CS-319 SPRING 2020 LABORATORY ASSIGNMENT

Prerequisites

- SourceTree, GitHub and Git softwares.
- GitHub Account

Assume that you are working in a software company and your team leader comes and tells you that she needs an ***ArrayOrganizer*** class and in the class, there has to be a method present which concatenates two ArrayList's and sorts them. After retrieving this task, you create a new project and inside this project, you create a class named ***ArrayOrganizer*** and write the desired method whose name is ***concatAndSort*.** After completing the class, your class looks like this:

```
import java.util.ArrayList;
import java.util.Collections;

public class ArrayOrganizer {

    public ArrayList<Integer> concatAndSort(ArrayList<Integer> list1, ArrayList<Integer> list2) {
        ArrayList<Integer> result = new ArrayList<>();
        result.addAll(list1);
        result.addAll(list2);
        Collections.sort(result);
        return result;
    }

}
```

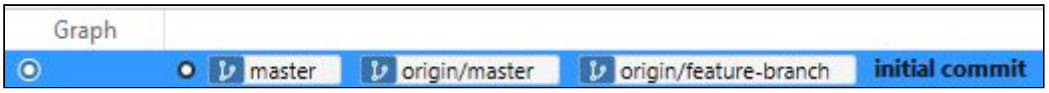
Then, you decide to initialize the git software in this project and push the project into a remote repository. After doing these operations, your commit history is looking as the following:

You have a ***single commit***, a ***local master*** branch and a ***remote master*** branch. The origin keyword indicates that it is a remote branch which is taught in the git lectures.

After doing these steps, your team leader views your code from GitHub and comes to you and says that *'I wanted to have them sorted in **descending** order, not in ascending order. You need to fix this bug."*

In parallel, the team leader goes to another developer and asks her for a new feature. What she desires is that in the same method, instead of having two lists concatenated, three lists needs to be concatenated.

In this case, another colleague receives the new feature task and she opens a new branch whose name is ***'feature-branch'***. After opening the new branch, the updated commit history (which is obtained after fetch operation) looks like this:



You still have a ***single commit***, a ***local master*** branch, a ***remote master*** branch and the newly added ***remote feature*** branch. The other developer develops the new feature and updates the ***ArrayOrganizer*** class as the following one:

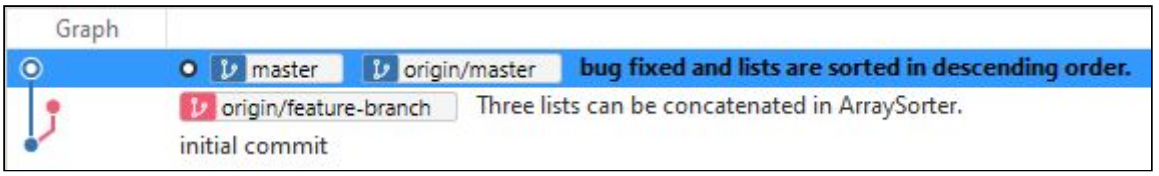
```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayOrganizer {

    public ArrayList<Integer> concatAndSort(ArrayList<Integer> list1, ArrayList<Integer> list2, ArrayList<Integer> list3) {
        ArrayList<Integer> result = new ArrayList<>();
        result.addAll(list1);
        result.addAll(list2);
        result.addAll(list3);
        Collections.sort(result);
        return result;
    }

}
```

After she completes the implementation, she pushes the changes to the remote repository. Simultaneously, you, as the bug fixer perform a fetch operation in your local repository and see the updated commit history as the following:

You start to fix the bug and sort the arrays in ***descending*** order in your ***master*** branch and the bug fixed code looks as below:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayOrganizer {

    public ArrayList<Integer> concatAndSort(ArrayList<Integer> list1, ArrayList<Integer> list2) {
        ArrayList<Integer> result = new ArrayList<>();
        result.addAll(list1);
        result.addAll(list2);
        Collections.sort(result, Collections.reverseOrder());
        return result;
    }

