Data Driven testing

1. **What is Data driven testing?**

   Read the data from external recourse & run the test Scripts is called Data driven testing

2. Why Data Driven testing?

   As per the rule of the automation data shouldn't not hardcoded(fixed) with in a test scripts, because data modification & maintenance is tedious job when you want to run the test with different data, instead we should get the data from external resource like Excel, properties file .

3. What are Advantages of Data driven testing

1. Maintenance of the test data is easy
2. Modification of the test data in external recourse is easy .
3. Cross browser /platform testing is easy (means change the browser in property File)
4. Running test scripts in different Environment is easy
5. Running test scripts in different credentials is easy
6. We can create the test data prior the Suite execution (we can also get the data from testData team)
7. Rerunning same test Script with multiple time with different data is easy

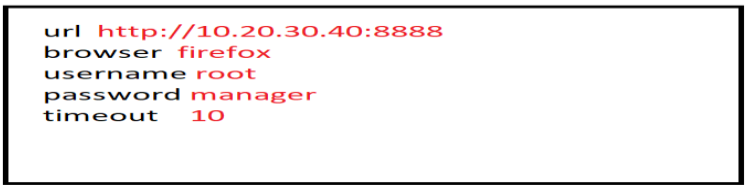
src/test/resources------Rightclick----new----file---Data.properties

Data driven testing from Properties File

1. What is Properties File?

Properties is java feature file where we can store the data in from of key & values pair,

Key & value data type should be always string .



data.properties

2. Why Property file ?

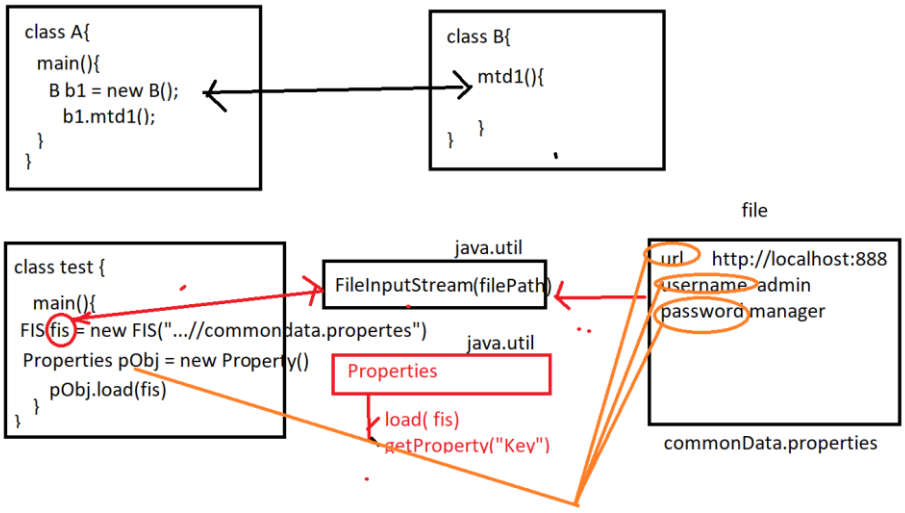
Property file is light weight & faster to read the data compare to any other file , & java as own Class to read the data from property

