



# کارگاه برنامه نویسی پیشرفته

## دستور کار شماره دو

### اهداف

---

- نحوه ساخت فایل gitignore
- شاخه‌ها در گیت
- دستورات تکمیلی گیت
- مفاهیم کلاس و شی
- میانبرها در اینتلیجی<sup>۱</sup>

---

<sup>۱</sup> IntelliJ



## فهرست مطالب

۳

۶

۸

۱۳

۱۵

۲۳

۲۵

۳۱

۳۶

آشنایی با gitignore

شاخه‌ها در گیت<sup>۱</sup>

نحوه ساخت شاخه

آشنایی با pull request

دستورات تکمیلی گیت

Collaboration در گیت

مفاهیم کلاس و شی

میانبرها در اینتلیجی

انجام دهید

---

<sup>۱</sup> Git

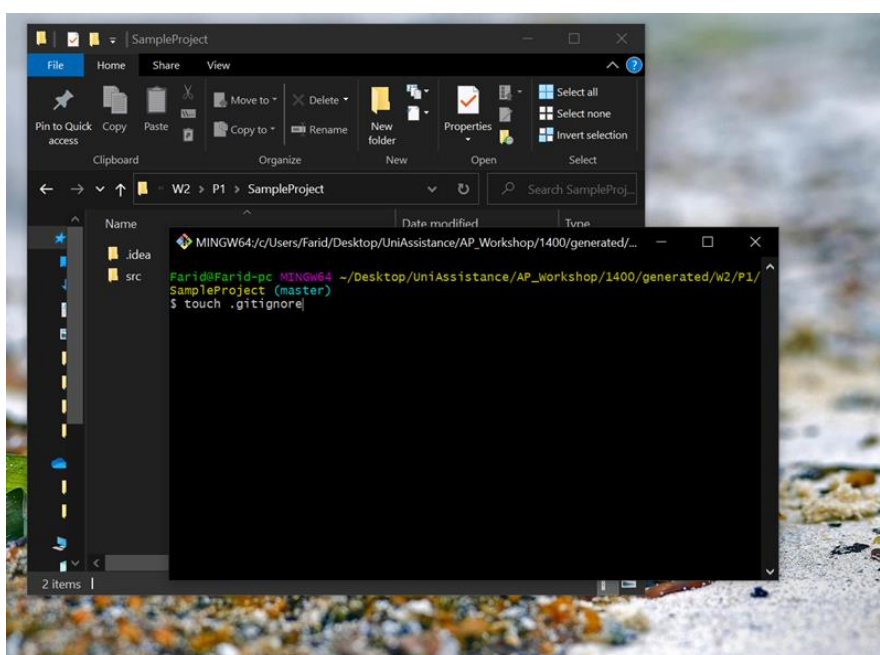


## آشنایی با gitignore

اکثر مواقع، در هر پروژه‌ای، فایل‌هایی داریم که نباید روی گیت باشند. یا حتی گاهی در پروژه‌های تیمی، نیاز است که بعضی از فایل‌ها را صرفاً به صورت محلی داشته باشیم. برای مثال، نگاهی به مخزن‌های قبلی خود در گیت‌هاب بیندازید. اگر از ابزار gitignore استفاده نکرده باشید، باید شاهد یک پوشه با نام idea باشید. این پوشه، شامل اطلاعاتی است، که صرفاً IDE شما برای هر پروژه تولید می‌کند و به آن نیاز خواهد داشت. بنابراین لزومی ندارد این پوشه، در کامیت‌های شما باشد و روی گیت قرار گیرد.

### نحوه استفاده

برنامه گیت بش<sup>۱</sup> را باز می‌کنیم و با استفاده از دستور gitignore touch یک فایل gitignore. در آدرس پروژه می‌سازیم:

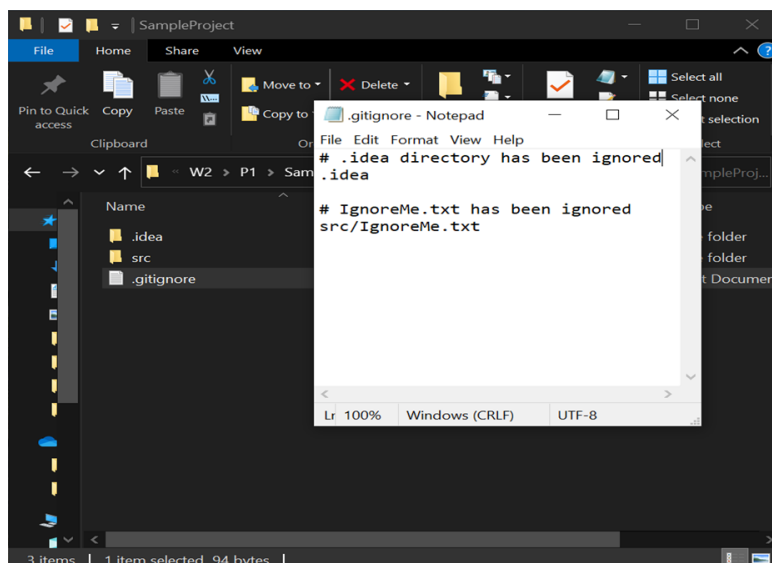


(ساخت فایل gitignore)

<sup>۱</sup> git bash



سپس در فایل gitignore ساخته شده، اسم و آدرس فایل‌هایی را که می‌خواهیم نادیده گرفته شوند را می‌نویسیم و تغییرات را ذخیره می‌کنیم.



(پس از ذخیره کردن فایل `.gitignore`، فایل `IgnoreMe.txt` و همچنین پوشه‌ی `.idea`، ایگنور می‌شوند)

### استفاده از تمپلیت‌های آماده

ابزارهایی برای تولید فایل gitignore متناسب با آنچه شما نیاز دارید وجود دارند. برای مثال، برنامه intelliJ پلاگین‌هایی برای این کار دارد که به آن نمی‌پردازیم. پیشنهاد ما برای ساخت فایل gitignore آماده، استفاده از لینک زیر است:

<https://www.toptal.com/developers/gitignore>



این سایت صرفاً با در اختیار داشتن چند کلیدواژه، گیت ایگنور مورد نیاز شما را تولید می‌کند. پس از آن کافیست محتوای تولید شده را در فایل `.gitignore` خود کپی نمایید:

**gitignore.io**

Create useful .gitignore files for your project

IntelliJ X Java X Create

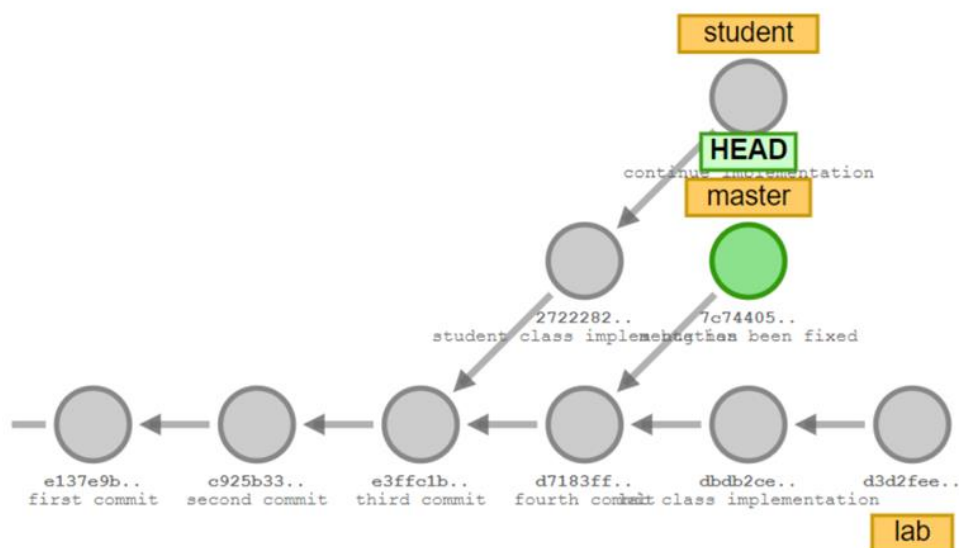
[Source Code](#) | [Command Line Docs](#)

(پس از کلیک بر روی دکمه‌ی `Create`، فایل `.gitignore` برای زبان جاوا و برنامه `IntelliJ` ساخته می‌شود)



## شاخه‌ها در گیت

در برنامه‌نویسی، اصولاً با مباحث بزرگی روبرو هستیم که برای حل کردنشان، آن‌ها را به مسائل ریزتر و کوچک‌تر، تقسیم می‌کنیم. در نتیجه، یک مسئله بزرگ تبدیل به چندین مسئله کوچک خواهد شد. در گیت بهتر است هر کدام از این مسئله‌ها را در شاخه‌ای<sup>۱</sup> مجزا پیاده‌سازی کنیم و هنگامی که پیاده‌سازی هر شاخه کامل شد، تغییرات را بر روی شاخه اصلی اضافه کنیم<sup>۲</sup>. این کار علاوه بر اینکه به نظم ذهنی خودمان و البته نظم ساختاری برنامه کمک می‌کند، چالش کار تیمی را نیز برطرف می‌کند. وقتی شاخه‌های مجزا داشته باشیم، هر فرد می‌تواند مستقل از فرد دیگری، قسمتی از پروژه را به صورت همزمان با افراد دیگر، جلو ببرد. برای درک بهتر ماجرا، به مثال زیر توجه کنید:



(شاخه‌ها در گیت)

در قسمت تمرین تحویلی این دستورکار، شما باید نمونه‌ای ساده شده از وضعیت کنونی کارگاه‌ها را پیاده‌سازی کنید.

به این صورت که باید پروژه‌ی شما یک کلاس برای هر دانشجو و همچنین برای هر کارگاه داشته باشد.

نکته: برای درک بهتر ساختار این پروژه می‌توانید به صفحه ۳۶ مراجعه فرمایید.

برای مثال می‌توانیم، پیاده‌سازی دانشجو‌ها را در یک شاخه، و پیاده‌سازی کارگاه را در شاخه‌ای دیگر انجام دهیم.

<sup>۱</sup> Branch

<sup>۲</sup> Merge



همانطور که می‌بینید، در این مثال سه شاخه داریم. یکی همان شاخه اصلی<sup>۱</sup> است که در نهایت تمام تغییرات باید روی آن مرج شود. شاخه‌های بعدی، شاخه student است که پیاده‌سازی کلاس Student را در آن انجام می‌دهیم، و شاخه‌ی بعدی نیز شاخه lab، که پیاده‌سازی کلاس Lab نیز در آن انجام می‌گیرد.

فرض کنید همین تمرین کوچک، یک تمرین تیمی دو نفره بود. در این صورت، هر فرد می‌توانست در شاخه متناظر، کلاس مربوطه را پیاده‌سازی کند. توجه کنید که شاخه‌ها فقط برای کار تیمی نیستند!

---

<sup>۱</sup> master



## نحوه ساخت شاخه

ترمینال‌های مختلفی هستند که می‌توانید از آن‌ها استفاده کنید: ترمینال سیستم عامل، ترمینال اینترنتی و یا گیت بش. در هر حال دستورات گیت در تمام ترمینال‌ها یکسان است و شما می‌توانید برای خود یکی را انتخاب، و با آن کار کنید.