}
```

Then, you also make a commit for the bug fix and push the changes to the remote repository and the commit history becomes like the following:



In your next daily meeting, you and your colleague say to your team leader that the given tasks are completed. Now, it is turn to merge these changes. The operation that needs to be done is ***merging feature branch into master branch***. The team leader asks you to make the merge operation.

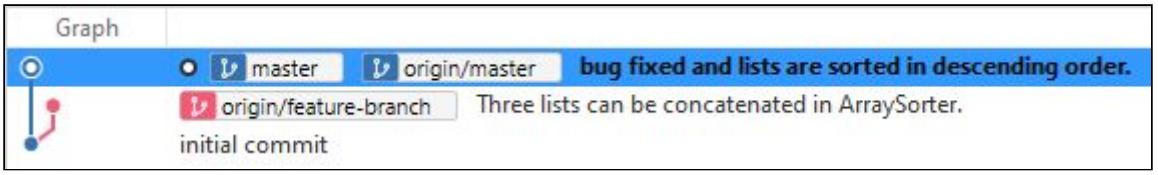
40 pts(forking, resolving conflicts and merging)

Up to this point, the commit history of the explained scenario is already present in the following remote repository:

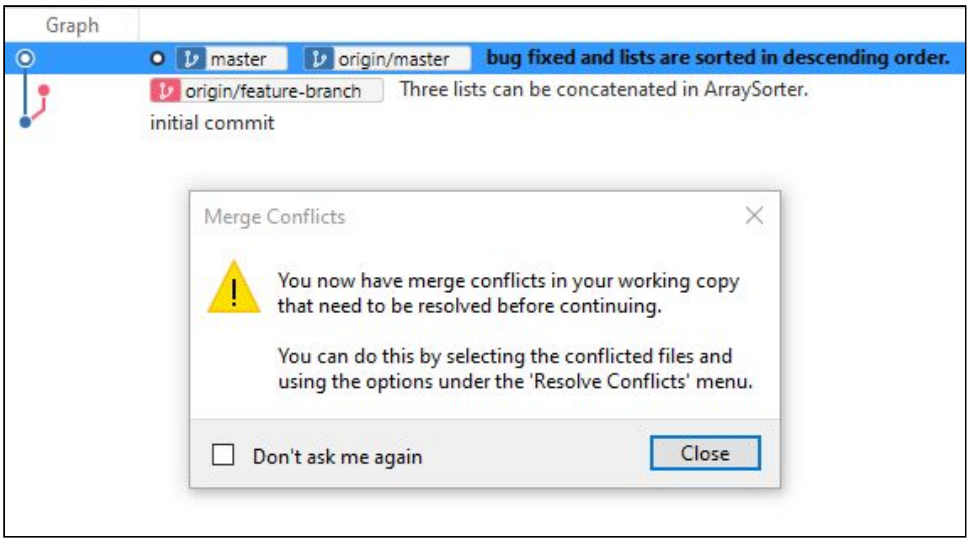
https://github.com/FatihSevbanUyanik/CS319_Lab_Assignment

You need to fork this repository to your account. A ***fork*** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, ***forks*** are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. Forking can be achieved by clicking the following button on the indicated repository above:

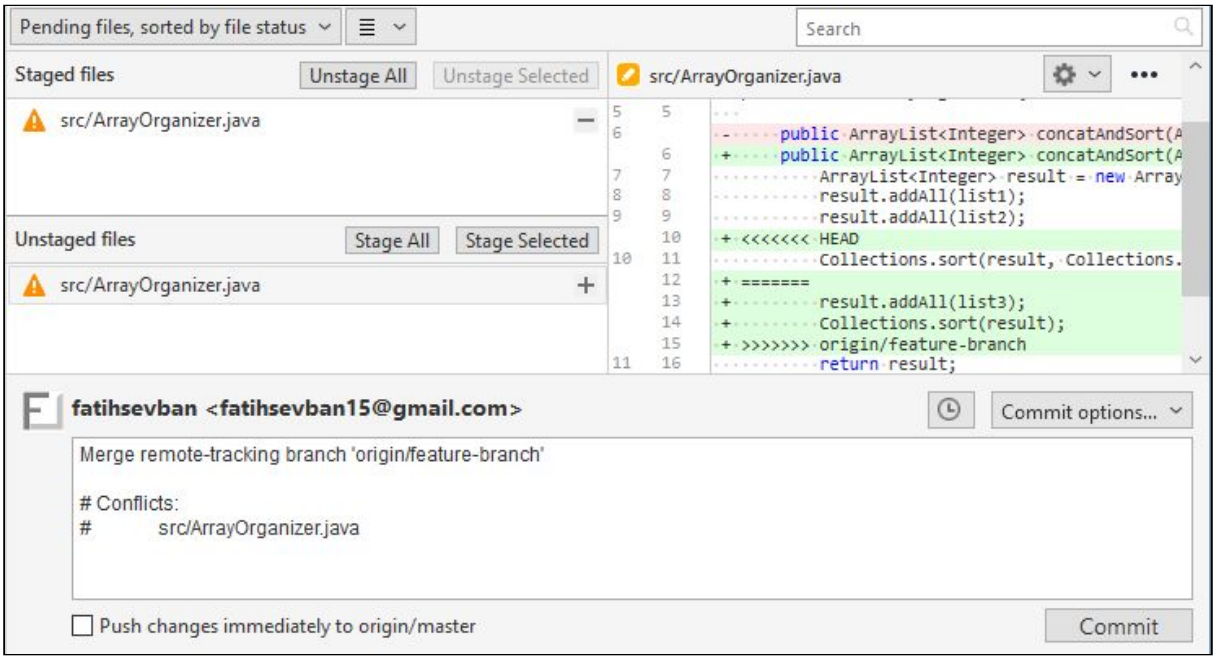
As soon as you hit the button, the project will be forked automatically to your profile. Now, you need to clone the repository to your local pc and get exactly a commit history as the following:



After, as your team leader desires, you need to perform the merge operation. Checkout to master branch and hit the merge button. In the opening dialog, you need to ***merge the remote feature-branch into master branch.*** While merging, since you and your colleague have changed the ***same*** places in the code, you should get a ***conflict***.



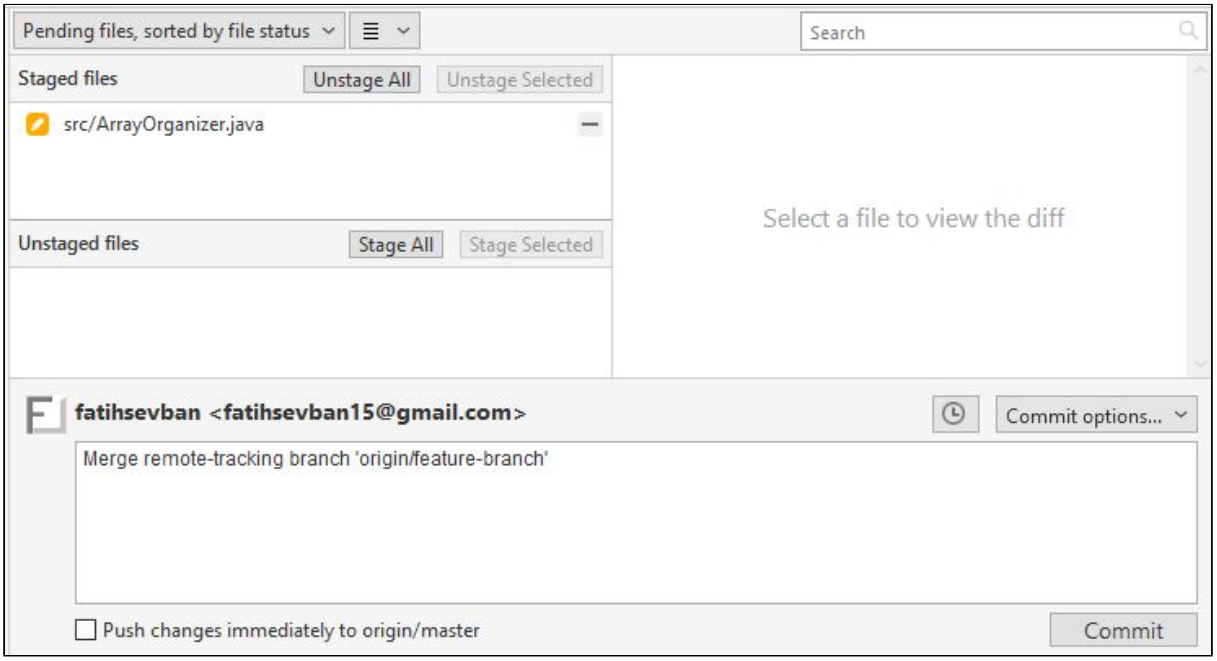
When you come to the ***file status***, you will see the file in which conflict has occurred. In our case, it is the ***ArrayOrganizer*** class.



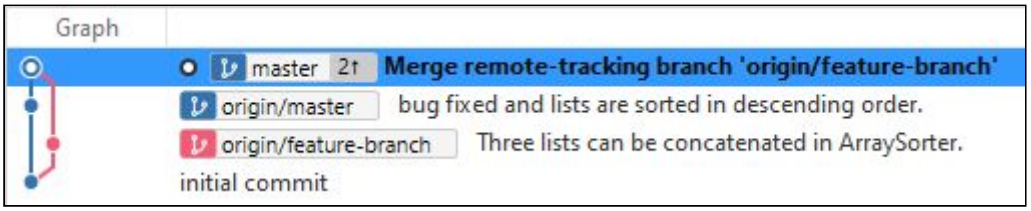
Find the corresponding file in your project and open it with ***VS code***. Also, if you want, you can do it directly fix it from your ***IDE*** as well. However, doing it with ***VS code*** is more convenient because it shows you many options. When you open the file with ***VS code***, you should get the following result:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayOrganizer {

    public ArrayList<Integer> concatAndSort(ArrayList<Integer> list1, ArrayList<Integer> list2, ArrayList<Integer> list3) {
        ArrayList<Integer> result = new ArrayList<>();
        result.addAll(list1);
        result.addAll(list2);
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
        Collections.sort(result, Collections.reverseOrder());
=======
        result.addAll(list3);
        Collections.sort(result);
>>>>>>> origin/feature-branch (Incoming Change)
        return result;
    }

}
```

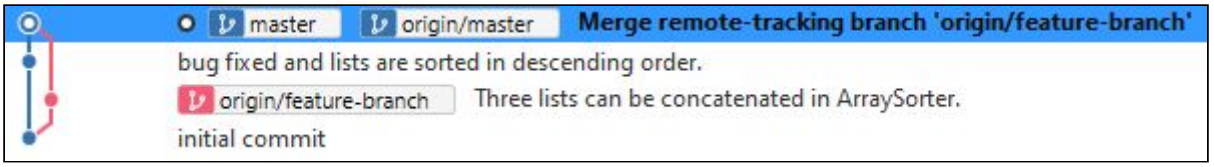
How you do is not important, but you need to resolve this ***conflict*** and the ***concatAndSort*** method needs to be able to concat these three arrays and sort them in ***descending*** order. After you resolved this issue, save the changes and come back to ***SourceTree***. Stage the last changes and remove the conflicted files in the commit message. After doing these, your SourceTree needs to look like the following before the commit:



If you are in this state, hit the commit button and open the commit history. It is supposed to look like the following:



As you see, your ***remote master*** branch is two steps behind your ***local master*** branch. Hit the push button and ***push*** the changes to the ***remote master*** branch. After pushing, you should have the following:



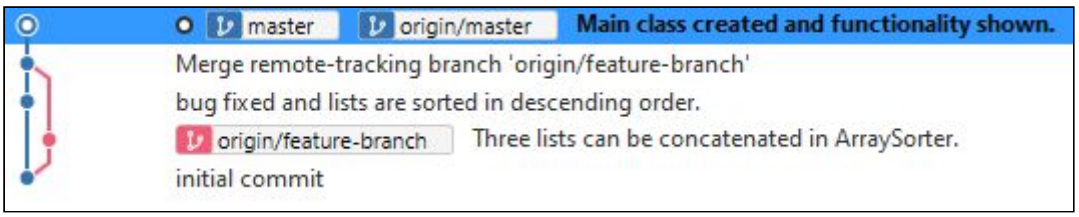
Now, your ***local and remote master*** branch are on the same commit. After, create a ***Main class*** in your project. The ***Main class*** should create three ArrayLists. The contents of the lists are as follows:

ArrayList1 → [100, 500, 900]
ArrayList2 → [800, 600, 300]
ArrayList3 → [400, 700, 200]

After, create an ***ArrayOrganizer*** object and concat the three lists and sort them via the ***ArrayOrganizer*** object. Hence, print the resulting array as follows:

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Result => [900, 800, 700, 600, 500, 400, 300, 200, 100]