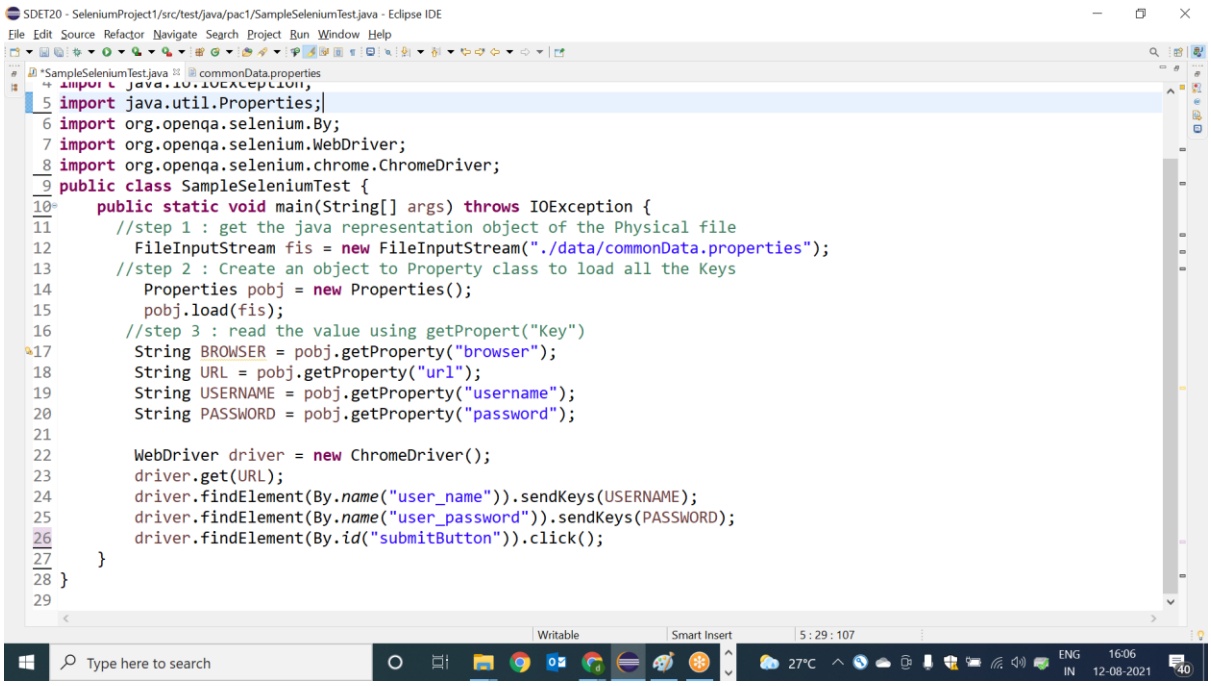
3. How to read data from properties File?

- Get the java representation Object of the Physical file using “FileInputSteam
- Create a Object of “Properties” class & load all the keys
- Read the data using getProperty(“Key”)

Note : properties file light weight & faster in execution compare to Excel

How to create property file in project?
Select src/test/resources right click new—file—name it as Data.prpperties





Data driven testing from Excel File

➔Apache Poi is the open source libraries used to get & write data from all Microsoft documents like Excel .
➔In real time most the company preferred the keep the test script data in Excel, because data will be in well-organized manner , so that modification & maintenance is easier.

Installation steps:

1. Go the Maven Project
2. Edit POM.xml file
3. Go to https://mavenrepository.com
4. Search for apache-POI, POI—OOXML,POI_OOXML-Schemas
5. Copy the dependency
6. Add dependency inside the <dependencies> in POM.xml

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>4.1.2</version>
```

```
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>4.0.0</version>
</dependency>
```

Reading data from excel:

```
1. FileInputStream fis = new FileInputStream("./dataread.xlsx");
2. Workbook wb = WorkbookFactory.create(fis);
3. String url = wb.getSheet("Sheet1").getRow(1).getCell(0).getStringCellValue();
4. String username = wb.getSheet("Sheet1").getRow(1).getCell(1).getStringCellValue();
5. String password = wb.getSheet("Sheet1").getRow(1).getCell(2).getStringCellValue();

6. WebDriver driver = new ChromeDriver();
7. driver.manage().window().maximize();
8. driver.get(url);
9. driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
```

```
10.     driver.findElement(By.id("email")).sendKeys(username);
11.     driver.findElement(By.id("pass")).sendKeys(password);
```

Write Data into the Excel:

```
1. FileInputStream fi=new FileInputStream(".\\src\\test\\resources\\Excel.xlsx");
2. Workbook wb = WorkbookFactory.create(fi);

