**ASSIGNMENT-14.1**

1. Create a calculator to work with rational numbers.

*Requirements:*

It should provide capability to add, subtract, divide and multiply rational

numbers

Create a method to compute GCD (this will come in handy during operations on

rational)

Add option to work with whole numbers which are also rational numbers i.e. (n/1)

- achieve the above using auxiliary constructors

- enable method overloading to enable each function to work with numbers and rational.

**SOLUTION:**

In Scala, We can define Auxiliary Constructors like methods by *using “def” and “this”* keywords. “this” is the constructor name.

Auxiliary Constructor is also known as Secondary Constructor. A Scala class can contain zero or one or more Auxiliary Constructors.

*Auxiliary Constructors are used to provide Constructors Overloading.*

In the below screenshot,

Line no: 12 is an auxiliary constructor.

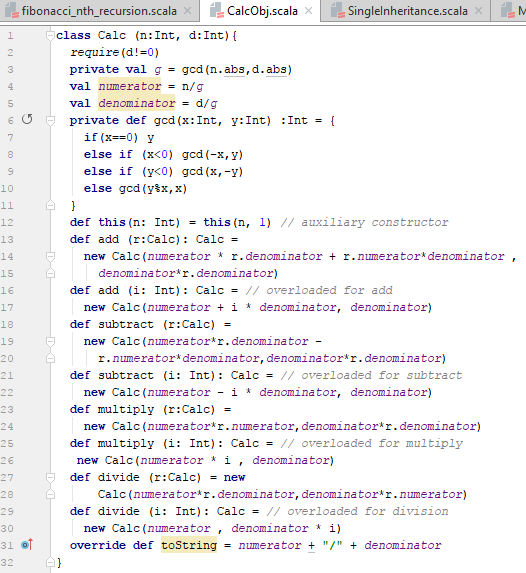
**def this**(n: Int) = **this**(n, 1)

Here, this is a one argument parameterized auxiliary constructor. As an auxiliary constructor should have the call to primary or previous auxiliary constructor, here it invokes the previous constructor as **this**(n, 1). The call should always be the first statement in the constructor.

Also, each of the logic for an input as an integer as well as rational number is defined and the methods add, subtract, multiply and divide are overloaded to handle the necessary scenarios.

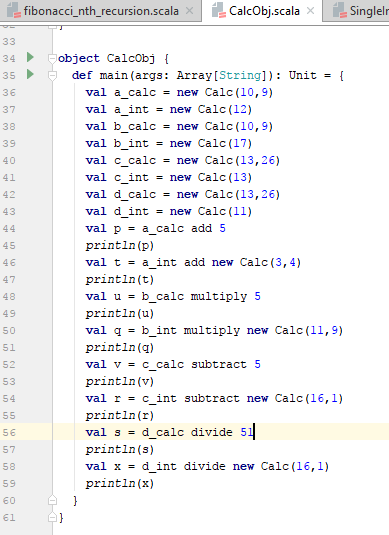
**def** add (r:Calc): Calc =  
 **new** Calc(*numerator* \* r.*denominator* + r.*numerator*\**denominator* ,  
 *denominator*\*r.*denominator*)  
**def** add (i: Int): Calc = **new** Calc(*numerator* + i \* *denominator*, *denominator*)  
**def** subtract (r:Calc) =  
 **new** Calc(*numerator*\*r.*denominator* -  
 r.*numerator*\**denominator*,*denominator*\*r.*denominator*)  
**def** subtract (i: Int): Calc =

**new** Calc(*numerator* - i \* *denominator*, *denominator*)  
**def** multiply (r:Calc) =  
 **new** Calc(*numerator*\*r.*numerator*,*denominator*\*r.*denominator*)  
**def** multiply (i: Int): Calc = **new** Calc(*numerator* \* i , *denominator*)  
**def** divide (r:Calc) = **new** Calc(*numerator*\*r.*denominator*,*denominator*\*r.*numerator*)  
**def** divide (i: Int): Calc = **new** Calc(*numerator* , *denominator* \* i)



Each of the methods are invoked be passing the parameters at compile time itself.

Below is the screenshot for the same.



Considering, one example from the above screenshot for add, it works in the below way:

**val** a\_calc = **new** Calc(10,9)

**val** p = a\_calc add 5  
*println*(p)

**Logic: new** Calc(*numerator* \* r.*denominator* + r.*numerator*\**denominator* ,  
 *denominator*\*r.*denominator*)

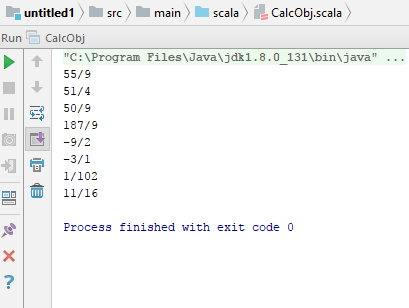
val p = (10,9) add 5

=(5\*9 + 10\*1 , 1\*9)

=(45+10,9)

=(55,9)

As per the below screenshot of outputs, the first one is explained above:



Hence, the calculated result matches the expected result.

Below is the attached file for the same:

