



Properties – Configuration Properties



Agenda

- Understand a current issue we have with **hardcoding**
- Understand how can we solve the issue
- Create a properties file
- Create a utility class to read from the properties file
- Practice



What is Test Data?

- Test data definition: data created or selected to satisfy the execution preconditions and inputs to execute one or more test cases.
 - Basically, all of the data we select (or create) to pass to AUT* to verify expected output is called test data
 - Example: username, password, search values etc.
-
- AUT*: Application Under Test



How have we been passing our data until today?

- We have been directly passing our test data in our tests
- Which means we are keeping our test data **inside our source code**
- This means we are **hard coding** our test data.



What is Hard Coding?

- Google.



What is hard coding?

- Hard coding is keeping your data (test data) in the source code.
- If you have to go in your java-selenium code to change your data, it means you have hard coded it.
- We will be creating a logic where we read our test data from outside of our code.



Issues we are trying to solve:

- We are **hardcoding** some of the **important test data**.
- We are passing our test data from within our source code itself.
- This means every time we need to change it, we must go in each and every place we used it and change one by one.
- Ex: Working on env1, but its down. What can we do with our existing structure?



- environments:

qa1.vytrack.com

qa2.vytrack.com

staging1.vytrack.com

staging2.vytrack.com

prod: www.vytrack.com

QA environment : is generally used testing purposes

staging env : is also used for testing but generally has better server support etc.

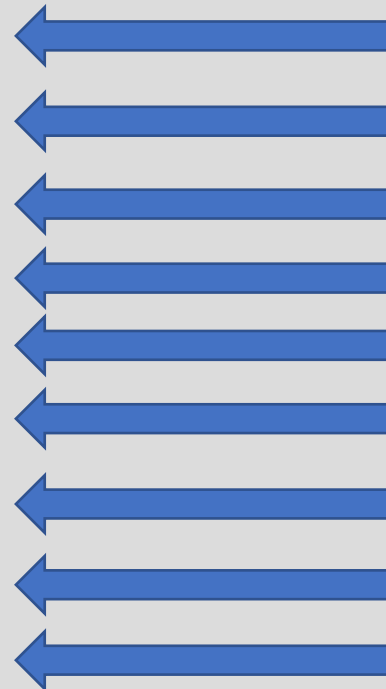
prod : is actual product that end user is using.




```
driver = WebDriverFactory.getDriver(browserType: "chrome");  
driver.manage().window().maximize();  
driver.get("https://login1.nextbasecrm.com/");
```

- tests
 - day1_selenium_intro
 - day2_locators_getText_getAttribute
 - day3_review_css_xpath
 - T1_CRM_locators_getText
 - T2_getText_getAttribute
 - T3_Login_button
 - T4_Reset_password_button
 - Warmup1
 - day4_findElements_checkbox_radio
 - day5_testNG_intro_dropdowns
 - day6_alerts_iframes_windows
 - day7_webtables_utils_javafaker

If I want to run all of my @Tests
In different environment,
I now have to go each and every
single class and @Test I created.
I have to change line by line.



In short terms:

- We are passing important test data inside of `.java` class directly.
- It is called "hardcoding"
- It is not scalable.
- It is not re-usable.
- It is not easy to maintain.
- Therefore, it is not a good practice!

After today's session, you should be able to:

- Pass certain important test data into our tests from outside of our code, external file we will be creating named:
“configuration.properties”
- We will be able to change the test data from outside of our **Java** class.



- Java is keeping track of some of the important properties regarding your computer and project etc.
- Let's see some of them!



Different type of file types:

A few common type of files.

- PDF File : something.pdf → portable document format
- DOCX : testData.docx → MS Word
- Text file : testData.txt → regular text file
- Excel file : testData.xlsx → a bit more complex with rows and cells
- CSV file : testData.csv → comma separated value
- Properties file : testData.properties → key=value format



We will be using **.properties** type of file

- What is a properties type of file?
- Basically, it stores data in **KEY = VALUE** pairs.
- It looks like this:

```
1 browser=chrome
2 env=https://www.google.com
3 username=helpdesk1@cydeo.com
4 password=UserUser
```

- Key will be inside of our Java class, and the value will be outside of java.
- This way we will be able to manipulate the data from outside of the code.