3. Sheet sh = wb.getSheet("Sheet1");
4. Row ro = sh.createRow(2);
5. Cell c = ro.createCell(1);
6. c.setCellValue("1011 7310 11536")
7. FileOutputStream fi1=new FileOutputStream(".\\src\\test\\resources\\Excel.xlsx");
   wb.write(fi1);
```

POM(Page object Model):

- It is an object repository

Why repository?

As per the role of automation we should not hardcode the element in test scripts, instead we should get elements from object repository, because in a Agile process, due to frequent requirement changes modifications and maintenance of elements is tedious job.

What is object repository/ element repository?

It is a collection of elements locators and business libraries in one place and its shared by multiples.

What is POM Design pattern?

It is a Java design pattern, preferred by Google to maintain elements in well organised way

What is page factory Design pattern?

It is an extended version of POM design pattern, which is used to access the elements available in form.

Advantages of POM?

- We can handle stale element reference exception.
- Reusability of element, no need to write Xpath again and again.

- Modifications in repository is easy, when GUI changes frequently.
- Maintenance of elements is easy because all elements are kept in one place
- More readability

What is stale element reference exception?
Whenever the address of an element got expired we will get stale element reference exception.

There are three stages in POM

1. Declaration
2. Initialization
3. Utilization

Declaration:
We declare the elements through below syntax

@FindBy(locator=”value”)
Private webElement component name;

Initialization :

Here we have to create a constructor shown as below

```
Public Constructor(WebDriver driver)
{
PageFactory.initElements(driver,this);
```

}

Utilization:
We are going to create simple method.

```
Public void demo()
{
componentName.action;

}
```

Rules for POM:

- Class name should be same as page name.
- For every new page we have to create new class
- Except typing and clicking we need to generate getter and setter methods.

TestNG:

- Test NG ---->Test next generation
- It is a unit testing tool used by developers while doing white box testing

Why Testing in development?

- In order to check the source code of the application

Why testing automation?

- We convert every test script into .XML file In order to perform different kinds of execution, for this we need testing**.**

TestNG is a automation testing framework

Advantages of TestNG

1. We can Give priority to the test scriptss
2. We can skip the test scripts execution
3. We can give invocation count
4. Report will be generated
5. We have annotations in testing
6. We can achieve sequential execution of the program
7. We can achieve parallel execution
8. We can achieve group execution
9. we can do assertions
10. we can re run the failed test cases
11. We have special annotation @Parameter
12. We have annotation @Data provider

Strcture of Java

```
Public class A{
Public static void main(Strings[]args){
}

}
```

Structure of TestNg

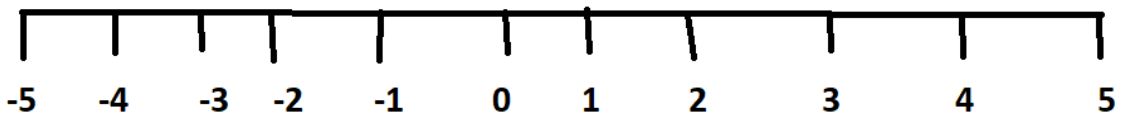
```
Public class A{

@Test
Public void demo(){

}

}
```

Note:

- If a class contains @Test that is a TestNG class
- We can have multiple @Test methods inside a single class
- Every @Test in a class we will called as test script

Priority:

- Importance given to the test script is called as priority
- By default priority is zero
- If two test cases are having same priority then it will execute in alphabetical order (first priority goes to uppercase)
- Priority of the execution always given to the lowest value



First the negative numbers
Second to zero
Third positive numbers

```java
package TestNG;

import org.testng.annotations.Test;

Run All
public class Priority_TestNg {

    @Test(priority = -1)
    Run | Debug
    public void test1() {
        System.out.println("Hi test1");

    }

    @Test(priority = 1)
    Run | Debug
    public void test2() {
        System.out.println("Hi test2");

    }

    @Test(priority = -6)
    Run | Debug
    public void test3() {
        System.out.println("Hi test3");

    }

}
```

Output:

```
<terminated> Priority_TestNg [TestNG] C:\Program Files\Java\jdk-11.0.16.1\bin\javaw.exe  (28-Jul-2023, 8:00:53 AM – 8:00:55 AM) [pid
[RemoteTestNG] detected TestNG version 7.4.0
Hi test3
Hi test1
Hi test2
PASSED: test3
PASSED: test1
PASSED: test2

