Pro Dev Session

Pro Dev Session

You may want to write this down.

GitHub and Git are powerful tools and we’ll continue to focus on them for a few days before we move on the new topics.

Today we’ll focus on the `reset` command.

Pro Dev Session

You may want to write this down.

Recall that we previously learned to use `git status` to see which files are unstaged and which are staged but have uncommitted changes.

Recall also that we used the `git log` command to see a list of commits and their SHAs.

Try these command out again to get a refresher on what they do.

Pro Dev Session

You may want to write this down.

The new command that we'll focus on today is `reset`. Reset can be used with one of two flags `--soft` and `--hard`.

The `reset` command is a destructive command and changes your git history. Don't be scared of it, but use it wisely.

The `git reset --soft <SHA>` will remove all commits from your latest back to the specified SHA (non-inclusive). Any changes made during that time are move to the unstaged area. You can see this by typing `git status`.

This is a great tool when you realize that your last 4 commits should have all been one.

Pro Dev Session

You may want to write this down.

The `git reset --hard <SHA>` will remove all commits and all changes from your latest back to the specified SHA (non-inclusive). Any changes made during that time are permanently deleted.

This is a great tool when you realize you took a wrong turn 4 commits ago and want to remove that history and those files/changes.

Pro Dev Session

You may want to write this down.

Before using the `reset` command, the safest thing to do is create a new branch using `git branch <branchname>`.

Creating a new branch creates a clone of the branch you are currently on. It’s like a safety net in a sense.

Git Practice

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Goals:

- Create a new repository and clone it.
- Create a new branch and switch to it.
- Add a new file and add commit and change. Repeat this 5 times.
- Now let's try resetting. First make a new branch called back-up. Don't switch to it.
- Now type `git status`. Everything should be staged and committed.
- Type `git log` and copy the SHA from 4 commits ago.
- Type `git reset --soft <SHA>` then `git status`. Notice that those files are now in the unstaged area. Changes aren't lost. Add and commit these. Type `git status` again. See how this could be useful for turning multiple commits into one?
- Now create another file, add, and commit. Type git status. Everything should be clean. Create another branch called backup2.
- Type `git log` and choose a SHA. Type `git reset --hard <SHA>`.
- Type `git log`. Note that all of your files and changes are gone. Completely.
- Now that you don't need those backups delete them with `git branch -D <branchname>`

20 minutes!

Stand Up

More TDD

More TDD

GETTING BETTER AT TESTING

Notebooks Ready? It’s time for a mini lecture.

More TDD

GETTING BETTER AT TESTING

We stated the importance of of Testing and TDD in the previous class, but we can’t overstate it enough.

We will spend the remainder of the course practicing TDD on virtually every project.

It may feel foreign now, but it will become second nature soon enough.

Let’s Take a Moment . . .

Let’s remember that we are dedicating 3 months of our time, to permanently change the trajectory of our lives.

This means that for 3 months we need to seize every opportunity for growth, embrace every struggle as a part of progress, and seek every chance to learn more.