Process finished with exit code 0
```

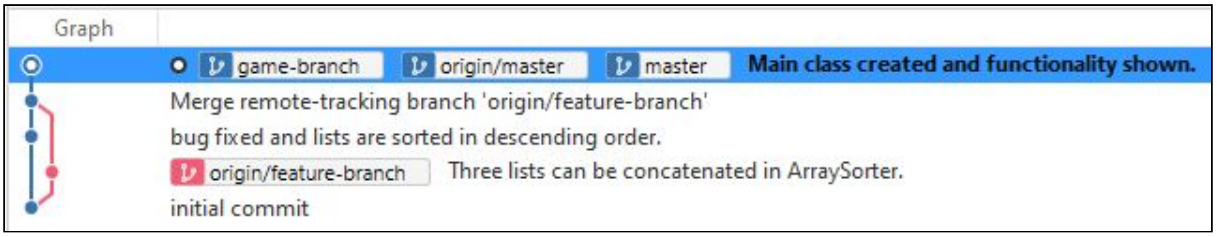
If you came to this point, now it is time to make a ***commit***. Make a commit and type ***"Main class created and functionality shown."*** as the commit message. Do not forget to ***push*** the changes to the remote repository. After these operations, you should have the following commit history:



20pts (comitting, branching and fast forward merging)

In the following two sections, a very basic die game will be implemented. You will use the same project. In the same project, we will create two java classes. These will be the die and player classes.

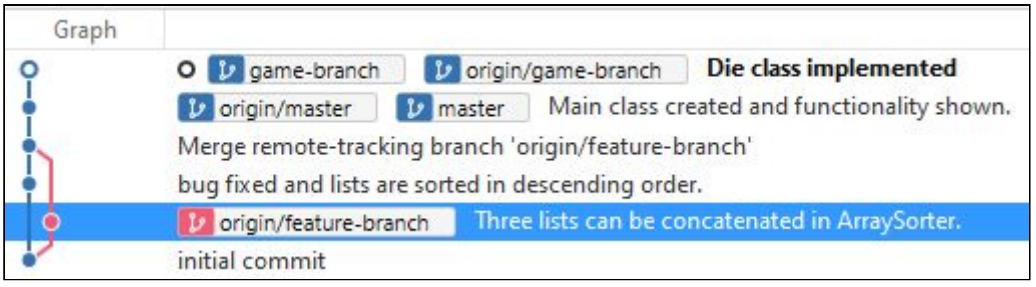
Initially, create a branch named ***game-branch.*** After creating this branch, you should have the following commit history:



Now, checkout to the ***game-branch.*** Then, in the same project, create a new ***Die class.*** The ***Die class*** needs to have a method that returns a number between one and six. The implementation needs to be like the following:

```
public class Die {

    public int rollDie() {
        return (int) (Math.random() * 6) + 1;
    }

}
```

After implementing, make a commit with a message of ***"Die class implemented.".*** Again, do not forget to ***push*** the changes to the ***remote branch.*** Importantly, you need to push the changes to the remote ***game-branch.*** It might not exist, however, after doing the push operation, it will be created automatically. After these operations, your history needs to look like the following:



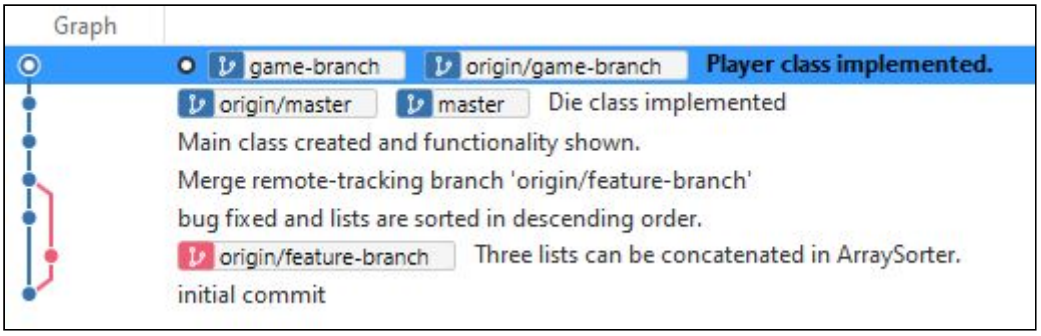
Now, let's make a ***merge*** operation, however, this will be a ***fast forwarding merge.*** We will need to ***merge the game-branch into master branch.*** For that, first checkout to ***master*** branch, and then make a ***fast forward merge*** as shown in the lectures. After doing a fast forward merge, ***push*** the changes to the ***remote master*** branch, hence, you should have the following commit history:

20 pts (comitting and no fast forward merging)

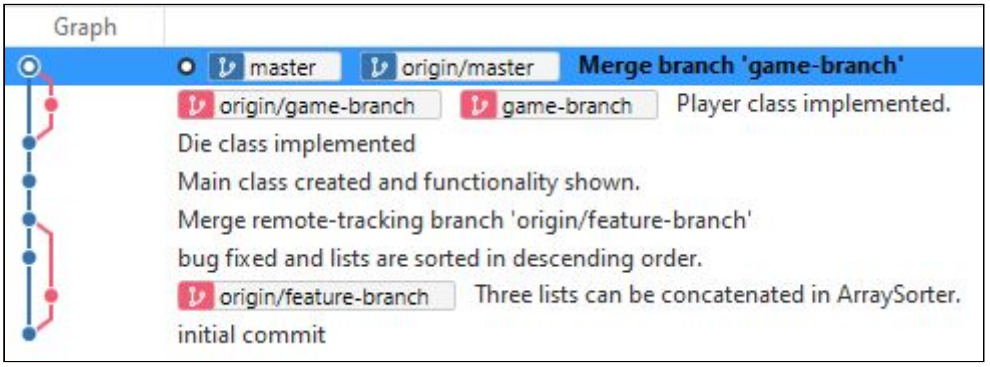
Now we will implement the ***Player class*** in the project. For that, first, checkout to the ***game-branch*** and then switch to the project. In the project, create the ***Player class.*** The ***Player class*** contains a ***score*** and ***name*** value. The name value will be passed through the ***constructor*** and the score value will be initialized to 0. Create the corresponding ***getter*** and ***setter*** methods and create a method whose name is ***addScore***. This will receive an integer and will add it to the score variable. Also provide a ***toString*** method that represents the student such as ***"<name> has <score> points"***. The player class should look something like the following:

```java
public class Player {

    private String name;
    private int score;

    public Player(String name) {
        this.name = name;
        this.score = 0;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getScore() {
        return score;
    }

    public void setScore(int score) {
        this.score = score;
    }

    public void addScore(int point) {
        this.score += point;
    }

    @Override
    public String toString() {
        return name + " has " + score + " points.";
    }

}
```

After implementing, make a commit with a message of ***"Player class implemented."*** Again, do not forget to push the changes to the remote ***game-branch***. After these operations, your history needs to look like the following:



Again, let's make a ***merge*** operation, however, this time, it will be a ***no fast forwarding merge.*** We will need to ***merge the game-branch into master branch.*** For that, first checkout to ***master*** branch, and then make a ***no fast forward merge*** as shown in the lectures. After doing a ***no fast forward merge***, ***push*** the changes to the ***remote master*** branch, hence, you should have the following commit history:

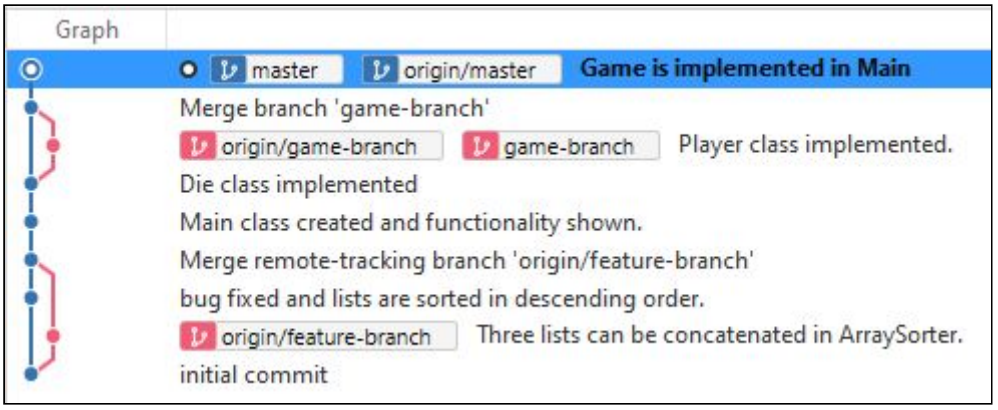


5 pts (committing)

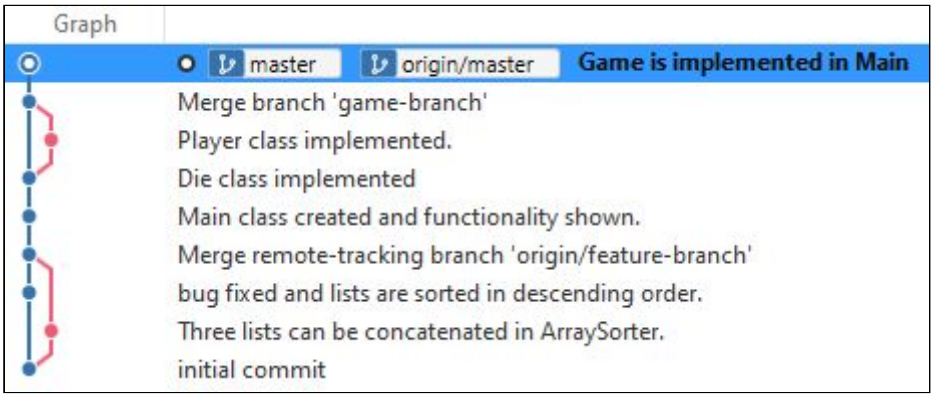
Now, we will implement the game functionality in the ***Main class.*** Checkout to **master** branch and in the ***Main class***, first, create 4 players whose names are **Tom, John, James and Henry**. After, create a ***Die object***. Run a for loop for ***5 rounds*** and in each round, roll the die and add the die result to the ***players score***. After completing 5 rounds in the for loop, print each users result. The output of the resulting ***Main.java*** file needs to look like the following:

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Result => [900, 800, 700, 600, 500, 400, 300, 200, 100]
-----------------------------------
Tom has 20 points.
John has 18 points.
James has 19 points.
Henry has 17 points.

Process finished with exit code 0
```

