* Nie je možná inicializácia ElementsCollection, musí byť hneď naplnená hodnotou(asi lebo je dedená abstraktnou triedou:

ElementsCollection listOfUsers;

ElementsCollection listOfUsers = getUserNameFromTable();

* Nie je možné dohladávať všetky elementy vrámci ElementsCollection, jedine jeden element, ktory mi potom kolekciu zmení na SelenideElement, treba hladať priamo vyhladavaný element(napr zo všetkých tr elementov chcem vytiahnúť td elementy s určitou klasou, nemožno to spraviť cez find ani findAll, dokonca to nie je možné obísť ani pomocou iterácie cez foreach, pretože metoda add nie je funčkná pre ElementsCollection):

ElementsCollection listOfUsers = $$("tr.mat-row").findAll(“td”).get(0);

ElementsCollection listOfUsers = $$("td.mat-column-userName");

* Pri načítavaní zoznamu napr uzivatelov z tabulky mi jeden údaj pomenoval ako displayed false miesto mena, bolo to pravdepodobne za to, že ten element už nebol vrámci stránky viditeľný ale všetky ostatné za ním boli korektne načítané, každopádne sa to vyriešilo skrolovaním na posledný element:

$("table > tbody > tr:last-child").scrollIntoView(false);

* Rozdiel medzi foreach, filter a map – na všetky iterovateľné kolekcie je možné použiť streamy, rozdiel medzi týmito troma funkciami je nasledovný:
  + foreach nevrati nič (vracia void), čiže on len vykoná nejakú funkciu s lambdou ktorú mu zadáme, napr print
  + filter očakáva v sebe podmienku typu boolean, čiže nie je to možné použíť na nejaké premapovávanie atď. Treba si uvedomiť že očakáva predikát, ktorý síce vracia objekt ale očakáva ten boolean
  + map je vhodný na mapovanie, čiže možno v ňom robiť lubovolné operácie na premenné ale jeho nevýhoda je tá, že sa mi nepodarilo pomocou nej vrátiť ElementsCollection, asi kvôli zdedenej abstraktnej triede, je možné vrátiť len typy listov(zozbierať ich do kolekcie), nasledovný príkaz preto nie je možné zrealizovať:

listOfUsers.stream().map(element -> element.find("td").getValue()).collect(Collectors.*toCollection*(ElementsCollection::new));

bude to pravdepodobne z dôvodu, že nemožno spraviť novú new inštanciu pre ElementsCollection, pre iný typ premennej to možno takto použiť, v tomto prípade je to použíté ako Class::Constructor, inak sa to dá použiť aj takto Class::method

* Problém pri načítavaní stringu zo súboru:

while ((l = inputStream.readLine()) != null) {  
 query = query.concat(l + "\n");  
}

* + je potrebné si tu uvedomiť, že načítavanie prvého riadku sa vykoná už v podmienke while, preto načítaný riadok potrebujem už v podmienke priradiť premennej l, inak by to nefungovalo korektne
* pri zavedení parametrov do písania sql skriptu pre hibernate som zistil, že ASC a DESC nemôžu byť nastaviteľné ako parametre pre SQL query, preto ich treba pridať jedine cez concat ako text:

Query q =  
 entityManagerUIM.createNativeQuery("SELECT \* FROM ids.\"Users\"\n" +  
 "order by \"UserName\" COLLATE \"C\" "+ordering+"\n" +  
 "limit :users" ,Users.class);

* + takto je to správne, pričom parameter :users sa následne definuje nasledovne:

q.setParameter("users",usersOnPage);

* + prípadne druhý spôsob je použitie otáznika(?) miesto :users a vtedy sa zadáva parameter pomocou indexu(asi od 1):

q.setParameter(1,usersOnPage);

* + treba ešte spomenúť, že v postgre bolo zoradenie iné ako sme zvyknutí, preto sortovanie je závislé od jazyka, ktorý mu možno nastaviť, toto nastavenie vidíme vyššie v query ako COLLATE “C“ (sortovanie na zaklade ASCII znakov)
* Pokiaľ chcem vytiahnuť len jeden stĺpec z tabulky, tak nedávam do query .class, pretože mi ho prerobí na objekt a neviem čo mám následne očakávať, pre vytiahnutie UserId môžem použiť nasledovné dva spôsoby:

int userId;  
entityManagerUIM.getTransaction().begin();  
Query q =  
 entityManagerUIM.createNativeQuery("SELECT \"UserId\" FROM ids.\"Users\"\n"+  
 "where \"UserName\"= :user limit 1");  
  
q.setParameter("user", user);  
userId = (Integer) q.getSingleResult();

alebo tento:

List<Object[]> userId;  
entityManagerUIM.getTransaction().begin();  
Query q =  
 entityManagerUIM.createNativeQuery("SELECT \* FROM ids.\"Users\"\n"+  
 "where \"UserName\"= :user limit 1");  
  
q.setParameter("user", user);  
userId = q.getResultList();  
System.*out*.println(userId.get(0)[0]);

