است که از نام origin استفاده شود:

```
A.Sh@DESKTOP-B3PHD2K MINGW64 ~/Desktop/AP_Workshops (master)
$ git remote add origin https://github.com/AshkanShakiba/AP_Workshops.git
```

(دستور remote add)

در هر مرحله با دستور git status می‌توان وضعیت فایل‌ها و دایرکتوری‌ها را مشاهده کرد. در تصویر زیر هر سه فایل در وضعیت Untracked قرار دارند و هنوز به گیت معرفی نشده‌اند:

```
A.Sh@DESKTOP-B3PHD2K MINGW64 ~/Desktop/AP_Workshops (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        File01.java
        File02.java
        File03.java
```

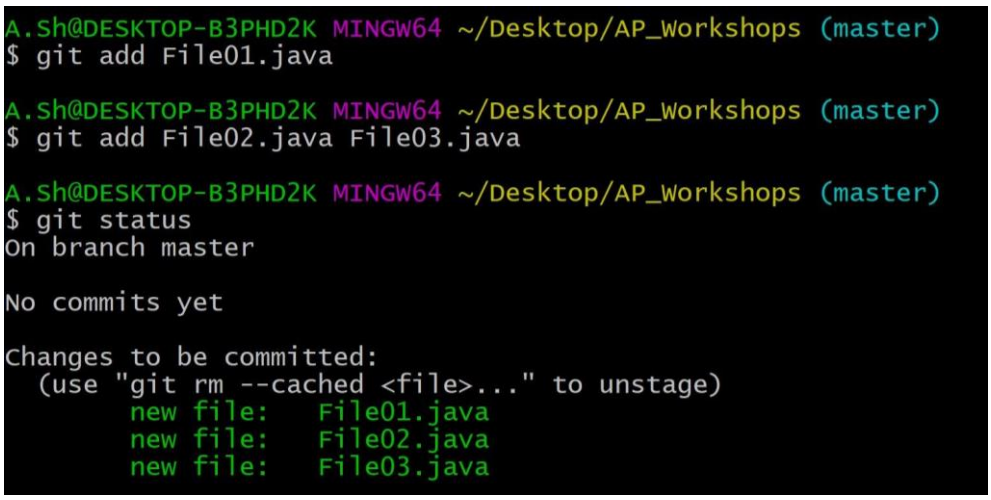
(خروجی دستور status)

با دستور زیر می‌توانید فایل‌ها و دایرکتوری‌ها را وارد وضعیت staged کنید:

git add <file>

به جای file، آدرس فایل یا دایرکتوری قرار می‌گیرد و می‌توانید بیش از یک فایل را نیز اضافه کنید. برای مثال با دستور زیر می‌توان دایرکتوری فعلی و همه‌ی زیردایرکتوری‌های آن را staged کرد:

git add .



(نحوه‌ی stage کردن فایل‌های مورد نظر)

در هر مرحله با استفاده از دستور git status، می‌توانید گزارشی از وضعیت فایل‌ها داشته باشید و پیش از هر کامیت یا اقدام دیگر، آنها را بررسی کنید.

همچنین با دستور زیر می‌توانید تغییرات staged شده را کامیت کنید تا در حافظه‌ی گیت ثبت شود و پس از این بتوانید به وضعیت کنونی فایل‌ها دسترسی داشته باشید: (به جای message، پیغامی مناسب برای کامیت خود اضافه کنید)

git commit -m ‘<message>’

نکته: در صورتی که تنها دستور git commit را وارد کنید، ویرایشگر متنی در ترمینال برایتان باز می‌شود که در آن می‌توانید پیغام‌تان را بنویسید و در انتها برای خروج از آن، یکبار کلید Esc را زده و سپس دستور wq: را وارد کنید و Enter را بزنید.

برای آشنایی با استاندارد نوشتن پیغام، به پیوست انتهای دستور کار مراجعه کنید.

در هر مرحله، می‌توان با دستور زیر به جزئیات کامیت‌های قبلی دسترسی داشت:

git log

در نهایت با دستور زیر می‌توان کامیت‌ها را به مخزن ساخته شده در گیت‌هاب فرستاد:

git push <remote-name> <branch-name>

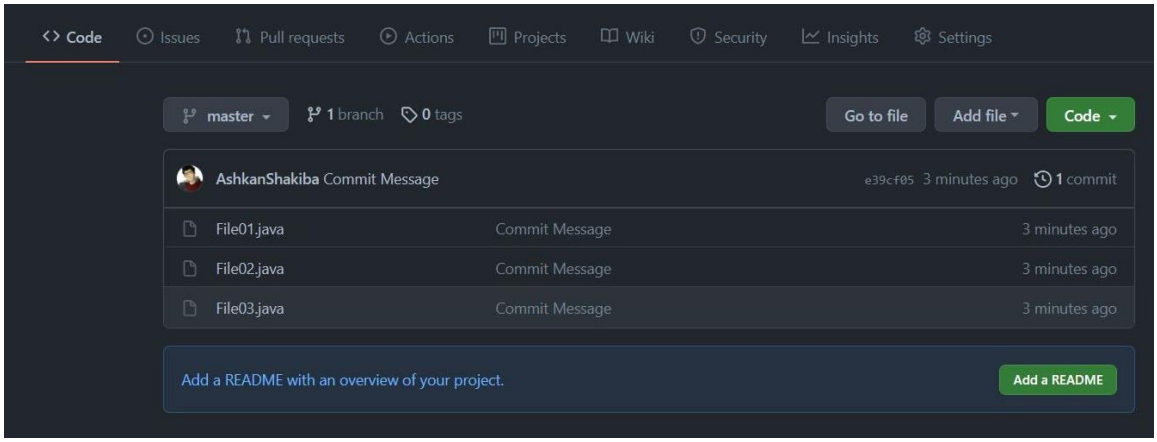
```
A.Sh@DESKTOP-B3PHD2K MINGW64 ~/Desktop/AP_Workshops (master)
$ git commit -m 'Commit Message'
[master (root-commit) e39cf05] Commit Message
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 File01.java
 create mode 100644 File02.java
 create mode 100644 File03.java

A.Sh@DESKTOP-B3PHD2K MINGW64 ~/Desktop/AP_Workshops (master)
$ git log
commit e39cf05b753426a1d5d75808fea3223af5d44d3d (HEAD -> master)
Author: Ashkan Shakiba <AshkanShakiba11@gmail.com>
Date:   Mon Aug 16 11:48:47 2021 +0430

    Commit Message

A.Sh@DESKTOP-B3PHD2K MINGW64 ~/Desktop/AP_Workshops (master)
$ git push origin master
```

(کامیت کردن و فرستادن آن به ریموت)

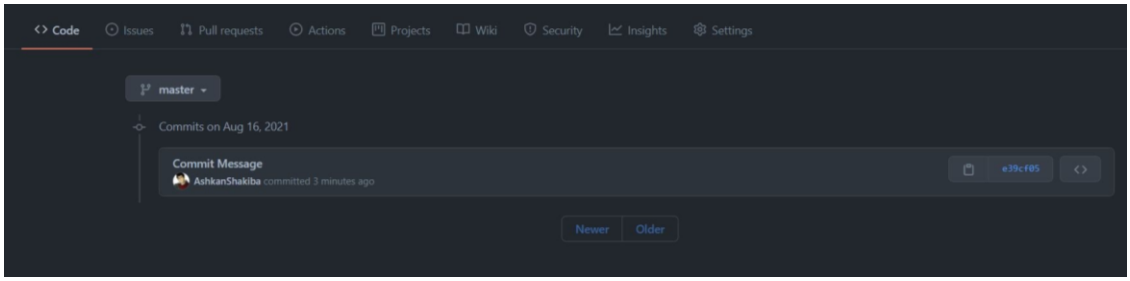
توجه کنید که به جای remote-name، نام ریموت که معمولاً origin است، قرار می‌گیرد و به جای branch-name، نام شاخه‌ای که بر روی آن در حال کار هستید؛ شاخه‌ی پیشفرض، master است که اگر تاکنون شاخه‌تان را عوض نکرده‌اید، از آن باید استفاده کنید. (شاخه‌ای که بر روی آن در حال فعالیت هستید در خط بالای هر دستور در انتها و درون پرانتز نوشته شده است)

لزومی ندارد که پس از هر کامیت این کار را انجام دهید و تعداد کامیت‌های پوش شده می‌تواند متفاوت باشد.

اکنون اگر به صفحه‌ی مخزن در گیت‌هاب مراجعه کنید، می‌توانید تغییرات اعمال شده را مشاهده کنید. ضمناً با کلیک بر روی تعداد کامیت‌ها، صفحه ای شامل جزئیات کامیت‌های پوش شده باز می‌شود:



(صفحه‌ی مخزن در گیت‌هاب)



(مشاهده‌ی سابقه‌ی کامیت‌های ارسال شده)