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

Invocation count

- In order to run the same test scripts more than one time we use invocation count
- Default Invocation count is one
- If we have given Invocation count in negative value then that will keep the execution of the particular method

```java
package TestNG;

import org.testng.annotations.Test;

Run All
public class Priority_TestNg {

    @Test
    Run | Debug
    public void test1() {
        System.out.println("Hi test1");

    }

    @Test
    Run | Debug
    public void test2() {
        System.out.println("Hi test2");

    }

    @Test(priority = -1,invocationCount=2)
    Run | Debug
    public void test3() {
        System.out.println("Hi test3");

    }

}
```

OutPut:

```
[RemoteTestNG] detected TestNG version 7.4.0
Hi test3
Hi test3
Hi test1
Hi test2
PASSED: test3
PASSED: test3
PASSED: test1
PASSED: test2

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 4, Passes: 4, Failures: 0, Skips: 0
===============================================
```

Skip the test case execution

- We can skip the execution of particular test scripts by giving enable =false

TestNG Annotations:

- Test NG annotations are used to control the execution flow of the program
- We have different types of annotation such as below

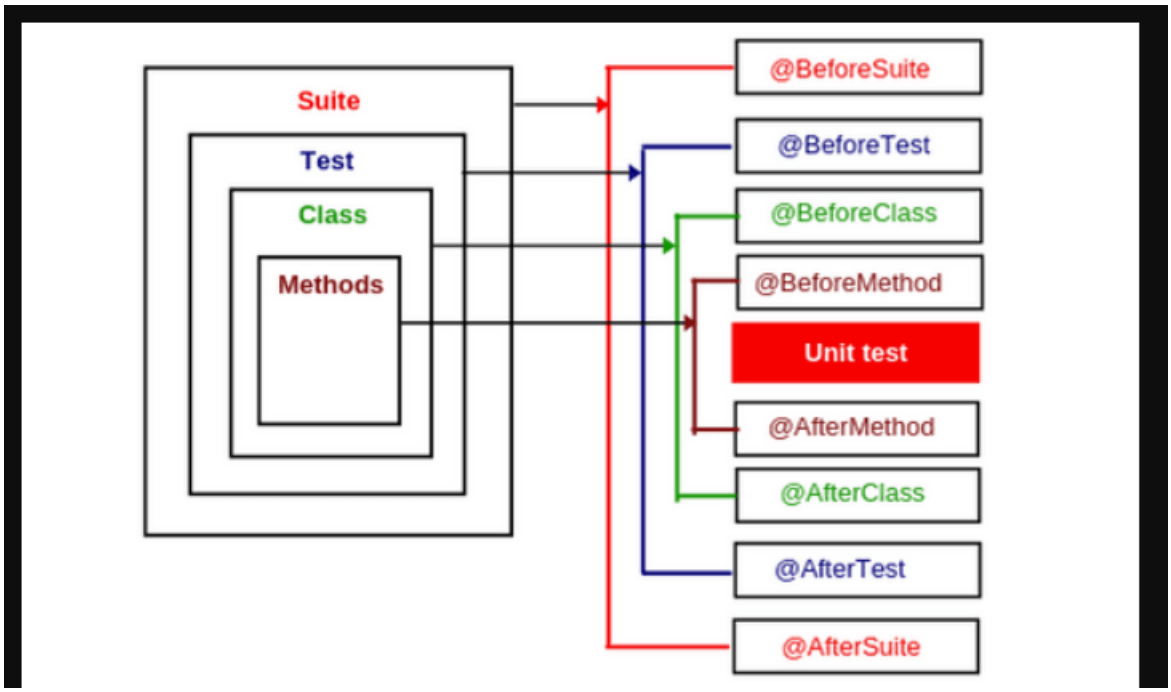
@ Before suite
@ Before test
@ Before class
@ Before method
@ Test