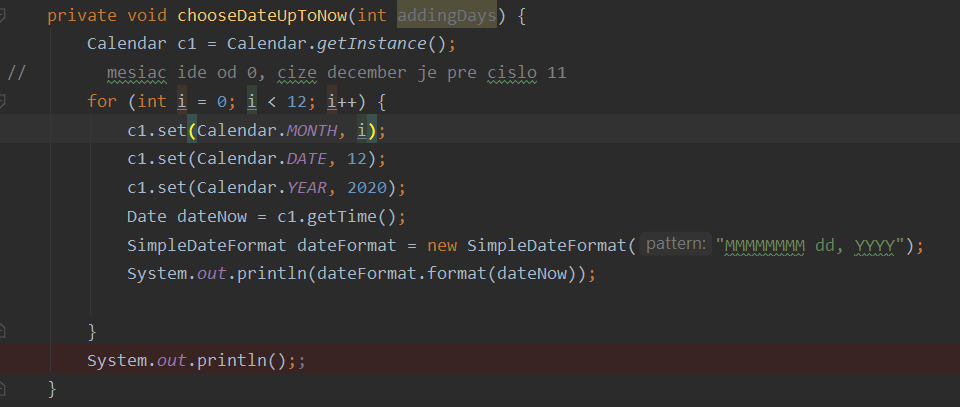
* Pri práci s dátumom v SQL ho treba premeniť na timestamp a pri update nechať query len vykonať, executeUpdate vráti počet zmenených záznamov v int:

int result;  
  
entityManagerUIM.getTransaction().begin();  
Query q =  
 entityManagerUIM.createNativeQuery("Update ids.\"UserSubstitutions\" \n"+  
 "set \"StartDate\"=:startDate, \"FinishDate\"=:finishDate \n"+  
 "where \"OffSiteUserId\"= :offSiteUser and \"DeputyUserId\" = :deputyUser", UserSubstitutions.class);  
q.setParameter("startDate", Timestamp.*valueOf*(startDate), TemporalType.*TIMESTAMP*);  
q.setParameter("finishDate",Timestamp.*valueOf*(finishDate),TemporalType.*TIMESTAMP*);  
q.setParameter("offSiteUser", offsiteUserId);  
q.setParameter("deputyUser", deputyUserId);  
result = q.executeUpdate();

* Pri načítavaní zoznamu elementov napríklad v tabuľke môže spadnúť do chyby, že element not found, je to z dôvodu, že ešte nebola tabuľka načítaná, prípadne element, ktorý zobrazuje page načítaný bol pod nulovou hodnotou a s tou chceme pracovať. Je potrebné pridať podmienku na zmiznutie loading baru
* Pre použitie databázy bolo treba pridať inicializáciu databázy, inak padalo do NullPointerException:

public class OnyxWebUsers {  
  
 public OnyxWebUsers() {  
 *page*(this);  
 //inicializacia, aby som to mohol pouzit aj v inych klasach  
 onyxWebLogin = new OnyxWebLogin();  
 databaseAction = new DatabaseAction();  
 }

* Počas testovania je lepšie zakomentovať v časti after WebDriverRunner.closeWindow(), pretože aj pri konfigurácii Configuration.holdBrowser...=true stránka nezostane otvorená ale zatvorí sa práve kvôli tomuto, počas spúšťania všetkých testov je nevyhnutné odkomentovať, inak by testy nerobil od začiatku ale v otvorenom okne kde prebehol predošlý test
* Práca s dátumom a časom – ak chcem dostať dátum v konkrétnom formáte, môžem využiť Date a SimpleDateFormat:

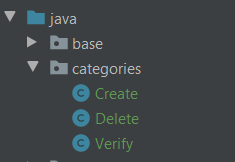


Mesiac v patterne ma nasledujucu syntax:

* M vrati jedno cislo mesiaca
* MM vrati dvojciferne cislo mesiaca
* MMM vrati nazov mesiaca v skratenej forme (3 pismena)
* MMMM vrati cely nazov mesiaca
* Definovanie kategórii testov:
  + do POM je potrebné pridať dependency :



* + Vytvoriť prázdne triedy pre dané kategórie:



* + Zadefinovanie vybraných kategórii v test Suite:
* package suites;  
    
  import categories.Create;  
  import org.junit.experimental.categories.Categories;  
  import org.junit.runner.RunWith;  
  import org.junit.runners.Suite;  
  import tests.OnyxWebUsersTest;  
    
  @RunWith(Categories.class)  
  @Categories.IncludeCategory(Create.class)  
  //Priklad pre definovanie viacerych tried  
  // @Categories.IncludeCategory({Create.class, Verify.class, Delete.class})  
  @Suite.SuiteClasses({  
   OnyxWebUsersTest.class,  
  })  
  public class OnyxCreateTestSuite {  
  }