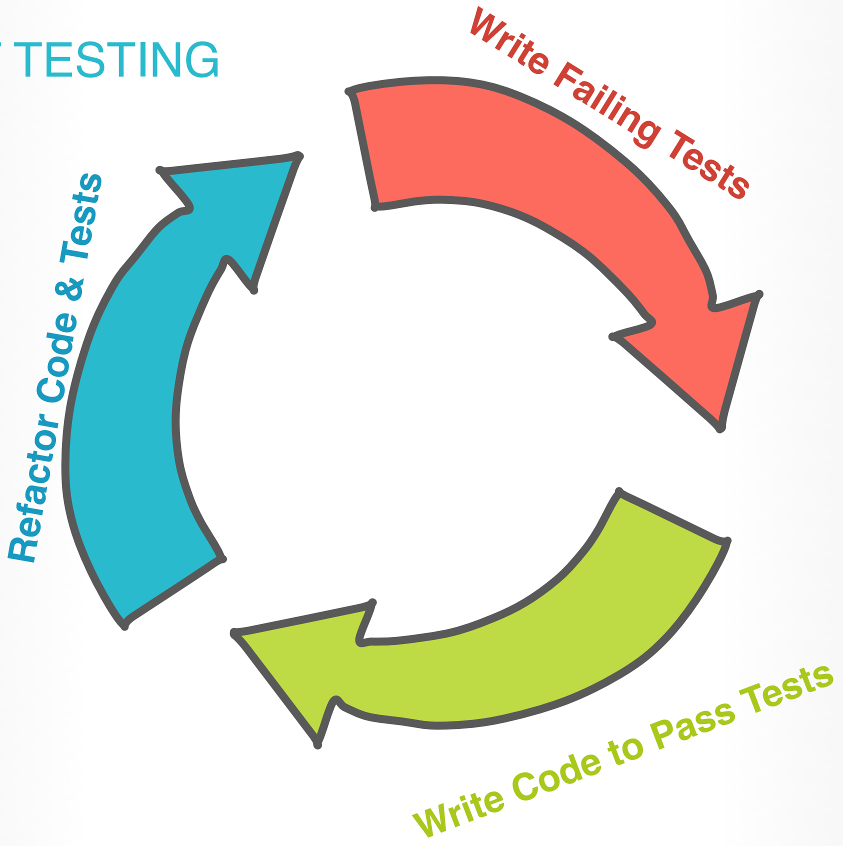
To this end, you cannot learn everything you need to be the most successful developer you can be during class hours. It is pivotal that you learn to learn independently.

If you want to make the most of this opportunity, you must read deeply about the concepts we cover in class, tackle side projects in every spare moment no matter how small, and combine this research and these practical skills to tackle increasingly difficult tasks.

Go home tonight and study testing. Challenge yourself this week to give your all and try to become the class expert on testing.

More TDD

GETTING BETTER AT TESTING



More TDD

GETTING BETTER AT TESTING

We'll spend some more time practicing the Red Green Refactor method, but first let's learn a little more about what JUnit can do.

Let's take a little detour and read through the documentation. This is a vital skill for all developers.

Note that JUnit 5 is out now, but we are using 4.12. That's because 4.12 ships with the Spring Initializr starter dependencies. You are welcome (and encouraged) to independently upgrade your projects to 5 if you like!

More TDD

PAIR PRACTICE

It’s time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Read through the documentation for 10 minutes. Be curious. Google. Learn.
- After 10 minutes, implement one new annotation that you learned (Note that the heading for assesertions in the documentation says **Annotation Type <Name>**
- Implement one additional feature you learned about.
- Be prepared to discuss your learnings.

20 minutes!

Stay Seated & Take 3 Deep Breaths.

RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes.
We'll start back in a few minutes.

Intermediate JUnit

Intermediate JUnit

MASTERING OUR CRAFT

Notebooks Ready? It’s time for a mini lecture.

Intermediate JUnit

MASTERING OUR CRAFT

Let’s take a crash course in some useful JUnit features.

@Before
@BeforeClass
@AfterClass
AssertArrayEquals
AssertEquals(double expected, double actual, double delta)
AssertThrows
AssertThat

TDD Practice

PAIR PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Write tests to test a method that takes an array of arrays and returns an array with the average of each inner array
- The method should throw an IllegalArgumentException if any of the numbers are negative.
- After you have failing tests, implement the method

Input: [[1,2,3] , [4, 4, 4], [2, 4, 6]]
Output: [2, 4, 4]

15 minutes!

Now that we know how to test Arrays, it’s time to turn our attention to testing object equality.

Critical Thinking

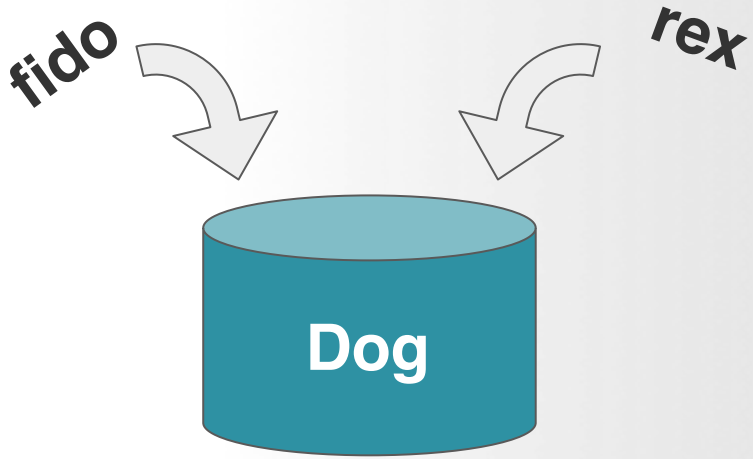
What does it mean for two objects to be equal?