After getting these outputs, make a commit with a message of ***"Game is implemented in Main class."***. Again, do not forget to ***push*** the changes to the ***remote master*** branch. After these operations, your history needs to look like the following:

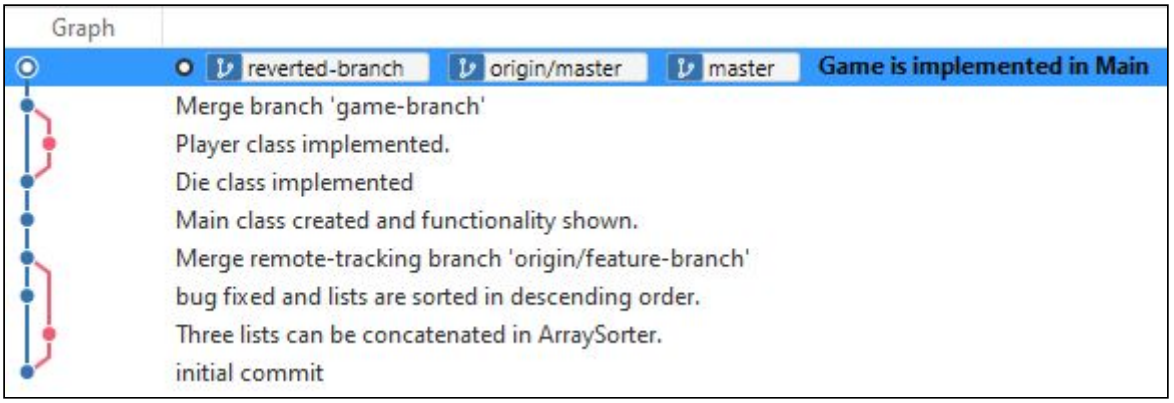


Now, let's delete all of the branches except the ***local and remote master*** branch. After the delete operation, you should obtain the following:

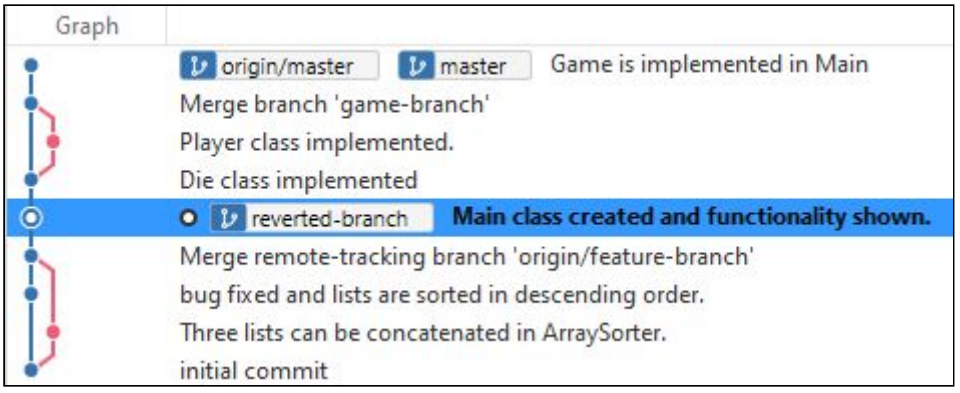


10 pts (reverting commit, branching)

Now, again, lets create a new branch whose name is ***reverted-branch.*** After creating the branch, we will have the following commit history:



Checkout to ***reverted-branch*** and then right click on the ***"Main class created and functionality shown."*** commit and select ***"Reset current branch to this commit".*** In the opening dialog, select the ***"discard all working copy changes"*** and hit okey. After doing it, your history should seem like the following:



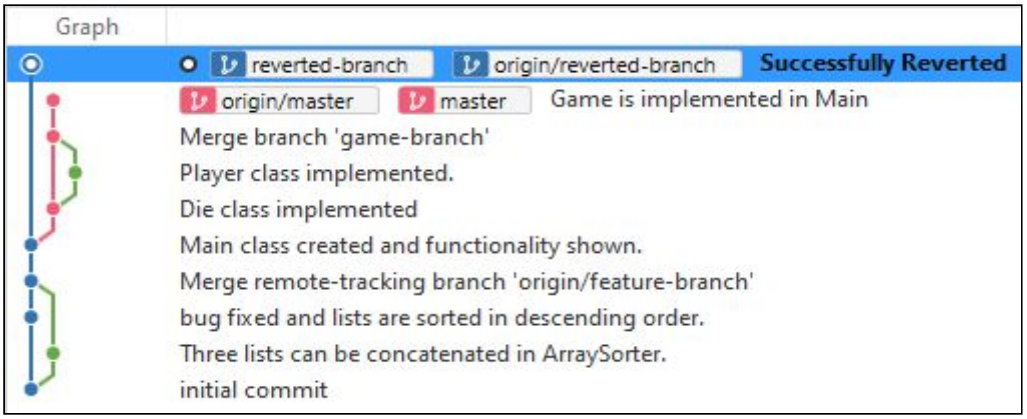
Now, switch to your project and in the ***Main class***, Add the following line of command to the end of the ***Main class:***

System.***out***.println(**"Successfully reverted in Git"**);

After adding it, run the Main.java file and you should get the following output:

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Result => [900, 800, 700, 600, 500, 400, 300, 200, 100]
Successfully reverted in Git

Process finished with exit code 0
```

After getting this output, make a commit with a message of ***"Successfully Reverted"***. Again, do not forget to ***push*** the changes to the ***remote reverted-branch***. It might not exist, however, after pushing, it will be created automatically. After these operations, your history needs to look like the following:



Submission: 5 pts (README)

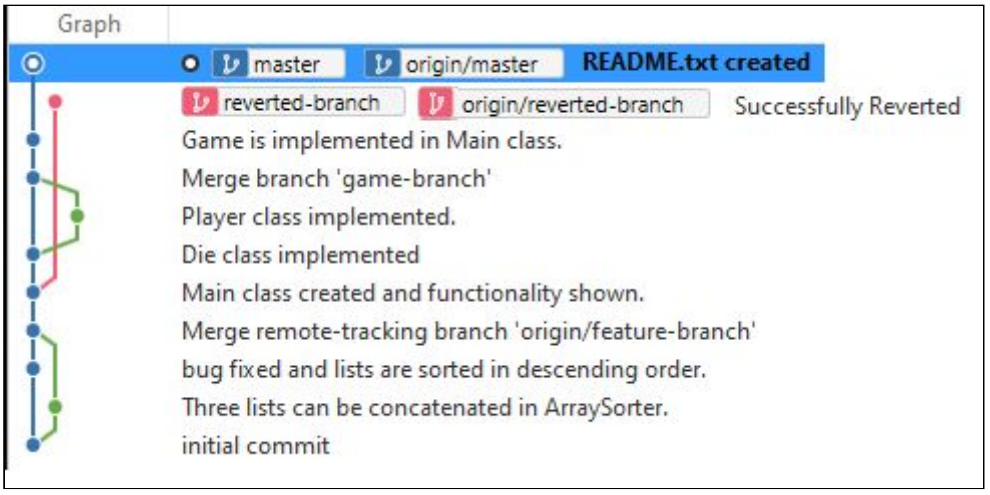
Now, lets checkout to ***master*** branch. Create a ***README.txt*** file. In the txt file, write your ***name*** and ***id***.

<name> <id> \n

As an example:

```
Fatih Uyanık 21602490
```

After adding the ***README.txt*** file, make a commit in master branch and the corresponding commit message is ***"README.txt created"***. After the commit, ***push*** the changes to the remote repository. After the commit, your history should be as the following:



After getting the above history, you finished the lab. Make sure that your repository is public. Please send your repository link, your name and student id to the following address via email: **ytsmartcode@gmail.com**