@ After method

@ After class

@ After test

@After suit



Eg:1

2@Test in----1 class

@BS

@BT

@BC

@BM

@Test1

@AM

@BM

@Test2

@AM

@AC

@AT

@AS

Eg:2

2classes---1@Test in each

@BS
@BT
@BC
@BM
@Test
@AM
@AC

@BC
@BM
@Test
@AM
@AC
@AT
@AS

Eg:3

2 Test 1 class and 1 @ Test in each class

@BS
@BT
@BC
@BM
@Test
@AM
@AC
@AT
@BT
@BC
@BM
@Test
@AM
@AC
@AT
@AS

How to generate the report

Only after execution right click on the project refresh. Expand test output folder----> right click on emailable report. html----> open with----> web browser

Sequential execution/Batch execution

- Select all the test cases(which and all we want to have batch execution) right click----> testNG---> convert to testNG give the name----->finish.
- Click on respective .xml file. Click on Run button,
- So that we can achieve Batch execution.

Suite file for batch execution:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-
1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="TestNG.Priority"/>
      <class name="TestNG.Invocationcount"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Parallel execution

Types of parallel execution

- Distributed parallel execution
- Compatibility parallel execution

Distributed parallel execution :

If we have 100 test cases and if we run them in Chrome and it takes 10 minutes of time to complete the execution, in order to reduce the execution time we can go for distributed parallel execution. so in this we have divided 50 test cases in Chrome and 50 test cases another Chrome like that..

Suite.xml file for Distributed parallel execution :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-
1.0.dtd">
<suite name="Suite" parallel="tests">
    <test thread-count="5" name="Test1">
        <classes>
            <class
name="ScrollBar_TestNG.Scroll_Bar_Coordinates" />

        </classes>
    </test> <!-- Test -->


    <test thread-count="5" name="Test2">
        <classes>

            <class
name="ScrollBar_TestNG.Scroll_Skillrary_Carrers" />
        </classes>
    </test> <!-- Test -->
```