### معرفی دستورات ضروری

```
git branch [branch name]
```

یک شاخه جدید به نام مورد نظر ایجاد می‌کند. (اما روی شاخه جدید نمی‌رود!)

```
git checkout [branch name]
```

به شاخه مورد نظر می‌رود.

```
git checkout -b [branch name]
```

یک شاخه جدید می‌سازد و روی آن می‌رود.

```
git merge [branch name]
```

شاخه مورد نظر را روی شاخه فعالِ مرج می‌کند. برای مثال، اگر روی شاخه A باشیم و دستور بالا را برای شاخه B اجرا کنیم، شاخه B روی شاخه A مرج خواهد شد.





مثال: می‌خواهیم یک شاخه جدید به نام test بسازیم و در آن یک کامیت<sup>۱</sup> انجام دهیم و سپس آن را روی شاخه اصلی، مرج کنیم:

```
MINGW64/c/Users/Farid/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/SampleProject

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/
SampleProject (master)
$ git branch test

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/
SampleProject (master)
$ git checkout test
Switched to branch 'test'

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/
SampleProject (test)
$ git commit -am "just a test"
[jest 91646a8] just a test
1 file changed, 1 insertion(+), 1 deletion(-)

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/
SampleProject (test)
$ git push -u origin test
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 416 bytes | 208.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:   https://github.com/farid-127/SampleProject/pull/new/test
remote:
To https://github.com/farid-127/SampleProject.git
 * [new branch]      test -> test
Branch 'test' set up to track remote branch 'test' from 'origin'.

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/
SampleProject (test)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/SampleProject (master)
$ git merge test
Updating 29267d4..91646a8
Fast-forward
 src/Main.java | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

Farid@Farid-pc MINGW64 ~/Desktop/UniAssistance/AP_Workshop/1400/generated/W2/P2/SampleProject (master)
$ git push
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/farid-127/SampleProject.git
 29267d4..91646a8 master -> master
```

※رفع ابهام:

```
git push -u origin [branch name]
```

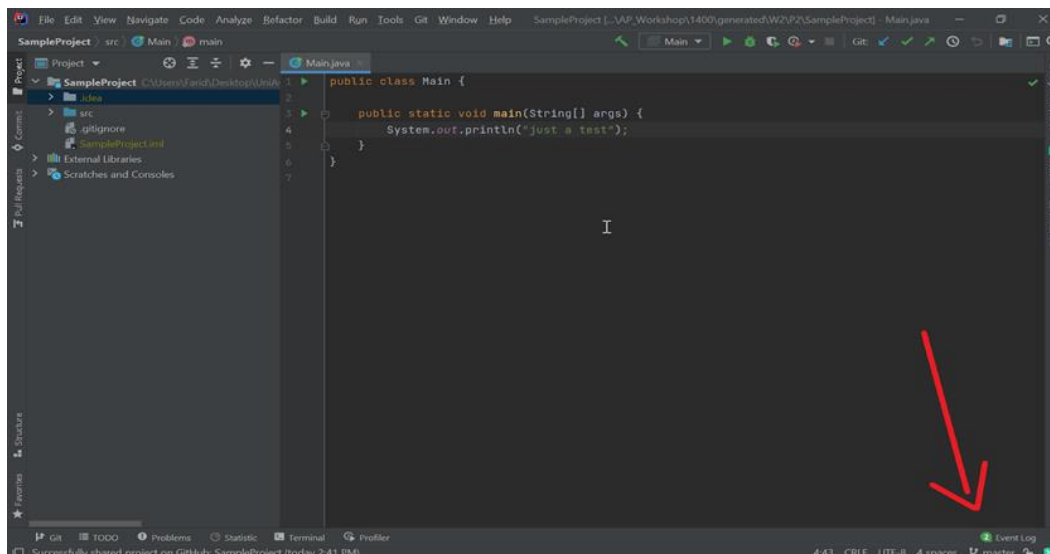
دستور بالا زمانی استفاده می‌شود که می‌خواهیم یک local branch را برای اولین بار push کنیم.

<sup>۱</sup> commit

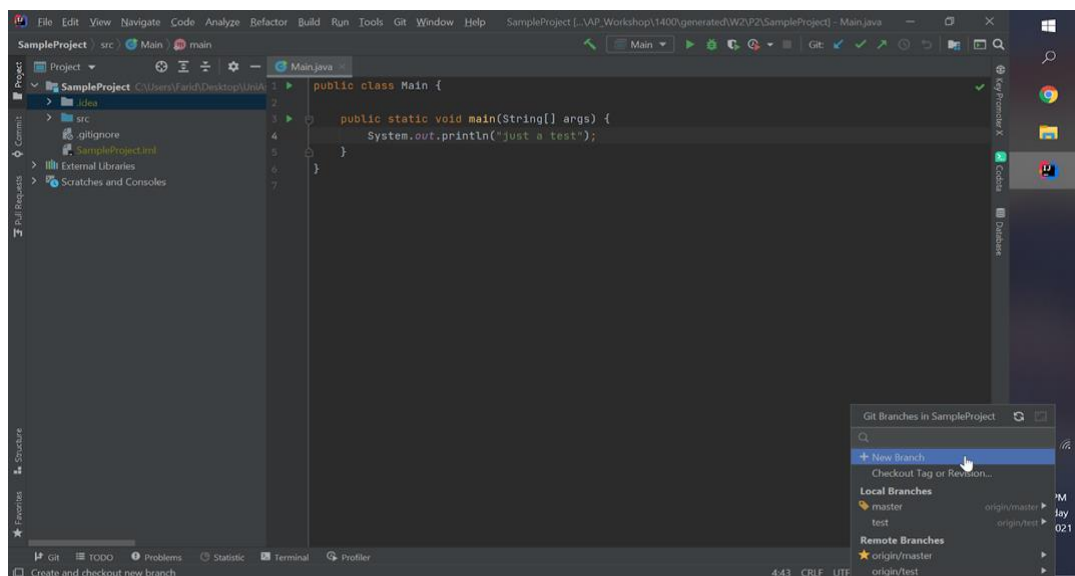


## ساخت شاخه با GUI Tool

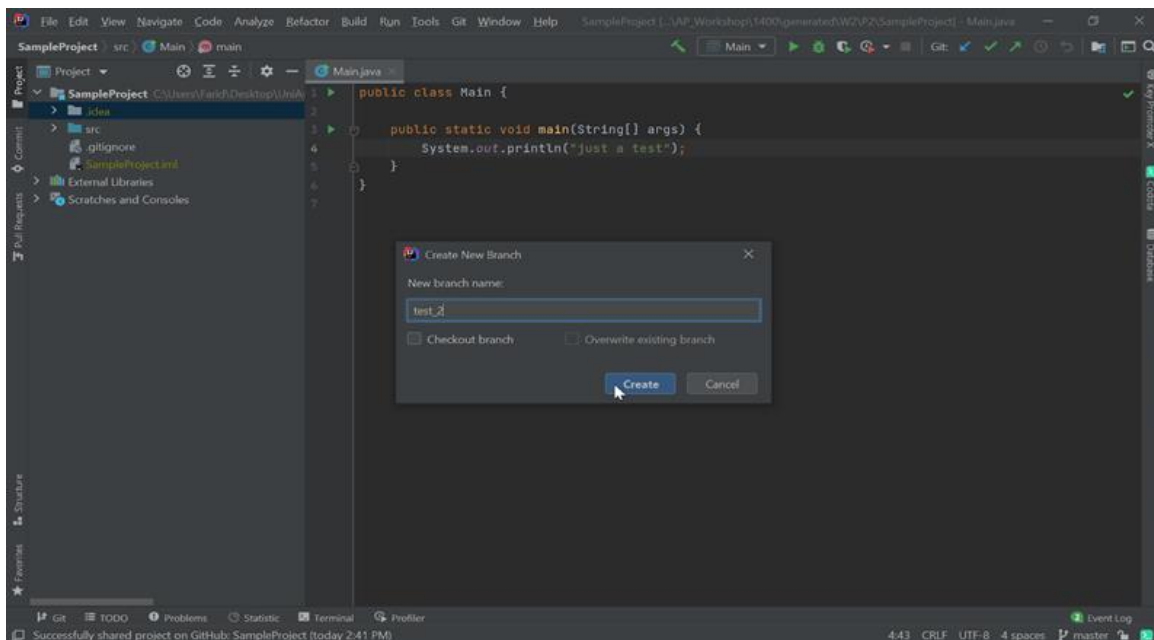
ابزارهای گرافیکی ای وجود دارند که می‌توانید از آن‌ها نیز استفاده کنید. برای مثال، اینتلیجی این قابلیت را دارد. همان مثال قبل را این بار در اینتلیجی انجام می‌دهیم:



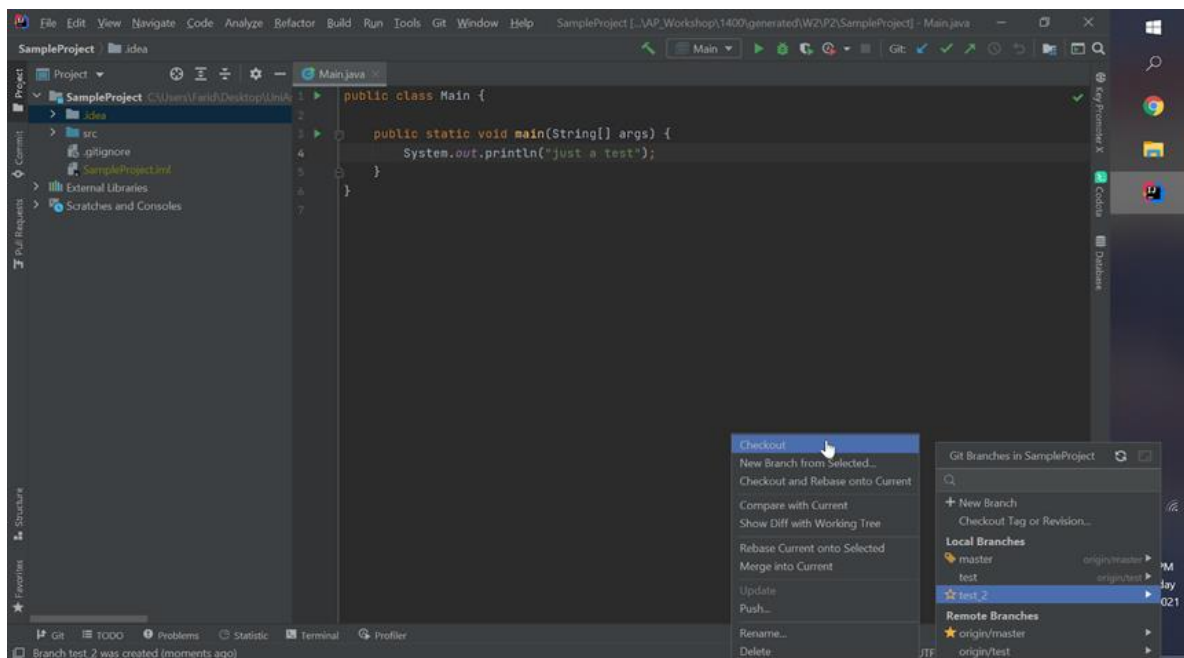
(روی دکمه master کلیک می‌کنیم)



