Overall test structure

UI Automation Test Robustness

Attachments:

Course2.rar

Robustness is the quality of being able to withstand stresses, pressures, or changes in procedure or circumstance.

In order to have a robust test structure, the following are going to be taken into account:

- Control
    - real-time controller
- Modularity
    - Creation of reusable modules in the test structure
    - Isolate some of the components that can be used idependently.
    - Clean-up or recovery components
- Fault-Resistant
    - Error handling and logging
    - Unatended execution
- Syncronization
    - configurable timeouts
- Flexibility
    - parameterized input, verification and configuration data
    - independent internal data model
    - code independent test logic

We had the initial test class, the code generated from the record and playback session:

```
package test1;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import Driver.PageDriver;
public class CopyOfContYahoo {
private String baseUrl;
private StringBuffer verificationErrors = new StringBuffer();

@Before
public void setUp() throws Exception {
 PageDriver.driver = new FirefoxDriver();
 baseUrl = "http://ro.yahoo.com/";
 PageDriver.driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
 }
@Test
public void testContYahoo() throws Exception {
 PageDriver.driver.navigate().to(baseUrl + "/");
 PageDriver.driver.findElement(By.xpath("//span[@class='tab-icon tuc-spr']")).click();
 PageDriver.driver.findElement(By.id("username")).sendKeys("mirimiramoii");
 PageDriver.driver.findElement(By.name("passwd")).sendKeys("Aalndala99");
 PageDriver.driver.findElement(By.id(".save")).click();
 try {
  assertEquals("Bun, ola",
PageDriver.driver.findElement(By.xpath("//div[@id='mediafpwave3']/div/a/em")).getText(
));
 }
 catch (Error e) {
  verificationErrors.append(e.toString());
 }
 PageDriver.driver.findElement(By.xpath("//span[@class='icon']/following::span[contain
s(text(),'Mail')]")).click();
 PageDriver.driver.findElement(By.xpath("//li[@id='Compose']/a/span[contains(text(),'C
ompunere')]")).click();
}
@After
public void tearDown() throws Exception {
 PageDriver.driver.quit();
 String verificationErrorString = verificationErrors.toString();
 System.out.println(verificationErrorString);
 if (!"".equals(verificationErrorString)) {
 fail(verificationErrorString);
 }
}
}
```

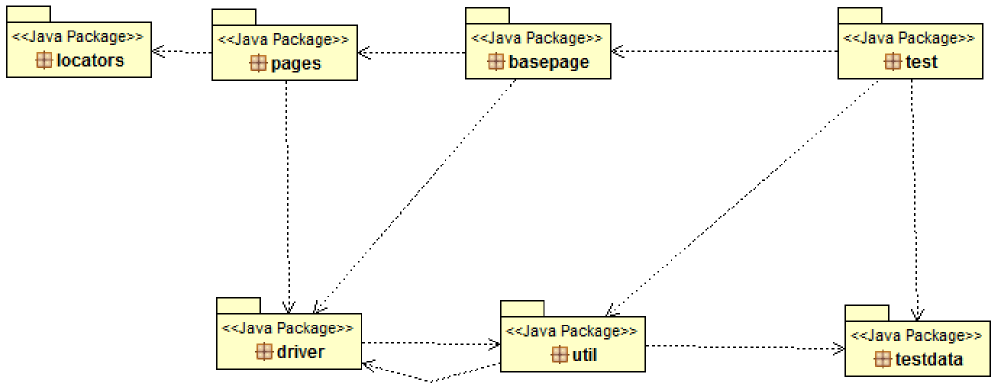
The above code is not reusable at all, and having all the id-s hard-coded in the test it will be hard to be re-used and will take a lot of time to be changed. A better approach on that direction would be that the tests will look like:

```
package test;
import org.junit.*;
import basepage.LoginBasePage;
import static util.Errors.verifyEquals;
import testdata.UsersData;
import util.Setup;
public class SendAMailTest extends LoginBasePage{

@Before
public void setUp() throws Exception {
 Setup.before();
}

@Test
public void testContYahoo_test() throws Exception {
 mainPage.clickSignIn();
 verifyEquals(pageTitle(),"Conectai-v la Yahoo");
 loginPage.login(UsersData.User.ADMIN,UsersData.User.ADMINPASS);
 verifyEquals("Bun, "+UsersData.User.ADMINNAME, mainPage.getLoggedinName());
 mainPage.clickEmail();
 verifyEquals(pageTitle(),UsersData.User.ADMIN+" - Yahoo Mail");
 mailPage.createNewMail();
 verifyEquals(pageTitle(),UsersData.User.ADMIN+" - Yahoo Mail");
}
@After
public void tearDown() throws Exception {
 Setup.after();
}
}
```

Having a structure package diagram looking like:



<<Java Package>> locators
<<Java Package>> pages
<<Java Package>> basepage
<<Java Package>> test
<<Java Package>> driver
<<Java Package>> util
<<Java Package>> testdata