.properties type file has rules:

- Data stored in key=value pairs
- There can only be 1 key with same name.
- When we try to get value, we must pass the identical “key”



- Key=value format is useful because:
 1. “key” part will be stored in `.java` classes so it will be hard coded.
 2. “value” part will be stored in `.properties` file, so it will NOT be hard coded.

```
1 browser=chrome
2 env=https://www.google.com
3 username=helpdesk1@cydeo.com
4 password=UserUser
```



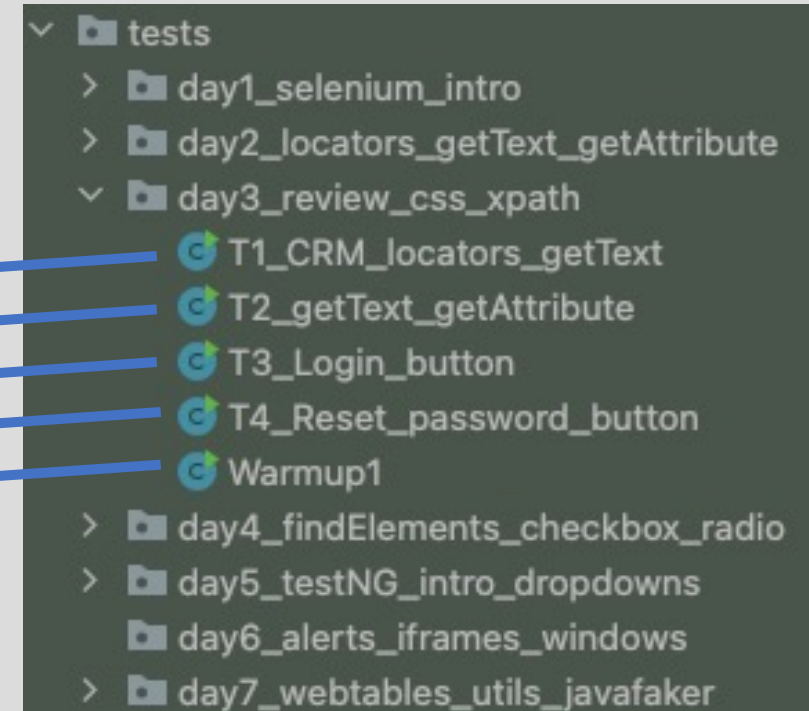
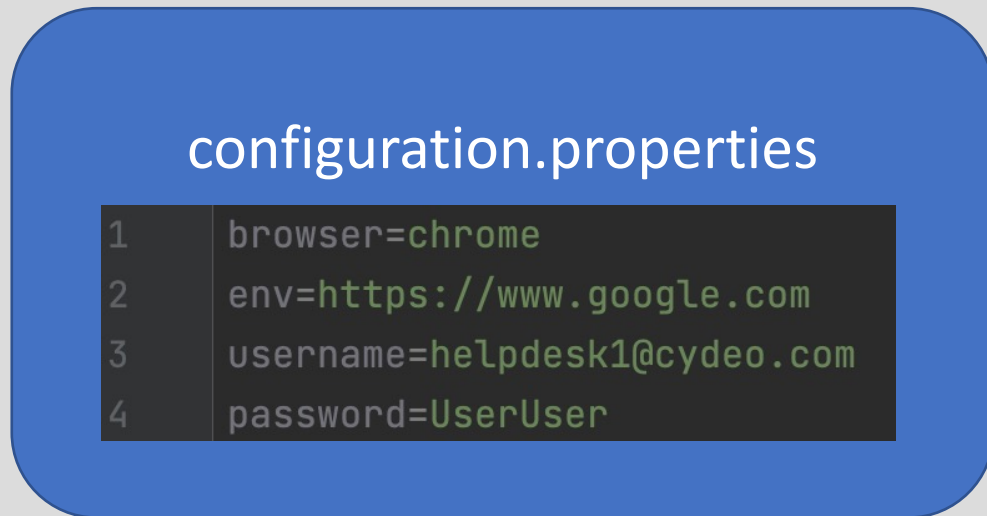
```
driver = WebDriverFactory.getDriver(browserType: "chrome");  
driver.manage().window().maximize();  
driver.get("https://login1.nextbasecrm.com/");
```