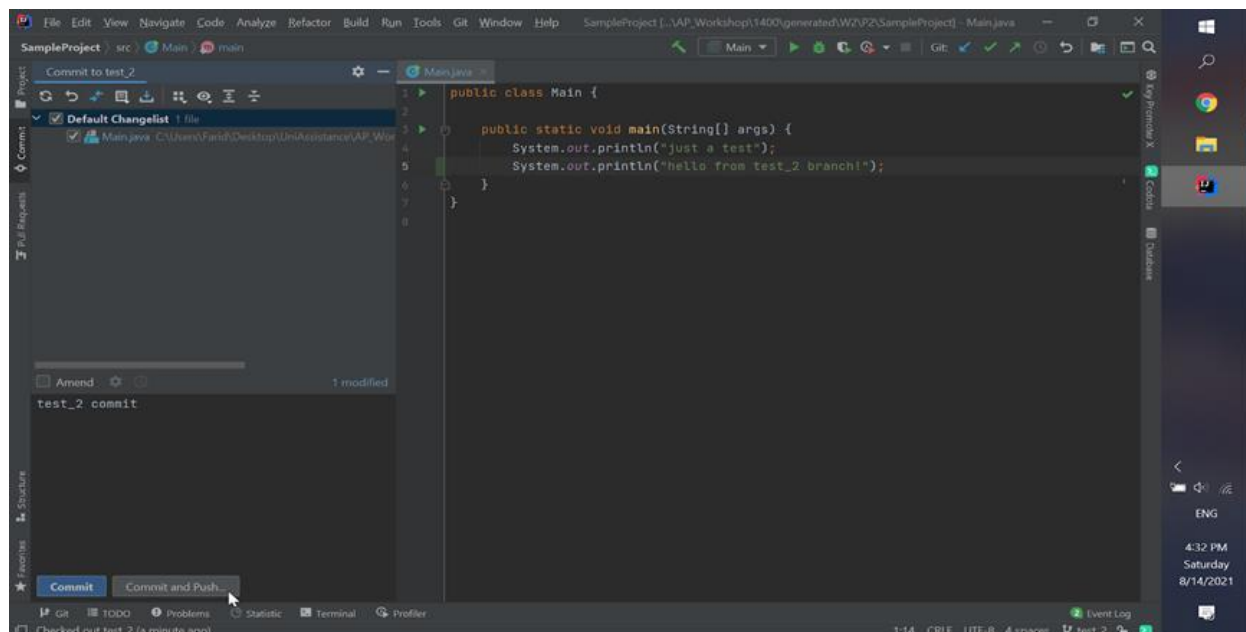
(گزینه New Branch را انتخاب می‌کنیم)



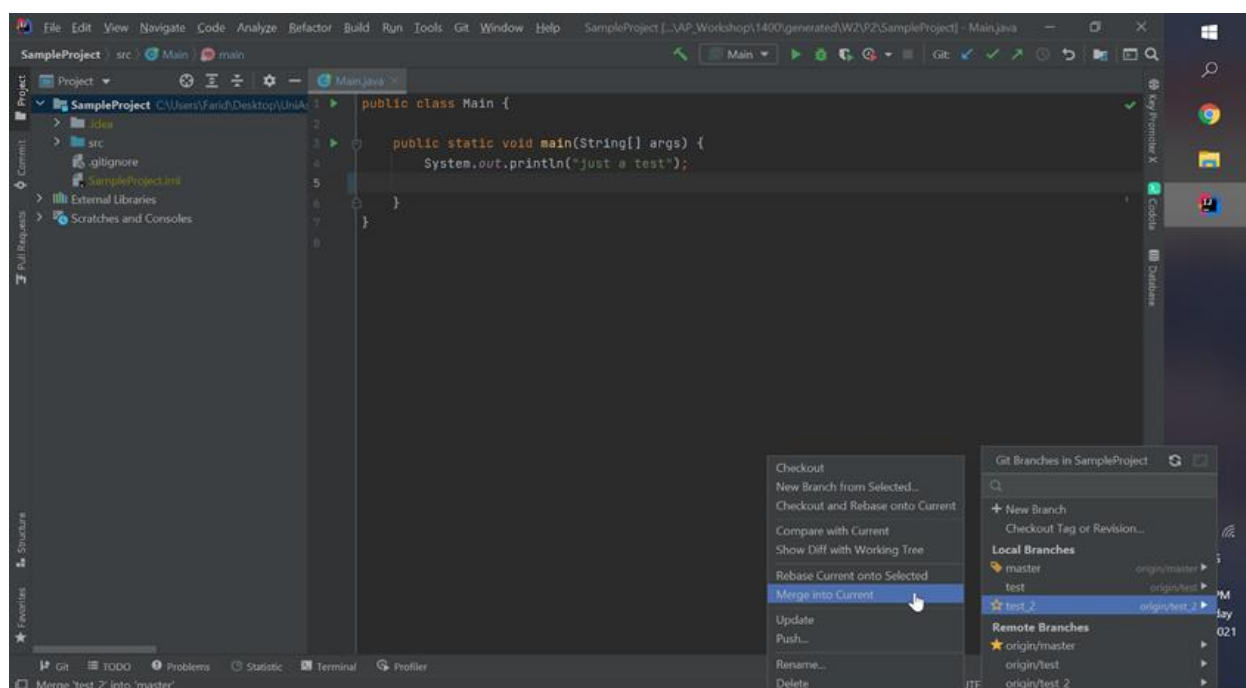
(نام شاخه جدید را نوشته و بر روی Create کلیک می‌کنیم)



(شاخه جدید را انتخاب کرده و روی آن کلیک می‌کنیم)



در نهایت می‌توانیم با checkout کردن بر روی شاخه master و مرج کردن شاخه قبلی، آن را با شاخه master مرج کنیم:



(انتخاب گزینه Merge into Current و مرج کردن شاخه‌ها)

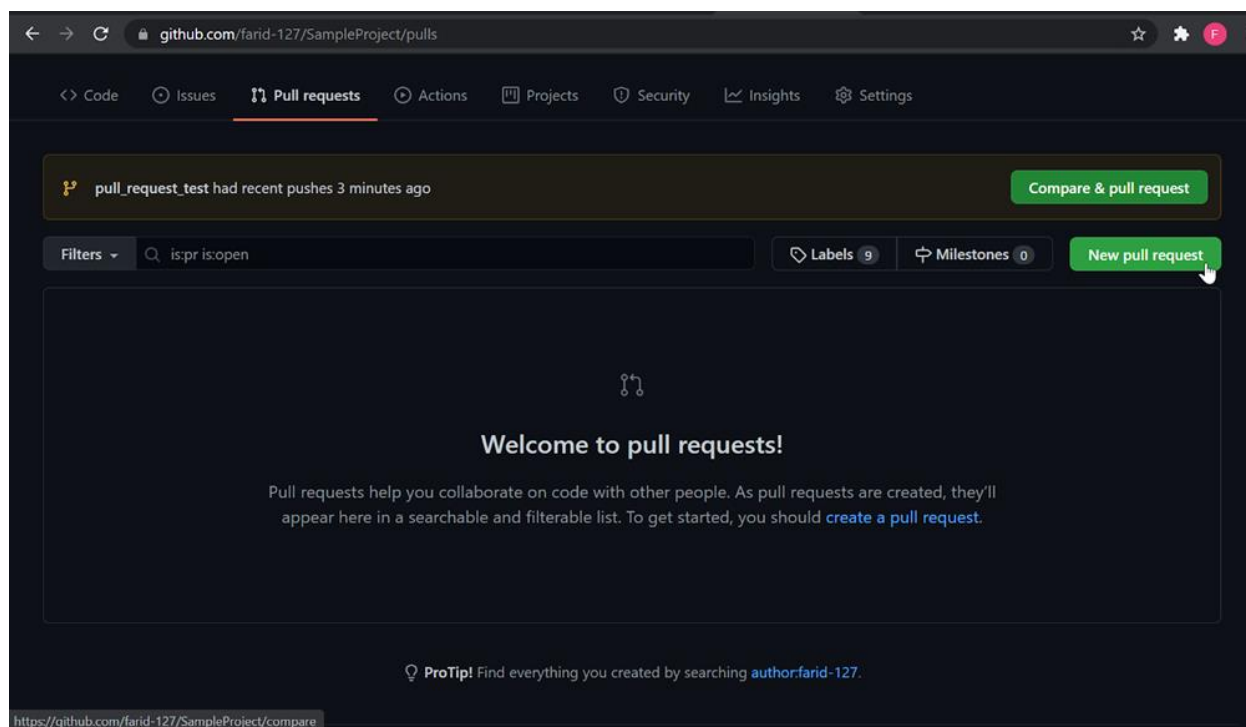


## آشنایی با pull request ها

فرض کنید در یک کار تیمی، هرکس پس از تمام شدن شاخه‌ای که روی آن کار می‌کرد، آن را روی شاخه master مرجع کند، بدون آنکه بقیه افراد تیم را در جریان بگذارد. این مسئله باعث ایجاد بی‌نظمی و عدم هماهنگی در روند پروژه می‌شود. برای اینکه چنین بی‌نظمی‌ای رخ ندهد، قبل از مرج کردن هر شاخه، می‌توان یک pull request ایجاد کرد. دیگر اعضای تیم می‌توانند پس از دیدن pull request، آن را تایید کنند.

با این حساب آیا pull request فقط برای کارهای تیمی کاربرد دارد؟  
خیر، توصیه می‌شود قبل از merge کردن از pull request استفاده کنید. با این کار نظم پروژه بیشتر خواهد شد و راحت‌تر می‌توان تغییرات پروژه را در هر مرحله مورد بررسی قرار داد.

## نحوه ایجاد pull request



(وارد تب pull requests می‌شویم)



## Compare changes

Compare changes across branches, commits, tags, and more below. If you need to, you can also [compare across forks](#).

base: master ← compare: master

Choose different branches

Choose a head ref

Find a branch

Branches Tags

✓ master default

pull\_request\_test

test

test\_2

Learn about pull requests

Create pull request

review just about anything

time ranges. In the same repository and across forks.

Example comparisons

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master ← compare: pull\_request\_test ✓ Able to merge. These branches can be automatically merged.

pull request test

Write Preview

H B I E < > @ ↶ ↷ ↵

just a pull request test for ap\_workshop

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Conversation 0 Commits 1 Checks 0 Files changed 1 +0 -0

farid-127 commented now

just a pull request test for ap\_workshop

pull request test commit e48a8fe

Add more commits by pushing to the pull\_request\_test branch on farid-127/SampleProject.

Continuous integration has not been set up

GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

✓ This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request You can also open this in GitHub Desktop or view command line instructions.