Intermediate JUnit

MASTERING OUR CRAFT

There are 2 definitions of object equality. The first is that the two variables are actually two references to the same object. This is known as shallow equality.

```
Dog fido = new Dog();
Dog rex = fido;
```

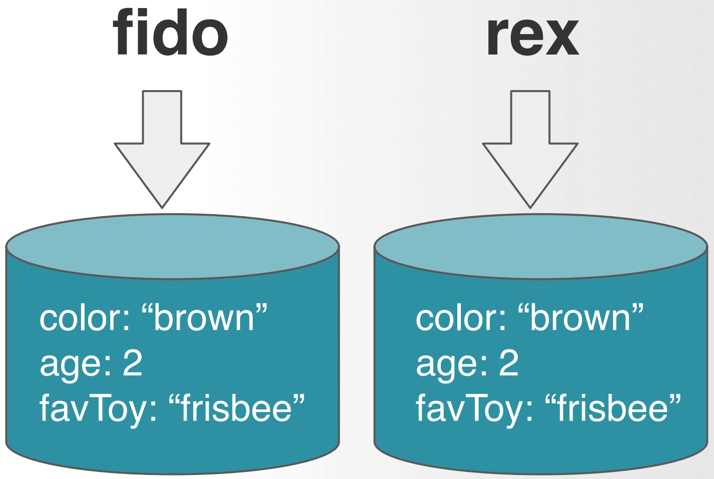


Intermediate JUnit

MASTERING OUR CRAFT

The second is that two objects share the same value for all of their properties. This is known as deep equality.

```
Dog fido = new Dog("brown", 2, "frisbee");
Dog rex  = new Dog("brown", 2, "frisbee");
```

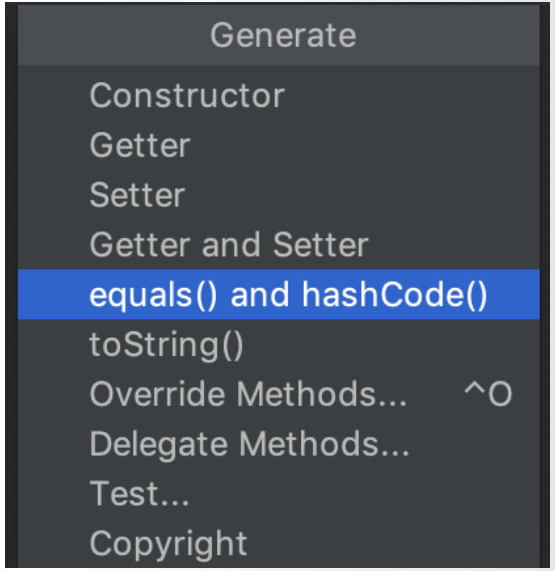


Intermediate JUnit

MASTERING OUR CRAFT

Checking deep equality involves telling the program how to measure the equality of two objects.

This means we need to implement (and more technically override) the equals method. Right click in your class, then select “Generate”.



lunch.

Lab Time

TDD Practice

PAIR PRACTICE

It’s time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Write tests to satisfy the requirements below.
- Then implement the code to pass the tests.

Your requirements state that you need a Physics library with the following functionality

- All numbers are doubles
- calculateForce which returns the product of mass and acceleration
- calculateVelocity which returns distance2 minus distance1 divided by time2 minus time1.
- calculateMass which returns weight divided by gravity
- calculateAcceleration which returns velocity2 minus velocity1 divided by time2 minus time1.

TDD Practice

PAIR PRACTICE

It’s time to fly. Focus. Work hard. Ask for help when you need it.

Goals:

- Write tests to satisfy the requirements below.
- Then implement the code to pass the tests.