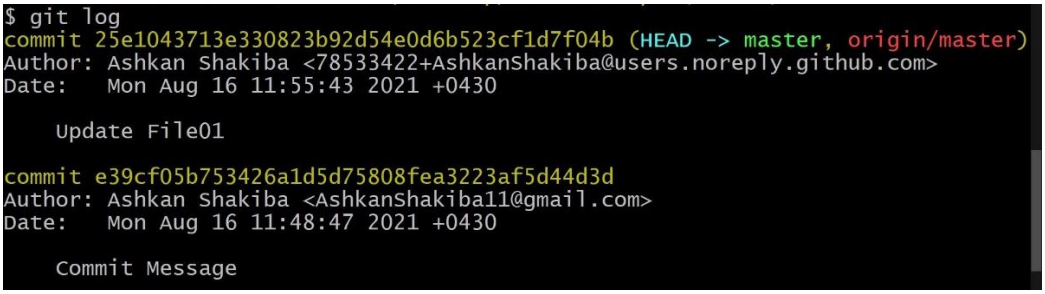
در این صفحه، به جزئیات و اطلاعات کامیت‌های پوش شده بر روی شاخه‌های مختلف مخزن دسترسی دارید. (شاخه پیشفرض master است)

همچنین اگر افراد دیگری به جز شما نیز در حال کار بر روی مخزن باشند، با دستور زیر می‌توانید کامیت‌هایی که دیگر افراد بر روی مخزن گیت‌هاب پوش کرده‌اند را بر روی مخزن local خود، دریافت (pull) کنید:

git pull <remote-name> <branch-name>

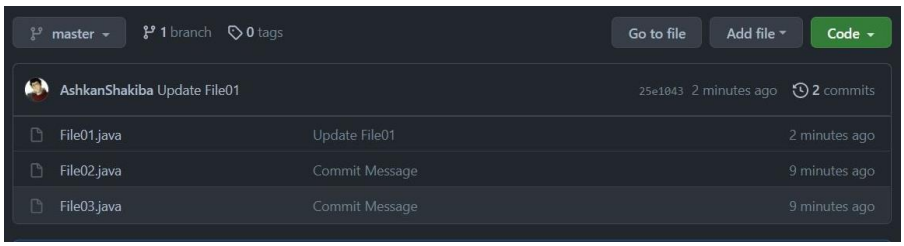
(دریافت کامیت‌ها از ریموت)

مشابه قبل و با کمک دستور git log می‌توانید به کامیت‌های انجام شده دسترسی داشته باشید:



(مشاهده‌ی سابقه‌ی کامیت‌ها)

همچنین این تغییرات بر روی صفحه گیت‌هاب مخزن نیز قابل دسترسی‌اند:



(مشاهده‌ی تغییرات در گیت‌هاب)

انجام دهید: ماشین‌حساب ساده

برنامه‌ای بنویسید که دو عدد و یکی از چهار عمل اصلی (* / + -) را از کاربر دریافت کرده، نتیجه را محاسبه کند و به کاربر نشان دهد.

نمونه ورودی و خروجی:

Input:

7

*

5

Output: 35

نحوه‌ی تحویل

قبل از پیاده‌سازی این تمرین، لازم است که مخزنی جدید در گیت‌هاب با نام AP-Workshop1 برای خودتان بسازید. دقت کنید که مخزنی که می‌سازید، حتماً از نوع private باشد که باقی افراد به آن دسترسی نداشته باشند. از این به بعد نیز لازم است مانند این تمرین، برای هر دستورکار یک مخزن جدید با نام AP-WorkshopN که در آن N شماره‌ی دستورکار است بسازید و تمرین خود را در آن انجام دهید.

برای پیاده‌سازی این تمرین، لازم است که پس از انجام دادن هر بخش، تغییرات داده شده را در کامیتی اعمال کنید و سپس تمامی کامیت‌ها را با هم به مخزن گیت‌هابتان پوش کنید. تعداد و جزئیات هر کامیت به صورت زیر است:

۱. ابتدا قسمت ورودی گرفتن تمرین را کامل کنید و تغییرات را در کامیتی با پیغام مناسب اعمال کنید.

۲. سپس تغییرات لازم برای محاسبه‌ی پاسخ را در کدتان اعمال کنید و در کامیت دیگری قرار دهید.

۳. پس از آن، بخش چاپ کردن پاسخ را نیز کامل کنید و در کامیت مجزایی قرار دهید.

۴. در آخر تمامی کامیت‌های ساخته شده را به سمت مخزن گیت‌هابتان پوش کنید.

پیوست: روش استاندارد نوشتن پیام کامیت

رایج است که برای نخستین کامیت، از پیغام Initial commit استفاده می‌شود.

همچنین برای کامیت‌های بعدی، پیشنهاد می‌شود که از فرمت

<type>: <description>

استفاده کنید که به جای type:

- اگر یک مشکل یا باگ را حل کرده‌اید، از fix استفاده کنید.
- اگر قابلیتی جدید به برنامه افزوده‌اید و آن را تکمیل کرده‌اید، از feat استفاده کنید.
- اگر بدون تغییر خروجی برنامه، کد نوشته شده را مرتب کرده‌اید، از refactor استفاده کنید.

سپس به جای description، توضیحاتی درباره کامیت اضافه کنید.