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

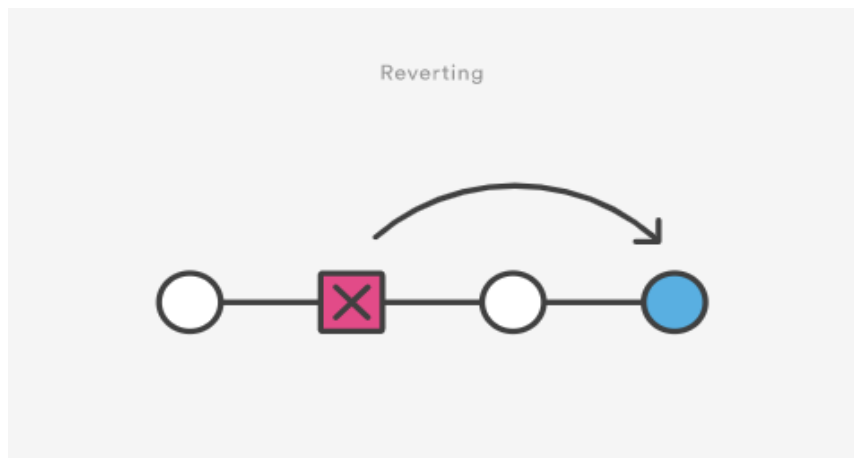
Linked issues



## دستورات تکمیلی گیت

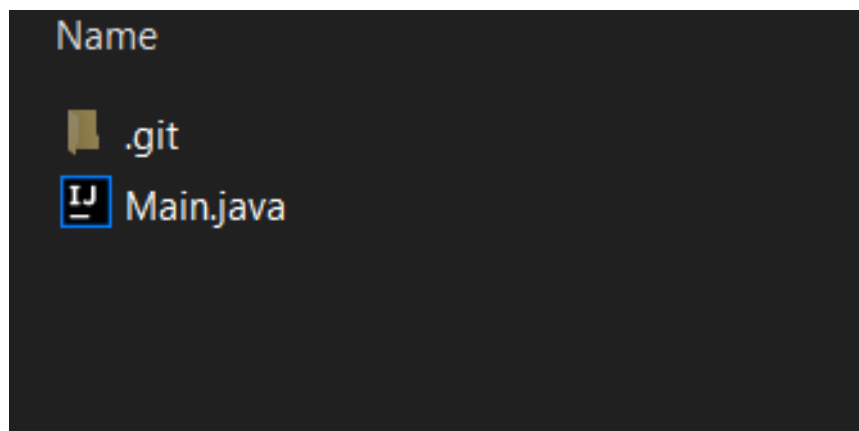
### دستور revert

با استفاده از دستور `git revert` می‌توان تغییرات ایجاد شده در یک کامیت مورد نظر را حذف کرد و یک کامیت جدید به وجود آورد:



(دستور `revert` یک کامیت را `undo` می‌کند)

فرض کنید یک مخزن ساده در اختیار داریم:



(ساختار مخزن)



یک فایل تکست را اضافه می‌کنیم:

```
ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/src/com/c
$ git commit -m "test.txt added"
[master edbc8fe] test.txt added
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.txt
```

(پس از اضافه کردن فایل، آن را کامیت می‌کنیم)

```
ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/
$ git log --oneline
edbc8fe (HEAD -> master) test.txt added
25ab88e (origin/master) first commit

ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/
```

(سابقه‌ی کامیت‌ها به شکل بالا خواهد بود)

- حال می‌خواهیم با دستور `revert`، فایل مورد نظر را حذف کنیم. برای این کار چندین حالت وجود دارد:
- از هاش کامیت<sup>1</sup> (رشته‌های زرد رنگ کنار هر کامیت) استفاده کنیم، برای مثال می‌توان نوشت:

```
git revert <hashcode>
```

- می‌دانیم که HEAD به آخرین کامیت اشاره می‌کند، در نتیجه با استفاده از دستور زیر می‌توان X کامیت به عقب بازگشت:

```
git revert HEAD~x
```

---

<sup>1</sup> hash commit



[illegible]

برای ویرایش کردن پیام کامیت، دکمه‌ی **افزودن** را فشار می‌دهیم تا وارد حالت افزودن<sup>۱</sup> شویم:

A screenshot of a Windows terminal window titled "MINGW64:/c/Users/ASUS/Desktop/workshopPlg/src/com/company". The terminal shows the command "Revert \"test.txt added\"" being executed. The output indicates that the commit edbc8fe6a90afb250d82db44943188b22aa7f3ed has been reverted. It prompts the user to enter a commit message, which is left blank. The status shows the branch is master and there are changes to be committed, specifically the deletion of test.txt. At the bottom, the file path workshopPlg/src/com/company/.git/COMMIT\_EDITMSG [unix] is shown along with a timestamp and a prompt to insert a commit message.

```
MINGW64:/c/Users/ASUS/Desktop/workshopPlg/src/com/company
Revert "test.txt added"

This reverts commit edbc8fe6a90afb250d82db44943188b22aa7f3ed.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   deleted:    test.txt
#
~
~
~
~
~
~
~
~
workshopPlg/src/com/company/.git/COMMIT_EDITMSG [unix] (00:19 15/08/2021)1,1 All
-- INSERT --
```

<sup>1</sup> Insert Mode





## آپشن amend

برای اعمال ویرایش روی آخرین کامیت، می‌توانیم از آپشن `--amend` در دستور `commit` استفاده کنیم. برای نمونه در مثال قبلی، فایل جدیدی می‌سازیم ولی کامیت جدید به وجود نمی‌آوریم:

```
ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/src/com/co
$ git add .

ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/src/com/co
$ git commit --amend -m "new text file added"
[master 4628ee7] new text file added
Date: Sun Aug 15 00:19:08 2021 +0430
1 file changed, 0 insertions(+), 0 deletions(-)
rename test.txt => new.txt (100%)

ASUS@ARIAN MINGW64 ~/Desktop/workshopPlg/src/com/co
$ git log --oneline
4628ee7 (HEAD -> master) new text file added
edbc8fe test.txt added
25ab88e (origin/master) first commit
```

(تغییرات روی آخرین کامیت اعمال می‌شوند)

## دستور reset

دستوری با هدف مشابه با `revert` و به منظور `undo` کردن تغییرات اعمال شده به کار می‌رود، با این تفاوت که دستور `revert` روش کم خطرتری می‌باشد و احتمال از بین رفتن دائمی تغییرات در دستور `reset` وجود دارد. دستور `reset` سه آپشن مهم دارد `--soft/hard/mixed` و مانند `revert` می‌توان از `HEAD~x` یا از همان هش کامیت استفاده کرد و به کامیت‌های قبلی بازگشت. فرض کنید ۳ کامیت به صورت زیر داریم:

```
-A-B-C(master)
```

در این حالت `HEAD` به کامیت `C` اشاره می‌کند.



با دستور **git reset --soft B** HEAD به کامیت B اشاره می‌کند ولی staging snapshot و working space دارای همان تغییرات کامیت C هستند.

دستور **git reset --mixed B** مانند آپشن soft باز هم HEAD به کامیت B اشاره می‌کند ولی با این تفاوت که staging snapshot هم تغییر می‌کند (working space تغییری نمی‌کند و برای ایجاد کامیت جدید باید تغییرات اعمال شده را با git add به حالت staged در آورد و سپس کامیت جدید را ایجاد کرد).

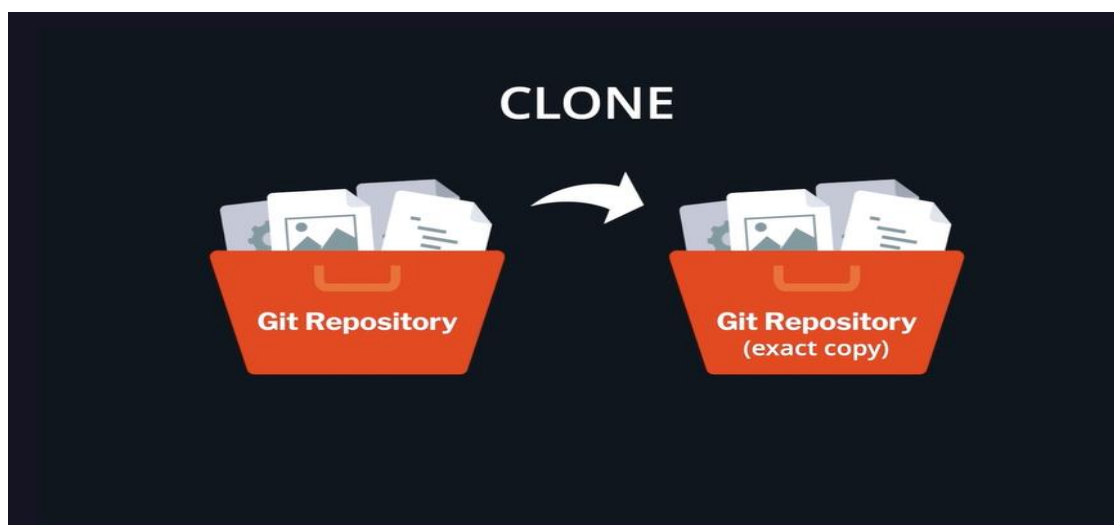
دستور **git reset --hard B** مانند آپشن mixed است، با این تفاوت که working space هم مطابق با کامیت B تغییر می‌کند و همین موضوع می‌تواند باعث از دست دادن تغییرات پیشین شود.

[مطالعه بیشتر درباره git reset](#)

[مطالعه بیشتر درباره تفاوت git revert و git reset](#)

## دستور clone

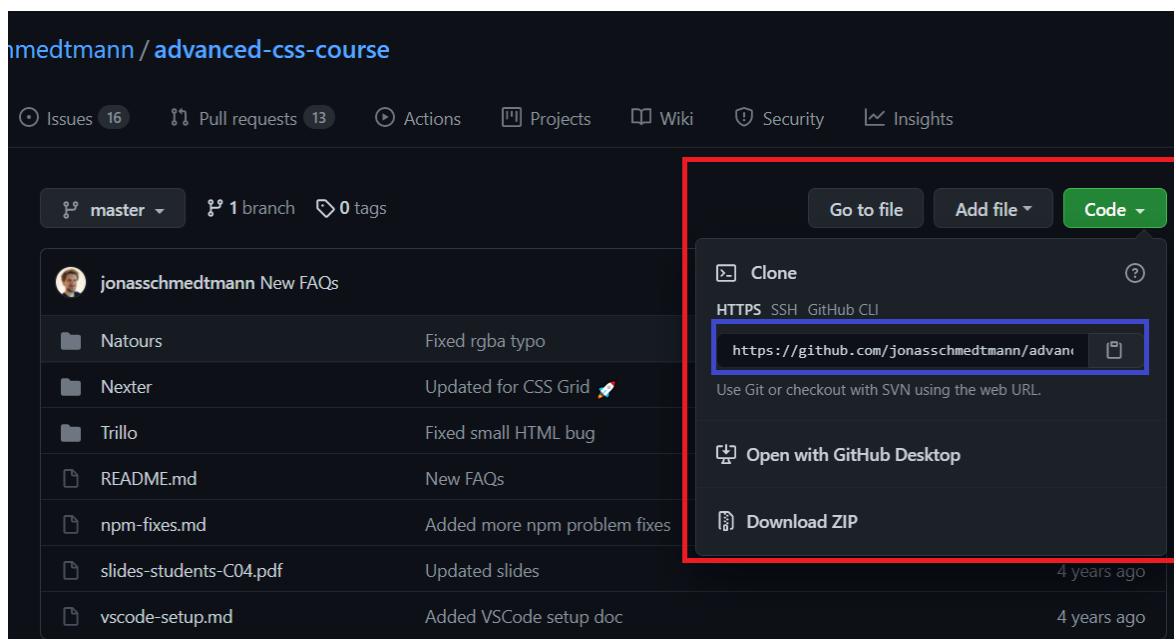
منظور از clone کردن یعنی داشتن آپدیت‌ترین نسخه از مخزن شخص دیگری با تمامی کامیت‌ها و تغییراتش در دایرکتوری مد نظر خودمان:



(کلون کردن مانند کپی کردن مخزن است)

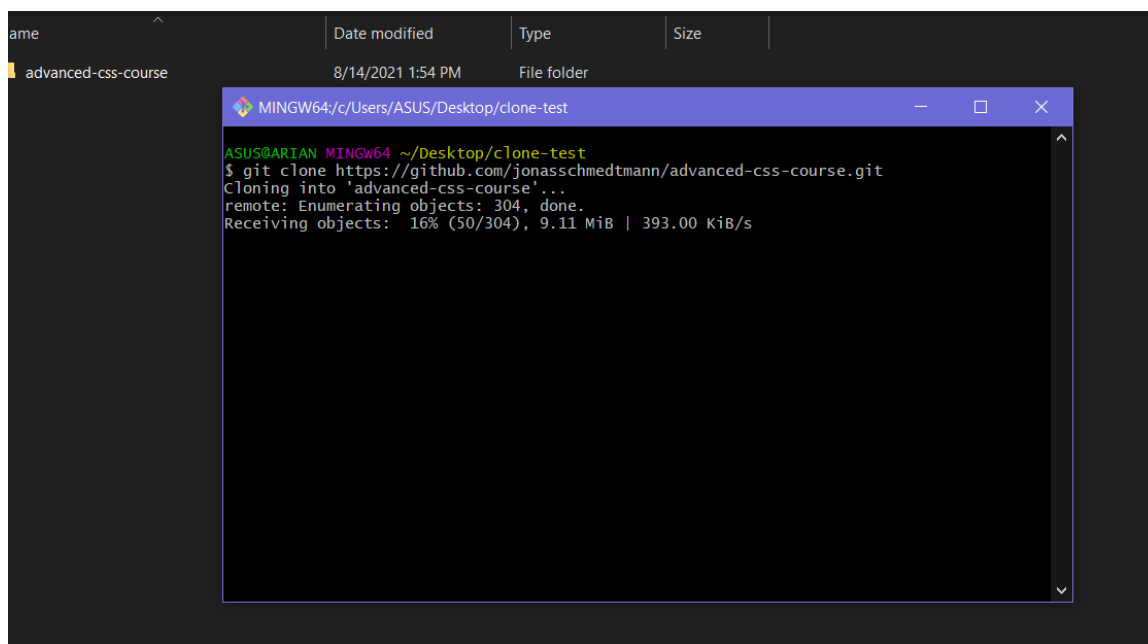


برای کلون کردن مخزن، ابتدا به صفحه‌ی مخزن مدنظر می‌رویم:



(روی Code کلیک کرده و HTTPS url را کپی می‌کنیم)

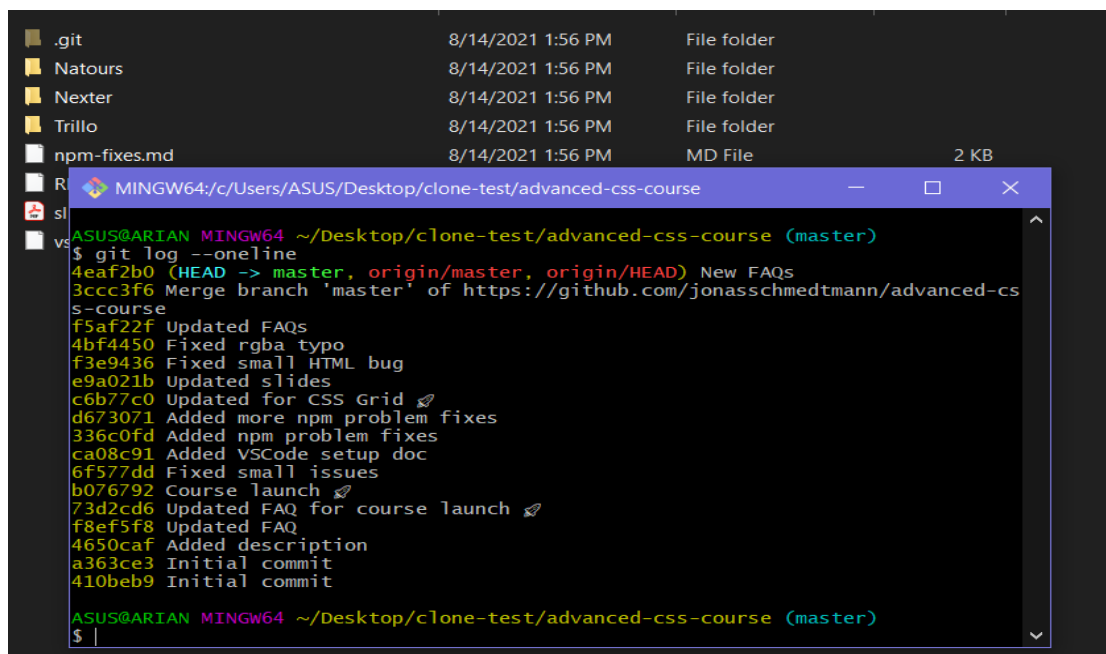
با استفاده از دستور `git clone [url]` یک کپی از مخزن ساخته می‌شود:



(در حال کلون کردن مخزن)



همانطور که گفته شد با عملیات کلون، یک کپی از مخزن ساخته می‌شود و با دستور `git log` می‌توان تاریخچه‌ی کامیت‌های سازنده‌ی اصلی مخزن را نیز مشاهده کرد:



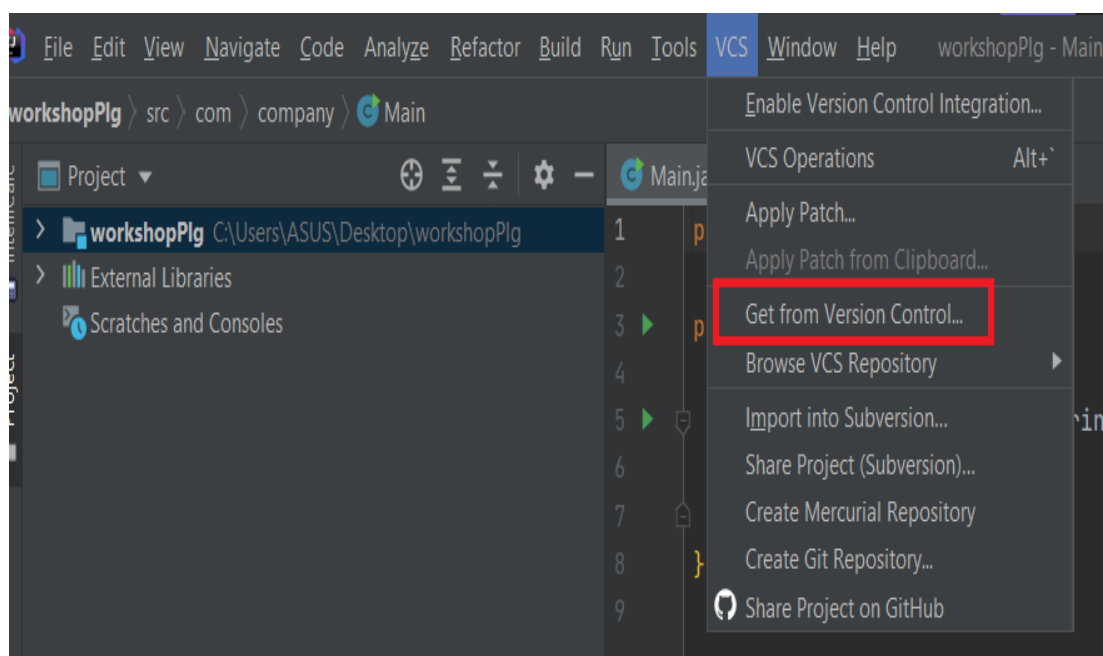
```
ASUS@ARIAN MINGW64 ~/Desktop/clone-test/advanced-css-course (master)
$ git log --oneline
4eaf2b0 (HEAD -> master, origin/master, origin/HEAD) New FAQs
3ccc3f6 Merge branch 'master' of https://github.com/jonasschmedtmann/advanced-css-course
f5af22f Updated FAQs
4bf4450 Fixed rgba typo
f3e9436 Fixed small HTML bug
e9a021b Updated slides
c6b77c0 Updated for CSS Grid
d673071 Added more npm problem fixes
336c0fd Added npm problem fixes
ca08c91 Added VSCode setup doc
6f577dd Fixed small issues
b076792 Course launch
73d2cd6 Updated FAQ for course launch
f8ef5f8 Updated FAQ
4650caf Added description
a363ce3 Initial commit
410beb9 Initial commit

ASUS@ARIAN MINGW64 ~/Desktop/clone-test/advanced-css-course (master)
$
```

(مشاهده سابقه کامیت‌های مخزن)

## کلون کردن مخزن در ایتلیجی

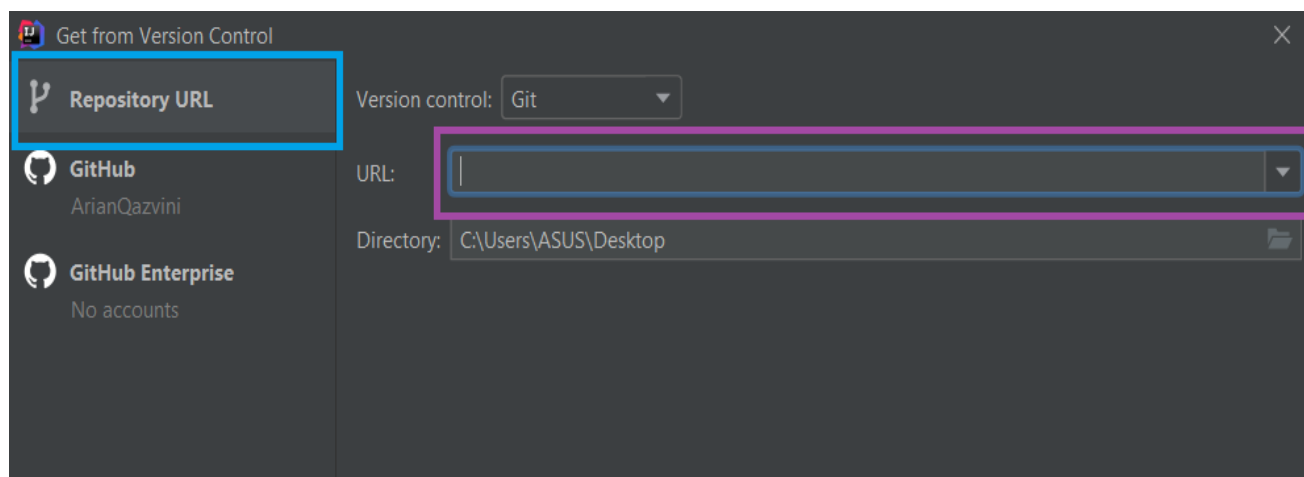
مطابق با مراحل زیر عمل می‌کنیم:



(از قسمت VCS، گزینه `get from version control` را انتخاب می‌کنیم)



سپس از منوی سمت چپ، Repository URL را انتخاب می‌کنیم و در قسمت URL، آدرس مخزن مورد نظر را قرار می‌دهیم همچنین در قسمت Directory، آدرسی که می‌خواهیم مخزن در آن کلون شود را مشخص می‌کنیم:



(کامل کردن فیلدهای خواسته شده)

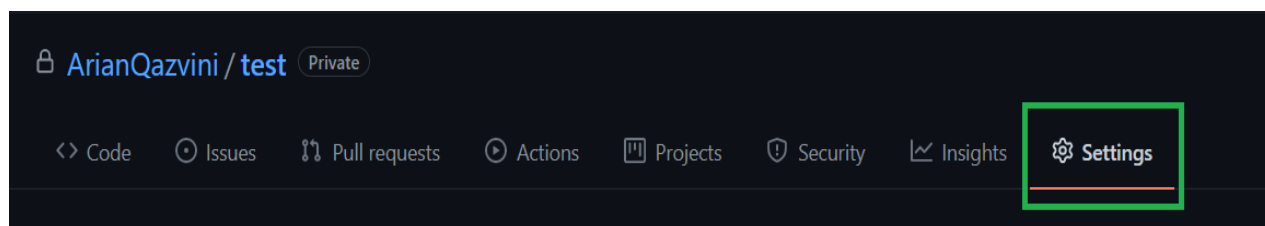
## Collaboration در گیت

فرض کنید می‌خواهیم با نفر دیگری بر روی یک پروژه به صورت مشترک کار کنیم. برای این منظور یک نفر باید مخزن شخصی بسازد و فرد دیگر، این مخزن را کلون کند و همچنین برای اینکه فرد دیگر اجازه‌ی اعمال تغییرات بر روی پروژه را داشته باشد، فرد سازنده‌ی مخزن باید به او اجازه دسترسی بدهد. در غیر این صورت کامیتهای او push نخواهند شد.



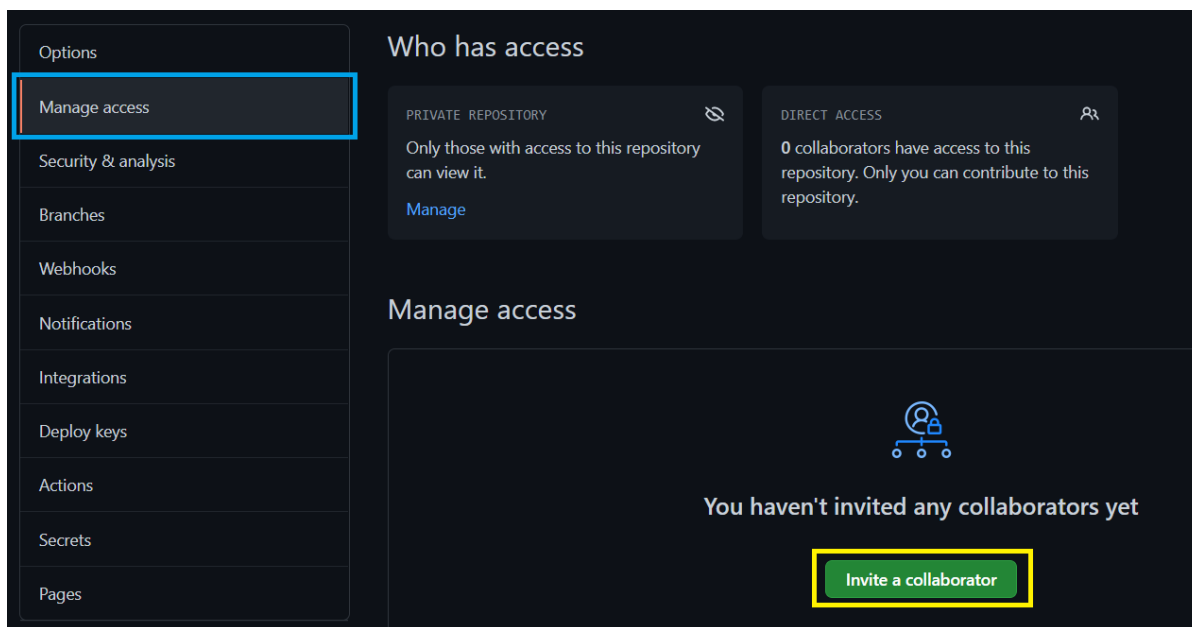
## نحوه دسترسی دادن در گیت‌هاب

پنجره Settings را از طریق صفحه مخزنمان باز می‌کنیم:



(انتخاب پنجره Settings از نوار بالای صفحه)

سپس از پنجره 'invite collaborator'، 'Manage Access' را انتخاب می‌کنیم و آیدی گیت‌هاب یا ایمیل فرد مورد نظرم را اضافه می‌کنیم:



(انتخاب گزینه 'Invite a collaborator' و اضافه کردن ایمیل فرد مورد نظر)





## مفاهیم کلاس و شیء

در این قسمت می‌خواهیم به توضیح مفاهیم شیء و کلاس در زبان جاوا بپردازیم. اول از همه باید بدانید که به طور کلی در جاوا یک کلاس از چهار بخش تشکیل شده است. می‌توانید بخش‌های مختلف یک کلاس را در تصویر زیر مشاهده کنید:

```
public class Main { // (1) Class definition

    // (2) Fields

    // (3) Constructor

    // (4) Methods
}
```

حال هر از یک این چهار بخش را به صورت کامل بررسی می‌کنیم:

۱. **کلاس<sup>۱</sup>** : در جاوا برای تعریف یک کلاس جدید، نام کلاس مورد نظر را بعد از کلمه‌ی کلیدی `class` قرار می‌دهیم. قبل از کلمه `class` می‌توانیم کلمه `public` را قرار دهیم. در صورتی که از کلمه‌ی `public` استفاده کنیم کلاس ما توسط هر کلاس دیگری قابل استفاده خواهد بود؛ در غیر این صورت تنها کلاس‌هایی که در پکیج یکسان با کلاس ما قرار دارند به این کلاس دسترسی خواهند داشت. در حال حاضر نیازی به دانستن مفهوم پکیج نیست و تنها کافیست از کلمه `public` قبل از کلمه `class` استفاده کنیم.

۲. **فیلدها<sup>۲</sup>** : هر کلاس می‌تواند شامل چندین فیلد باشد. فیلدها در واقع متغیرهایی هستند که نشان‌دهنده مشخصات مشترک نمونه‌های ساخته شده از یک کلاس می‌باشند.

عدد صحیح، عدد اعشاری و آرایه‌ها همگی نوع‌های مجاز برای تعریف فیلد هستند. تعریف یک فیلد درست مانند تعریف یک متغیر `local` است با این تفاوت که مانند تعریف کلاس باید سطح دسترسی به آن فیلد را هم مشخص کنیم.

---

<sup>۱</sup> Class

<sup>۲</sup> Fields



ساختار تعریف فیلد:

access modifier + type + name

```
public class Main {  
    private String name;  
    private String id;  
    private int[] grades;  
}
```

همان‌طور که می‌بینید، برای تمامی فیلدهای این کلاس سطح دسترسی پرایوت<sup>۱</sup> را در نظر گرفته‌ایم. این سطح دسترسی به این معناست که تنها اجزای این کلاس اجازه دسترسی به فیلدها را دارند و هیچ کلاس خارجی‌ای این اجازه را ندارد. همچنین می‌توانیم سطوح دسترسی public، protected و default را برای یک فیلد قرار دهیم. public به این معناست که تمام کلاس‌ها می‌توانند به این فیلد دسترسی پیدا کنند. با سطوح دسترسی default و protected در آینده آشنا خواهید شد.

معمولاً از سطح دسترسی پرایوت برای فیلدهای یک کلاس استفاده می‌کنیم تا به این طریق، از مقداردهی اشتباه و ناخواسته به این فیلدها جلوگیری کنیم. همچنین از متدهای گتر و ستر<sup>۲</sup> برای دسترسی و مقداردهی این فیلدها استفاده می‌کنیم که جلوتر با آن‌ها آشنا خواهیم شد.

**در نام‌گذاری فیلدها، به نکات زیر توجه کنید:**

۱. نام فیلد باید با حرف کوچک شروع شود.
۲. نام انتخابی را تا حد ممکن بامعنی و واضح انتخاب کنید و از عبارات رمزی و اختصاری خودداری کنید.
۳. در صورتی که نام انتخابی شامل بیش از یک کلمه بود، از ساختار camelCase استفاده کنید.
۴. استفاده از کامنت‌گذاری مناسب می‌تواند به فهم کارکرد هر فیلد کمک زیادی بکند.

<sup>۲</sup> Private

<sup>۲</sup> Getter & Setter



۳. **کانستراکتور**<sup>۱</sup>: نوعی رویه است که هنگام ایجاد شیء از یک کلاس صدا زده می‌شود و تمام عملیات لازم هنگام ایجاد شیء جدید از جمله مقداردهی فیلدها را انجام می‌دهد. دقت کنید که یک کانستراکتور می‌تواند ورودی‌هایی داشته باشد و از آن‌ها استفاده کند اما کانستراکتور بر خلاف متدها هیچ نوع خروجی‌ای ندارد و حتی نباید از کلمه کلیدی void نیز برای آن استفاده کرد. همچنین می‌توان سطح دسترسی یک کانستراکتور را مشخص کرد. اما معمولاً از سطح دسترسی public برای کانستراکتورها استفاده می‌شود.

**کانستراکتور پیش فرض:** هنگام ایجاد یک شیء، سازنده آن باید حتماً صدا زده شود. اما اگر سازنده‌ای برای کلاس مورد نظر وجود نداشته باشد چه اتفاقی می‌افتد؟ در این صورت جاوا به صورت خودکار یک سازنده‌ی خالی برای آن کلاس در نظر می‌گیرد که هیچ ورودی‌ای ندارد و از آن استفاده می‌کند.

```
public class Main {  
    private String name;  
    private String id;  
    private int[] grades;  
    public Main (String givenId, String givenName) {  
        id = givenId;  
        name = givenName;  
        grades = new int[10];  
    }  
}
```

---

<sup>1</sup> Constructor



شاید این سوال برایتان پیش بیاید که اگر نام ورودی‌های سازنده با نام فیلدها یکسان باشد چگونه می‌توان فیلدها را مقداردهی کرد. همان‌طور که می‌دانیم، ورودی‌های یک تابع، متغیرهای local آن تابع محسوب می‌شوند و به همین خاطر نمی‌توانیم با استفاده از نام فیلدها به آن‌ها دسترسی پیدا کنیم. کلمه کلیدی this برای اشاره کردن به شیء فعلی استفاده می‌شود. به این صورت که پس از این کلمه، یک علامت . و در انتها نام فیلد مورد نظر را وارد می‌کنیم تا به صورت مستقیم به فیلدهای کلاس دسترسی داشته باشیم:

```
public class Main {  
    private String name;  
    private String id;  
    private int[] grades;  
    public Main (String id, String name) {  
        this.id = id;  
        this.name = name;  
        this.grades = new int[10];  
    }  
}
```

**۴. متدها<sup>۱</sup>:** متدها، رویه‌های نشان‌دهنده رفتار یک شیء هستند. همان‌طور که فیلدها نشان‌دهنده خصوصیات و ویژگی‌های یک شیء بودند.