Your requirements state that you need a Summing library with the following functionality

- You need a class that has four properties primeSum, evenSum, oddSum, sum. Each property is an integer.
- You need an additional class with 1 property - a List of instances of the above class.
- You must be able to add new instances to the List. A new instance is created by passing in an integer. The sum of all of the primes from 1 to this number should be calculated and stored as primeSum. Similarly, the sum of all even numbers from 1 to this number should be calculated and stored as evenSum. Oddsum stores the
  sum of odds and sum stores the sum of all numbers from 1 to the supplied number.
- The latter class should have a method to flatten the List to an array, which is to say, take the 4 numbers from each object in the list and store them in an array.
  e.g.: [primeSum1, evenSum1, oddSum1, sum1, primeSum2, evenSum2, oddSum2, sum2, …]

Breathe

Stay Seated & Take 3 Deep Breaths.

RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes.
We’ll start back in a few minutes.

Mockito

Mockito

THAT CODE’S GOT MOCKSY!

Notebooks Ready? It’s time for a mini lecture.

Mockito

THAT CODE’S GOT MOCKSY!

Many of the methods in our application aren’t stand alone units of code like the examples we’ve been using.

The web layer relies on the service layer.

The service layer relies on the data layer.

The data layer relies on the database.

Mockito

THAT CODE’S GOT MOCKSY!

Suppose you are on the service layer team and you want to write your unit tests and begin coding. But the data layer isn’t built yet.

How could you test your Service layer as a stand alone unit?

Enter Mockito!

We can Mock (or fake) our data layer for the purpose of testing the functionality of our service layer.

Mockito

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
public class Movie{

    private String name;
    private int releaseYear;

    public Movie(String name, int releaseYear){
        this.setName(name);
        this.setReleaseYear(releaseYear);
    }

     //getters and setters excluded for brevity
}
```

Mockito

```
public class MovieService {

   private MovieDao movieDao;

   public MovieService(MovieDao movieDao){
       this.movieDao = movieDao;
   }

   public List<Movie> searchMyMovie(int id, String name){
       List<Movie> myMovies = movieDao.getUserMovieList(id);
       return myMovies
               .stream()
               .filter(movie -> movie.getName().contains(name))
               .collect(Collectors.toList());

   }
}
```

Mockito

```
@RunWith(MockitoJUnitRunner.Class)
public class MovieServiceTest {

   @Mock
   MovieDao movieDaoMock;

   @InjectMocks
   MovieService movieService = new MovieService(movieDaoMock);

   Movie movie1;
   Movie movie2;
   Movie movie3;

   List<Movie> movieList;


...
```

Mockito

```
    @Before
    public void setup(){
         movie1 = newMovie("Jurassic Park", 1994);
         movie2 = newMovie("Jurassic Park II", 1997);
         movie3 = newMovie("A Walk in the Park", 2011);

         movieList = Arrays.asList(movie1, movie2, movie3);
    }


    @Test
    public void shouldFindMoviesByNameFromUserMovieList(){
         List<Movie> expectedList = Arrays.asList(movie1, movie2);
         when(movieDaoMock.getUserMovieList(1)).thenReturn(movieList);
         assertEquals(expectedList, movieService.searchMyMovie(1, "Jur"));
     }

}
```

Mockito

PAIR PRACTICE

It’s time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Revisit your wine application from yesterday.
- Refactor your tests so that they rely on a Mock DAO instead of hard-coded arraylists. These tests should fail.
- Refactor your code to pass all tests.

20 minutes!

Sneak Peek

More JUnit & Mockito

BRINGING IT ALL TOGETHER

Next class we will continue to cover Mockito and JUnit, testing the Web and Data Layers of our application.

Wrap Up

Module 5 Lesson 3

HOMEWORK

You don’t have to submit your nightly homework, but you are expected to complete it.

Complete any remaining katas.

Read:

The Benefits of Using assertthat

and

Using Hamcrest

Daily Assessment Today

You may leave after a staff member approves your assessment.

Daily Assessment

Work INDIVIDUALLY to complete all of the goals below.

Goals:
- Write a test for a method that divides two doubles. Think through all edge cases
- Implement the code to pass the tests.