```
</suite> <!-- Suite -->
```

Compatibility parallel execution:/Cross Browser Execution:

- Running the same test cases on multiple browsers at same time parallelly is known as compatibility parallel execution
- We can achieve this with the help of @parameters

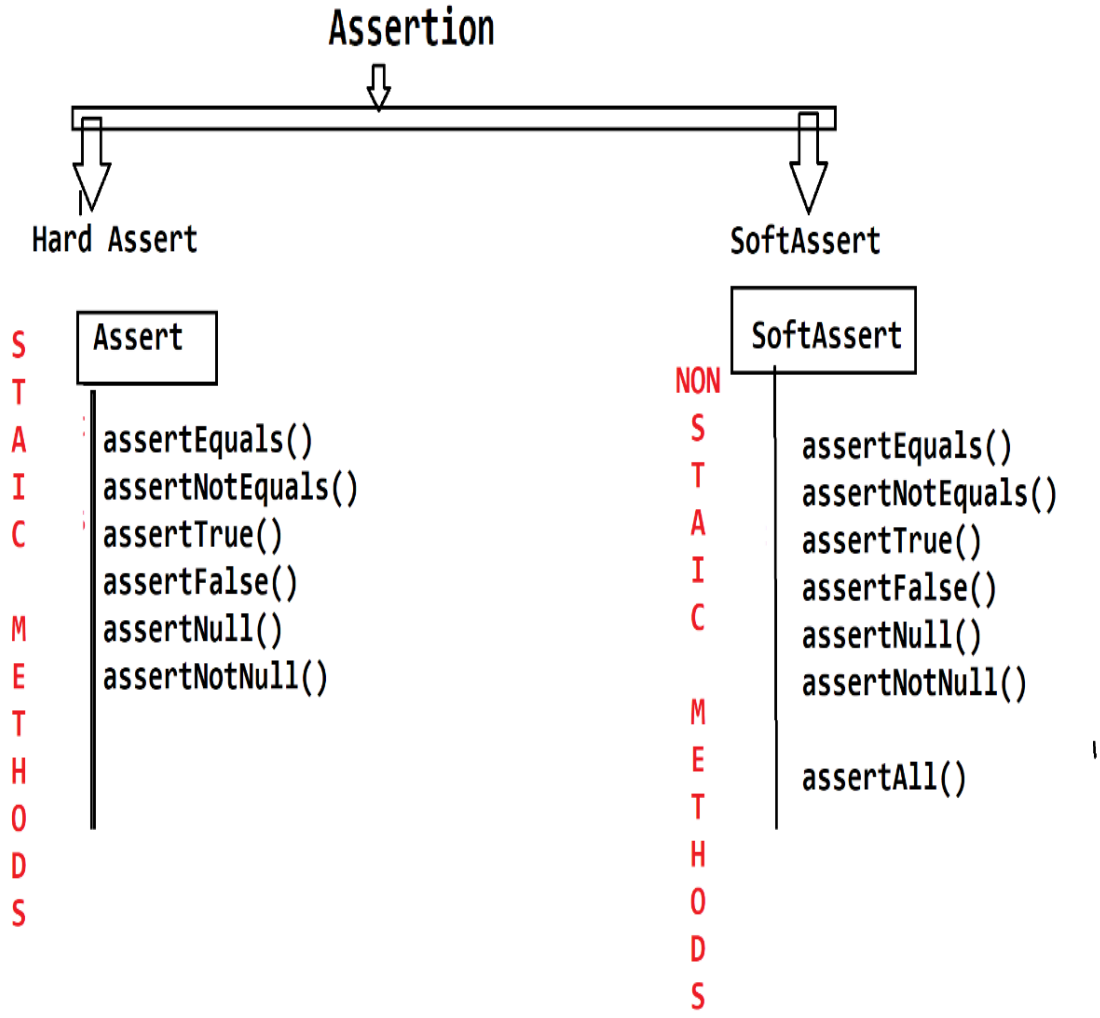
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-
1.0.dtd">
<suite name="Suite" parallel="tests">
  <test thread-count="5" name="Test1">

    <parameter name="browsername"
value="chrome"></parameter>
    <classes>
      <class
name="ScrollBar_TestNG.Scroll_Skillrary_Carrers"/>
    </classes>
  </test> <!-- Test -->

  <test thread-count="5" name="Test2">
  <parameter name="browsername"
value="firefox"></parameter>
    <classes>
      <class
name="ScrollBar_TestNG.Scroll_Skillrary_Carrers"/>
    </classes>
  </test> <!-- Test -->

</suite> <!-- Suite -->
```

We can achieve group execution:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-
1.0.dtd">
<suite name="Suite">
<groups>
<run>
<include name="smoke"></include>

</run>


</groups>

  <test thread-count="5" name="Test">
    <classes>
      <class name="TestNG_Group.Invocation"/>
      <class name="TestNG_Group.Invocation_1"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Assertions:

- Assertion is a feature available in TestNG used to validate test scripts expected results
- As per the Rule of the automation every expected result should be verified with Assert statements, because java “if else “statement will not have capability to fail the testNG test
    - Scripts
- There are 2 types of Assertions in TESTNG


Assertion
Hard Assert
SoftAssert
Assert
STATIC METHODS
assertEquals()
assertNotEquals()
assertTrue()
assertFalse()
assertNull()
assertNotNull()
SoftAssert
NON STATIC METHODS
assertEquals()
assertNotEquals()
assertTrue()
assertFalse()
assertNull()
assertNotNull()
assertAll()

Hard Assertion	Soft Assertion
All methods are static in nature	All methods are non-static in nature
It does not allow further execution of test if the line containing hard assert gets failed.	Next steps would be executed even if the line containing soft assertion gets failed.
Whole test case gets failed if at least 1 hard assert fails.	AssertAll() extra lines of code are required to track the fail status.
To verify mandatory fields we go for hard assert	To verify non mandatory fields we go for soft assert

How to Re run the failed testcases:

- Correct the mistake in particular program
- Refresh the project
- Expand test.output folder
- Click on testNG.failed.xml ( Failed_TestCase.java) file and run that suite fail.

@Data provider:

- Whenever we want to execute same test script with different data then we will go for this @Dataprovider