برای تعریف یک متد مانند تعریف تابع عمل می‌کنیم با این تفاوت که ابتدا مثل قبل، سطح دسترسی متد را مشخص می‌کنیم، سپس به ترتیب نوع خروجی متد و نام متد می‌نویسیم و در نهایت داخل پرانتز، نوع و نام ورودی‌های متد را مشخص می‌کنیم:

```
public void sayHello() {  
    System.out.println("Hello World!");  
}  
  
public int getSum(int a, int b) {  
    return a + b;  
}
```

---

<sup>1</sup> Methods



در نام‌گذاری متدها، به نکات زیر توجه کنید:

۱. درست مانند فیلدها، حرف اول را کوچک می‌نویسیم.
۲. نام متد باید کاملاً واضح و نشان‌دهنده کاری باشد که متد قرار است انجام دهد.
۳. هر متد باید دقیقاً یک کار را انجام دهد. اگر متوجه شدیم یک متد قابلیت تقسیم شدن به چند متد مجزا را دارد، باید این کار را انجام دهیم؛ در نتیجه می‌توانیم از این متد دفعات بیشتری استفاده کنیم.

**گتر و ستر:** همان‌طور که پیش‌تر گفتیم، در اکثر اوقات از سطح دسترسی private برای فیلدها استفاده می‌شود تا جلوی ایجاد تغییرات ناخواسته و مقداردهی نادرست گرفته شود. در این حالت از متد getter برای دسترسی به مقدار فیلد و از setter برای ایجاد تغییرات کنترل شده در مقدار فیلد استفاده می‌کنیم. فرض کنید کلاس ما دارای فیلد id می‌باشد و حداکثر طول مجاز برای آن، ۱۰ باشد. در این صورت متدهای گتر و ستر به این شکل خواهند بود:

```
public String getId() {  
    return id;  
}  
  
public void setId(String id) {  
    if (id.length() == 10)  
        this.id = id;  
}
```

**کپسوله سازی<sup>۱</sup>:** یکی از مهم‌ترین مفاهیم شیء‌گرایی در زبان‌های برنامه‌نویسی، کپسوله سازی است. در این روش، کد و داده بسته‌بندی شده و به صورت یک واحد منفرد در نظر گرفته می‌شوند. در کپسوله سازی متغیرهای یک کلاس از دید دیگر کلاس‌ها مخفی می‌شوند و تنها با استفاده از متدهای یک کلاس اجازه دسترسی به آن‌ها وجود خواهد داشت.

همان‌طور که دیدید، در جاوا فیلدهای یک کلاس را به صورت private تعریف می‌کنیم و تنها به وسیله متدهای گتر و ستر می‌توانیم به آن‌ها دسترسی داشته باشیم. در این روش، کلاس‌ها کنترل کامل داده‌های درون خود را خواهند داشت و هیچ کلاس دیگری اجازه ایجاد تغییرات ناخواسته در این داده‌ها را نخواهد داشت. به این روش، پنهان‌سازی داده<sup>۲</sup> هم گفته می‌شود.

<sup>۱</sup> Encapsulation

<sup>۲</sup> Data Hiding



## ایجاد شیء

**مرحله ۱)** برای ساختن شیء از یک کلاس ابتدا نام کلاس را مشخص می‌کنیم سپس نام شیء را طبق قواعد نامگذاری متغیرها تعیین می‌کنیم.

**مرحله ۲)** پس از قرار دادن علامت « = » از کلیدواژه‌ی new استفاده می‌کنیم. این کلمه مشخص می‌کند که قصد ایجاد یک شیء جدید و تخصیص حافظه به آن را داریم.

**مرحله ۳)** سپس دوباره نام کلاس مورد نظر را وارد می‌کنیم با این تفاوت که باید یک پرانتز باز و بسته پس از آن قرار دهیم. این پرانتزها مشخص می‌کنند که قصد استفاده از سازنده‌ی آن کلاس را داریم. در صورتی که سازنده‌ی کلاس، ورودی‌هایی داشته باشد، باید آن ورودی‌ها را داخل پرانتز وارد کنیم.

کلاسی به نام Person در نظر بگیرید که فیلدهای آن عبارت اند از نام و سن شخص. حال قطعه کد زیر را در نظر گرفته و خروجی آن را حدس بزنید:

```
Person p1 = new Person("Ali", 24);
Person p2 = new Person("Mahdi", 34);

String p1Name = p1.getName();
System.out.println(p1Name);

String p2Name = p2.getName();
System.out.println(p2.getName());
```

شاید این سوال برایتان پیش بیاید که هنگامی که یک شیء را در متغیری ذخیره می‌کنیم، یا آن را به عنوان ورودی به یک تابع پاس می‌دهیم، جاوا از روش call by value یا call by reference استفاده می‌کند؟ در جواب به این سوال باید بگوییم که در درس‌های آینده، هنگام آشنایی با مدل حافظه در جاوا به خوبی این موضوع را درک خواهید کرد اما به طور کلی جاوا هنگام ذخیره‌ی اشیاء از روش call by reference استفاده می‌کند. به عنوان مثال متغیر p1Name در مثال بالا تنها یک اشاره‌گر به یک شیء از نوع رشته<sup>۱</sup> است. هر چند باید به این نکته توجه کرد که در جاوا امکان دسترسی مستقیم به آدرس اشیاء را نداریم و تنها می‌توانیم از مقدار آن‌ها استفاده کنیم.

---

<sup>۱</sup> String



## میانبرها در IntelliJ

خیلی اوقات نه تنها نیاز داریم که تمرکزمان را از روی تایپ کردن برداریم، بلکه می‌خواهیم با تایپ کردن کمتر، کد بیشتری بنویسیم. میانبرها<sup>1</sup> این امکان را فراهم می‌کنند. برخی از میانبرهای کاربردی در نرم‌افزار IntelliJ در زیر آمده‌اند:

• **Ctrl + V و Ctrl + X، Ctrl + C**

ابتدایی‌ترین میانبرهایی هستند که در خارج از محیط برنامه‌نویسی نیز قابل استفاده‌اند. به ترتیب از چپ به راست برای cut، paste و copy فایل، کلمه یا جمله مورد استفاده قرار می‌گیرند.

• **Ctrl + Shift + (left/right arrow key)**

با استفاده از این میانبر می‌توان کلمه به کلمه در جمله پیمایش کرد و آن‌ها را انتخاب کرد.

• **Ctrl + /**

با استفاده از این میانبر می‌شود خطوط انتخابی یا خطی که موس بر روی آن قرار دارد را کامنت کرد.

• **Ctrl + Shift + /**

با استفاده از این میانبر می‌توان یک بلوک برای کامنت کردن تعداد خطوط دلخواه انتخاب کرد.

• **Ctrl + D**

با استفاده از این میانبر می‌توان خطوط انتخابی یا خطی که موس بر روی آن قرار دارد را در خط بعدی کپی و پیست کرد.

• **Ctrl + (+/-)**

با استفاده از این میانبر می‌توان داخلی‌ترین بلاک را باز (+) یا بسته (-) کرد.

• **Ctrl + (up/down arrow key)**

با استفاده از این میانبر می‌توان خطوط انتخابی را در بین دیگر خطوط جابجا کرد.

• **Alt + J**

---

<sup>1</sup> Shortcuts



با قرار دادن موس بر روی کلمه دلخواه، می‌توان کلمات هم‌نام را به تعداد دلخواه انتخاب کرد و آن‌ها را به طور همزمان تغییر داد. برای از بین بردن حالت چند نشانگر بوجود آمده در صفحه، می‌توان از میانبر **Alt + Shift + J** استفاده کرد.

• **Ctrl + Q**

با قرار دادن موس بر روی اسم یک تابع و استفاده از این میانبر می‌توان اطلاعات متد، نظیر پارامترهای ورودی و جاوادلک و پکیج متد را مشاهده کرد.

• **Ctrl + Shift + I**

با قرار دادن موس بر روی اسم یک تابع و استفاده از این میانبر می‌توان به نحوه پیاده‌سازی دقیق یک متد، پی برد.

• **Ctrl + Alt + L**

با استفاده از این متد می‌توان تمامی کد نوشته شده در هر صفحه را مرتب کرد.

• **(on laptop: Fn + F2) F2**

با استفاده از این میانبر می‌توان به نزدیک‌ترین ارور بعد از مکان کنونی موس رفت.

• **Ctrl + F1**

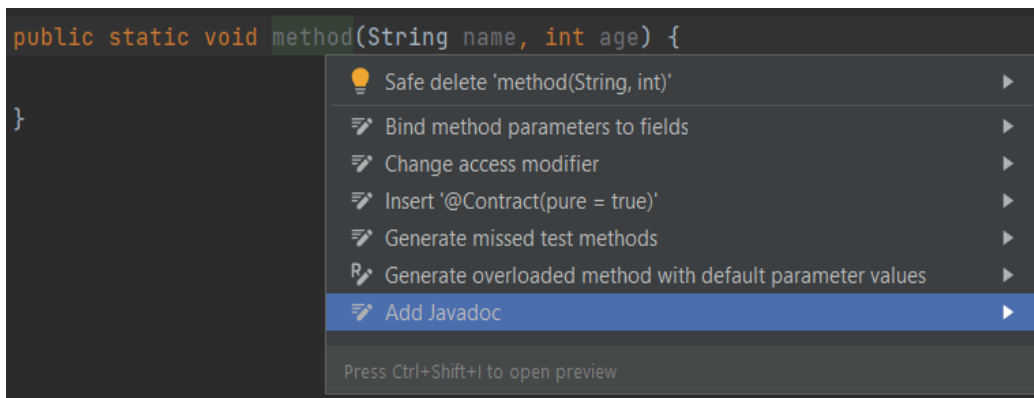
بعد از انتقال موس به محل ارور می‌توان با این میانبر علت ارور را مشاهده کرد.

• **Alt + Enter**





با قرار دادن موس بر روی یک ارور و با استفاده از این میانبر می‌توان راه پیشنهادی برای از بین بردن این ارور را مشاهده کرد همچنین با قرار دادن موس بر روی اسم متد در محل تعریف آن و استفاده از این میانبر می‌توان برای متد جاواداک به وجود آورد:



(ساخت جاواداک برای متد)

- **Alt + (left/right arrow keys)**

با استفاده از این میانبر می‌توان بین کلاس‌های مختلف که در بالای صفحه چیده شده‌اند، جابه‌جا شد.

- **Ctrl + Tab**

با این میانبر می‌توان بین اکثر قسمت‌های محیط کار و کلاس‌های تعریف‌شده، جابه‌جا شد.

- **Ctrl + B**

با قرار دادن موس بر روی تعریف یک اسم و استفاده از این میانبر، تمامی کاربردهای آن لیست شده و به هر کدام که مدنظر است می‌توان منتقل شد.

- **Alt + Insert**

با استفاده از این میانبر می‌توان طیف وسیعی از متدهای مورد نیاز، کانستراکتور و ... را بصورت خودکار تولید کرد (بیشتر در تعریف متد برای کلاس‌های جدید کاربرد دارد).



#### • Live templates

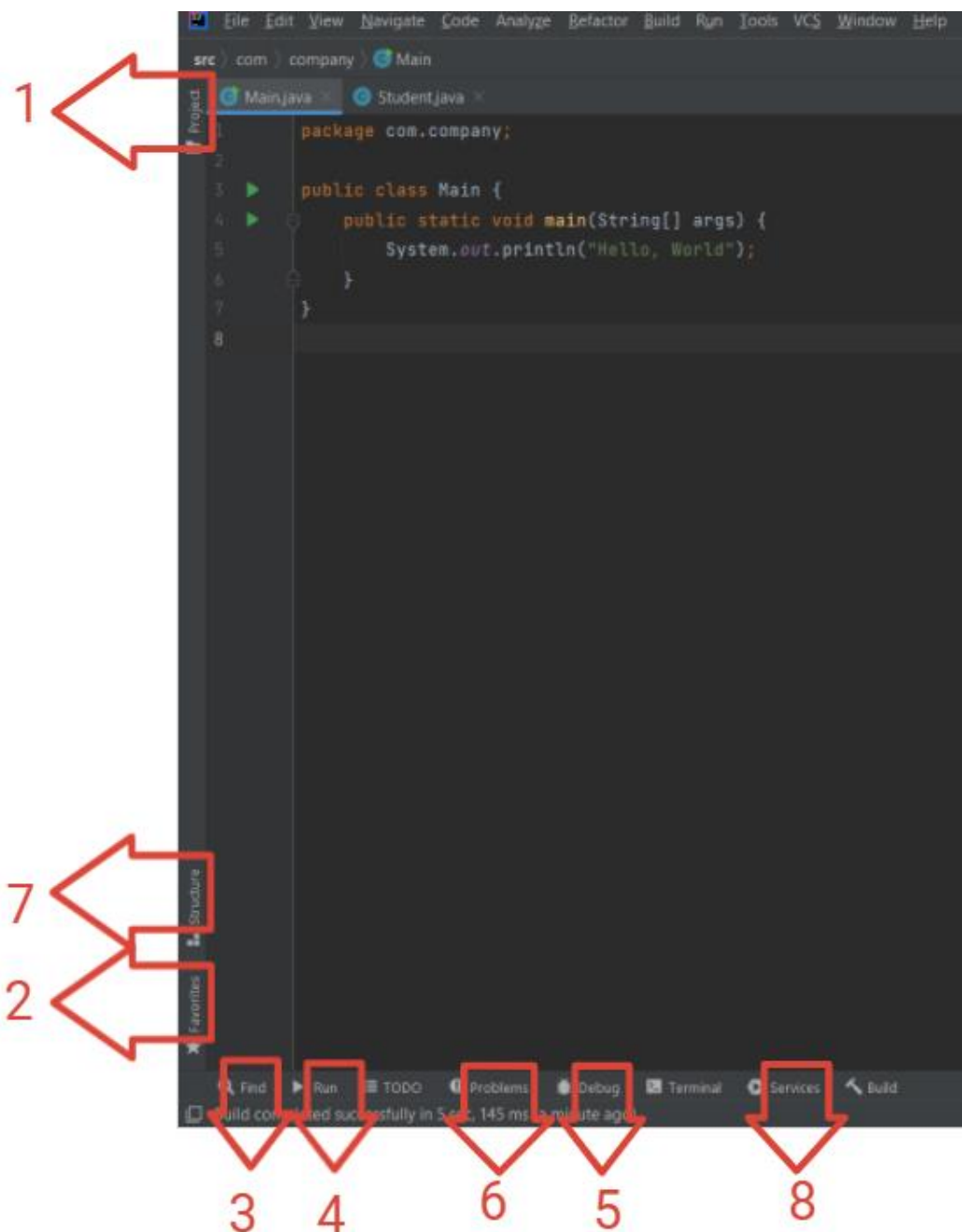
از دیگر میانبرهایی هستند که با نوشتن آن‌ها و فشردن Enter می‌توان ادامه آن‌ها را بصورت خودکار کامل کرد (در مکان استفاده از آنها مطابق با سینتکس جاوا باید عمل کرد):

- **sout:** `System.out.println( );`
- **souf:** `System.out.printf(" ");`
- **main/psvm:** `public static void main(String[ ] args){ }`
- **psi:** `public static final`
- **ifn:** `if(args == null){ }`
- **fori:** `for(int i = 0; i < ; i++){ }`



• **Alt + (number)**

با استفاده از این میانبر می‌توان به اکثر قسمت‌های محیط کار دسترسی پیدا کرد و یا آنها را بست:





## انجام دهید

در این قسمت می‌خواهیم به کمک مفاهیم کلاس و شی، یک کارگاه برنامه‌نویسی پیشرفته را شبیه‌سازی کنیم که از تعدادی دانشجو تشکیل شده است. ابتدا یک ریپازیتوری با نام «AP-Workshop2» ایجاد کنید. حال، به ترتیب مراحل زیر را انجام دهید:

### ساخت کلاس Student

```
public class Student {  
  
    /* add fields ... */  
  
    /* add a constructor ... */  
  
    /* add some methods ... */  
  
}
```

(کلاس Student)

– به ویژگی‌های یک دانشجو فکر کنید و آن‌ها را مانند شکل زیر به عنوان فیلد به این کلاس اضافه کنید (فرض کنید که id یا شماره دانشجویی، باید شامل دقیقا ۷ رقم باشد):  
مشاهده می‌کنید که برای رعایت کپسوله سازی، سطح دسترسی تمام فیلدها private تعریف شده است.

```
public class Student {  
  
    private String firstname;  
  
    private String lastname;  
  
    private String id;  
  
    private double grade;  
  
}
```

(فیلدهای کلاس Student)



- در این مرحله باید کانستراکتور کلاس را تشکیل دهیم:

```
public class Student {  
  
    private String firstname;  
  
    private String lastname;  
  
    // student's id contains at least 7 digits, e.g: 9931078  
    private String id;  
  
    private double grade;  
  
    public Student(String firstname, String lastname, String id) {  
        this.firstname = firstname;  
        this.lastname = lastname;  
        this.id = id;  
  
        // assuming that the student's grade is zero  
        grade = 0;  
    }  
  
}
```

(کانستراکتور کلاس Student)

همان طور که مشاهده می‌کنید، بعضی از فیلدها را می‌توان با پارامتر ورودی کانستراکتور مقداردهی کرد و برای بقیه، مقداری پیش‌فرض در نظر گرفت (مانند grade در شکل بالا).

**به کامنت‌گذاری‌ها دقت کنید:** برای توضیح بیشتر و خوانایی کد، حتماً کامنت‌گذاری مناسب را رعایت کنید. کامنت‌ها نه تنها به برنامه‌نویسانی که قرار است کدتان را بررسی کنند کمک می‌کند، بلکه برای خودتان هم لازم است، تا دلیل وجود هر تکه کد را در صورت فراموشی، به خاطر بیاورید.

- در ادامه، متدهای گتر و ستر تمام فیلدها را در صورت نیاز اضافه کنید (فراموش نکنید که نام هر متد یا متغیر را به صورت camelCase بنویسید). در سترها محدودیت مقدار متغیر مربوطه را حتماً در نظر بگیرید و برای ورودی‌های نامعتبر پیام مناسبی به کاربر نشان دهید.



- متد `printStudentInfo` را به این کلاس اضافه کنید:

```
public void printStudentInfo() {  
    System.out.println(firstname + " " + lastname  
        + "\nID: " + id + "\nGRADE: " + grade);  
}
```

(پیاده‌سازی متد `printStudentInfo` در کلاس `Student`)

- در مرحله‌ی آخر، یک کلاس `Main` که شامل متد `main` می‌باشد، مانند شکل زیر تشکیل دهید و کلاس `Student` را تست کنید:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Student std1 = new Student( firstname: "James", lastname: "Gosling", id: "0987654");  
        Student std2 = new Student( firstname: "Dennis", lastname: "Ritchie", id: "1234567");  
  
        std1.setGrade(18);  
        std2.setGrade(17.5);  
  
        std1.printStudentInfo();  
        std2.printStudentInfo();  
  
    }  
  
}
```

(پیاده‌سازی کلاس `Main` برای تست کردن کلاس `Student`)

- کدتان را با یک پیام مناسب کامیت کنید.



## ساخت کلاس Lab

- فیلدها و کانستراکتور را مانند شکل زیر به این کلاس اضافه کنید:

```
public class Lab {  
  
    private Student[] students;  
    private String teacherName;  
    // the day in which this lab is held  
    private String dayOfWeek;  
    private int maxSize;  
    private int currentSize;  
    private double avgGrade;  
  
    public Lab(String teacherName, String dayOfWeek, int maxSize) {  
        this.teacherName = teacherName;  
        this.dayOfWeek = dayOfWeek;  
        this.maxSize = maxSize;  
        // create an array of students with the size of "maxSize"  
        students = new Student[maxSize];  
    }  
}
```

(فیلدها و کانستراکتور کلاس Lab)

به روش ساخت آرایه در کانستراکتور توجه کنید. با نوشتن این خط کد، یک آرایه از جنس Student با سایز مشخص شده، در حافظه ساخته می‌شود و پس از آن می‌توانید به این آرایه، اشیاء کلاس Student را اضافه کنید.



- در آخر، متدهای زیر را ایجاد کنید. کامنت‌گذاری مناسب را رعایت کرده و در صورت نیاز، پیام مناسبی به کاربر نمایش دهید:

Lab		
f	students	Student[]
f	teacherName	String
f	dayOfWeek	String
f	maxSize	int
f	currentSize	int
f	avgGrade	double
m	Lab(String, String, int)	
m	calAvg()	void
m	enrollStudent(Student)	void
m	getAvgGrade()	double
m	getCurrentSize()	int
m	getDayOfWeek()	String
m	getMaxSize()	int
m	getStudents()	Student[]
m	getTeacherName()	String
m	printLabInfo()	void
m	setAvgGrade(double)	void
m	setCurrentSize(int)	void
m	setDayOfWeek(String)	void
m	setMaxSize(int)	void
m	setStudents(Student[])	void
m	setTeacherName(String)	void

(ساختار کلاس Lab)





## توضیحات:

متد `printLabInfo` باید تمام اطلاعات کارگاه و دانشجویان آن را در کنسول چاپ کند.

پس از کامل کردن این کلاس، آن را مانند مرحله قبل در کلاس `Main` تست کنید.

- دو دانشجو با اطلاعات زیر بسازید:

```
James Gosling - ID: 0987654 - grade: 18  
Dennis Richie - ID: 1234567 - grade: 17.5
```

- حال متد `printStudentInfo` را روی این دانشجویان صدا بزنید.

- سپس یک شیء از کلاس `Lab` با اطلاعات زیر ساخته و دانشجویان را به آن اضافه کنید:

```
teacherName: Mr.Smith  
dayOfWeek: Monday  
maxSize: 30
```

در نهایت، متد `printLabInfo` را صدا بزنید.

تغییرات اعمال شده را کامیت کرده و در نهایت تمام کامیت‌های خود را `push` کنید.