- tests
 - day1_selenium_intro
 - day2_locators_getText_getAttribute
 - day3_review_css_xpath
 - T1_CRM_locators_getText
 - T2_getText_getAttribute
 - T3_Login_button
 - T4_Reset_password_button
 - Warmup1
 - day4_findElements_checkbox_radio
 - day5_testNG_intro_dropdowns
 - day6_alerts_iframes_windows
 - day7_webtables_utils_javafaker

If I want to run all of my @Tests
In different environment,
I now have to go each and every
single class and @Test I created.
I have to change line by line.



1. Instead of writing passing some test data in every single class, we will provide from 1 file.



2. All the rest of our **.java** classes will be reading from that file.
3. When we change 1 file, all of the rest of the code will be updated automatically.



What are we trying to achieve?

- The data we will be storing in our configuration.properties is going to be the kind of data that will be able to **change the flow** of the execution of our test suites.
- Examples:
 - Be able to login with different usernames
 - Be able to enter different password
 - Be able to change the browser type we are running our test suite
 - Be able to change the environment we are running our test suite on



How to read from a **properties** file?

1. We need to use Properties class. The object of this class can read .properties files.

```
Properties properties = new Properties();
```

2. We need to use FileInputStream class. This class allows us read data from a file.

```
FileInputStream file = new FileInputStream(name: "file.path");
```

3. After opening file stream using its path, we load the file to properties object.

```
properties.load(file);
```

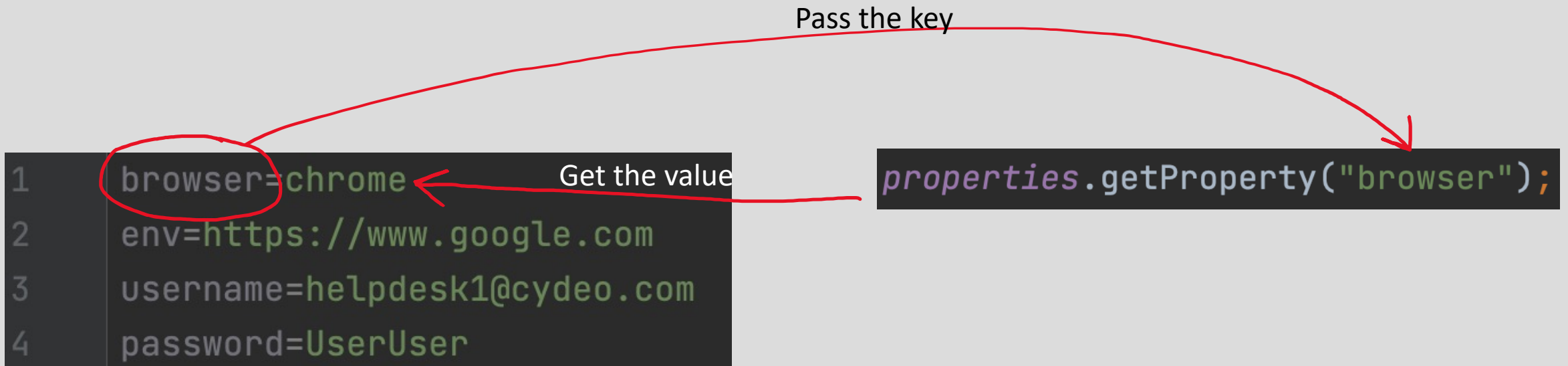


4. Once object is loaded with the file, we can get any “value” by passing “key”

```
properties.getProperty("browser");
```



Pass the **key** → Get the **value**



Let's create our ConfigurationReader utility class

- Reading one time is good but we need to create a re-usable utility method so we can read from our .properties file whenever we want.
1. Create a new class: ConfigurationReader.java
 2. Copy/paste our .properties file reading logic
 3. Edit if needed
 4. Create a new utility method to re-use our logic
 5. Method should accept “key” as a **String**, and return “value” as **String**

