The first assignment

Creating a bare-bones project

1. Create a new Java project
    a. **Note:** You can use the Java Hello World template when prompted
    b. Resources: LINK

2. Build the project
    a. Resources: LINK

3. Run the Main class to verify that everything basic is working as expected
    a. Resources: LINK

Adding classes that represent web pages

- **Planning phase:**

    1. Analyzing the website structure
        1. Navigate to https://www.imdb.com
        2. Observe the home page
        3. Search for a movie you love and observe where you landed
        4. Click on Sign in button and observe where you landed
        5. Click on Create account button and observe where you landed

    2. Analyzing the page structure
        - For each page previously visited, note down every element on the page that you personally feel you should interact with if you were writing an automated test

          NOTE: Your list for the Home page might look something like:

          Logo, search field, sign in button, help link, see more opening this week link, recently viewed, contact us link, copyright link...

- **Mapping phase:**

    1. Within the **src** folder in your project, create a new package named **pages**

    2. For each of the pages visited in the planning phase, create a class named ***PageName*Page** within the newly created **pages** folder

3. For each of the previously noted elements, do the following:

   - Navigate to the class that corresponds to the page the noted element is located on e.g. navigate to HomePage.java if you would like to map the search field element
   - Create a **field** for that will be **private**, have a return type of **String**, be named after the element that you are currently mapping with an **underscore** in front of its name and have a **value** that represents the element's name.
     - Resources: LINK
     - For example, a mapped search field could look something like this:

```
package pages;

public class HomePage {
    private String _searchField = "Search field";
}
```

   - For each field that you created in the previous step, create a **method** that will be **public**, have a return type of **void**, be named after an element that it corresponds to with a "**get**" in front. Method's body should just **print the value of the appropriate field**.
     - Resources: LINK, section Return Values
     - For example, a method for search field could look something like this:

```
package pages;

public class HomePage {
    private String _searchField = "Search field";

    public void getSearchField(){
        System.out.println(_searchField);
    }
}
```

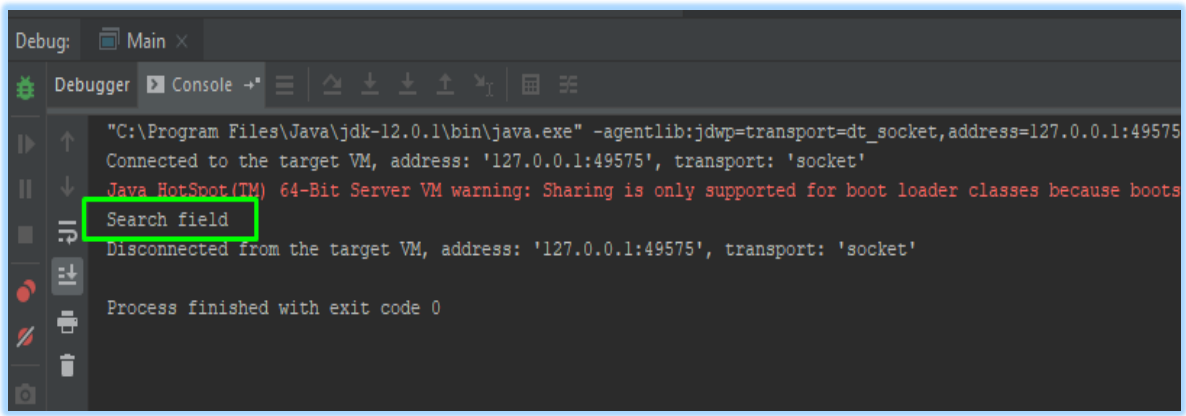
- **Quick test of everything we have done so far:**
    1. Within your main() method, inside the Main class, create a **new instance of HomePage** and place it inside a local variable named **homePage**
        - Resources: LINK
    2. Call any method you previously created within the HomePage class using the newly created instance.
        - Your main() method should look something like this:

```
import pages.HomePage;

public class Main {

    public static void main(String[] args) {
        HomePage homePage = new HomePage();
        homePage.getSearchField();
    }
}
```

    3. Run or Debug your main() method.
    4. You should be able to see the following in your console:



    5. Feel free to repeat this step before proceeding with the assignment until you feel comfortable with instantiating the classes and calling their methods.
    6. Enjoy your newly created structure before we continue upgrading it.

Introducing basic inheritance

- **Analysis phase:**
  1. Review your newly created Page classes
  2. Notice how every method has the **exact same body** and that is System.out.println("something")
  3. Realize that if we wanted to change the message being printed to, for example, "Now printing the value: *value of the field*", we would have to go through each and every method and copy paste this line there
  4. Decide to rebel against the oppressive existence of code duplication and take action!
- **Refactoring phase:**
  1. Create a new class under pages folder and name it BasePage
  2. Create a public and void method named printMessage that takes a **string as an argument**
     - Resources: LINK section Method parameters
  3. In the body of the method, write the code to **print** the string that we are passing as an argument.
     - Resources: LINK section Method parameters
     - In case you forget how to print a message - LINK
  4. For each Page class previously created, make it **inherit** the newly created **BaseClass**
     - Resources: LINK section The syntax of Java Inheritance
  5. **Replace** every call of System.out.println within each Page class with the newly created printMessage method call.
     - You can either do this manually or utilize the Intellij's CTRL + SHIFT + R shortcut to replace text on the project level (be aware that you could mistakenly replace the System.out.println in the BasePage as well)
     - For example, your HomePage could now look like this:

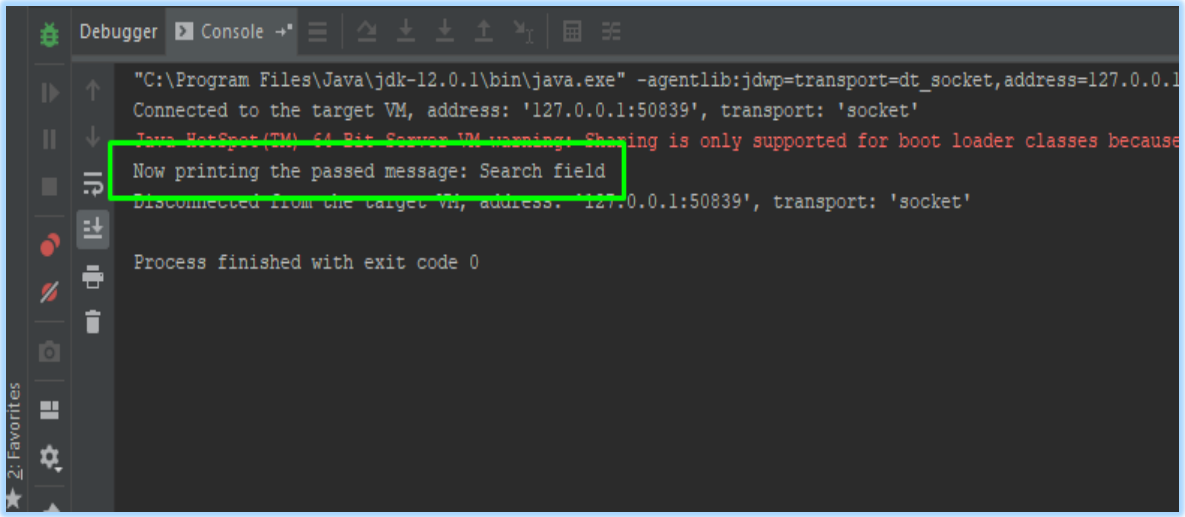
```
package pages;

public class HomePage extends BasePage {
    private String _searchField = "Search field";

    public void getSearchField(){
        printMessage(_searchField);
    }
}
```

  6. Change the printMessage method within the BaseClass to now print "Now writing the passed message: *message_that_we_pass_as_an_argument*" instead of just "*message_that_we_pass_as_an_argument*"
     - Resource: LINK

- **Quick test of our refactoring**
    1. Run or debug your main() method, just like previously done
    2. You should be able to see the following in your console:



    3. Nothing else should have changed aside from this.

    If now we wanted to further change the message that we are printing, **we could do it in a single place**, without the previous need to copy paste the new message all across the project. This will heavily translate to much **lower maintenance cost** that adds up over time.

Centralizing Page instantiation

- **Analysis phase:**
    1. Review the body of your main() method in the Main class
    2. Realize that it is quite a hassle to separately instantiate every page class that you need
    3. Wish that you could do it in one place
    4. Decide to yet again take action!
- **Execution phase:**
    1. Create a new class named Imdb under pages folder
    2. For each Page class that you have that is not BasePage, add a field to the Imdb class that is **private**, have a **return type equal to the Page class that you are currently mapping (we previously did this only with String),** is named after the Page class that you are currently mapping with an **underscore** in front of the name and has NO initial value.
    3. For each created field, create a method that is **public**, has the same **return type** as the appropriate field (so, the class that you are currently mapping) and is named like the page that it is supposed to return

4. Add a body to each of the methods that will have the following functionality:
    - If the corresponding field is currently equal to **null**, create a **new instance** of the corresponding Page class and **assign it** to the field
    - **Return** the value stored within the field after the first condition is checked
5. This must sound complicated, so here is an example of how the Imdb class with the HomePage properly mapped should look:

```
package pages;

public class Imdb {
    private HomePage _homePage;

    public HomePage homePage(){

        if ( homePage == null){
            _homePage = new HomePage();
        }

        return _homePage;
    }
}
```

6. Once the previous steps have been completed for each Page class, now it is time to utilize what we just reafactored.

- **Quick test of our refactoring**
    1. Navigate to your main() method and temporarily comment out the code in it's body
        - To comment out the code, click on the line that you want to comment out or select the code by dragging the mouse and press CTRL + /
    2. Now, within the main() method, create a new instance of Imdb class and store it in a variable named Imdb.
    3. Call the method that corresponds to the desired page and then call the method that corresponds to the desired element.
    Your main() method could look like this:

```
import pages.Imdb;

public class Main {

    public static void main(String[] args) {
        Imdb imdb = new Imdb();
        imdb.homePage().getSearchField();
    }
}
```

4. Run or debug the main() method – the previous result should not have changed

All of your pages and elements on them are now **easily accessible from a single file** and we could theoretically access every page element right from the Main class if we so desired.