Unit testing:

1.Test analysis and design. (more humanly + sometimes done with ai)

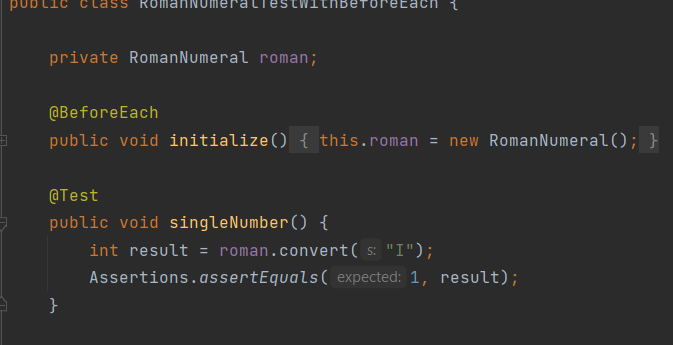
2. Executing test. (automated)

Junit:

@Test = test case function

@BeforeEach = run before each test case.

Assertions.Asserteqals()= Assertion means to assume.

****

**Common Vocabularies:**

Failure**:** Software or system behaving in a way that is not expected.

Defect: Failure is caused by Defect( smtimes called bugs or faults)

It is an incorrect statement or flaw that is causing that behaviour.

Error: incorrect behavior that causes incorrect behavior. Eg. We forgot a corner case.

Verification: Are we making the system in the right way?( is everything bug free)

Validation : Is the system doing what it needs to do? (are we building right software)

Dijkastra says “Testing shows the presence of defect but doesn’t ensure absence of defect”

Testing is all about prioritizing.

Getting started with functional testing:

Boundary test example:

Leap year code test: if(year%400==0) yes; else if (year%4==0 and year%100!=0)yes;else no;

Test example:

2016 (divis by 4but not by 100 and 400)

2400(divis by 400 and divis by 4 and not divs by 100)

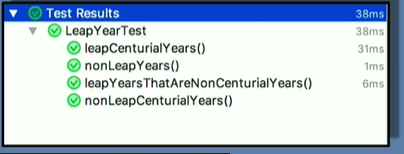
1900(div by 4 and div by 100 but not by 4000)

1901(neithier)

So in testing leap year it will be enough if we do only these 4 tests. They are called equivalent partitions. All tests in one partitions is equivalent.

Eg: for partition (divis by 4but not by 100 and 400) we will test 2016/2024/2020 all of them will give same result.

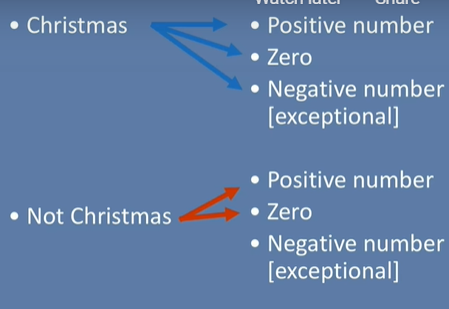
So in junit test our test method name will be:



So we will write the automated tests for the program. So if the programmer changes the code and introduce a bug, we can catch it without doing anything next time.

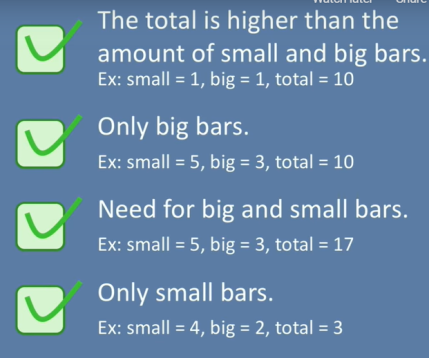
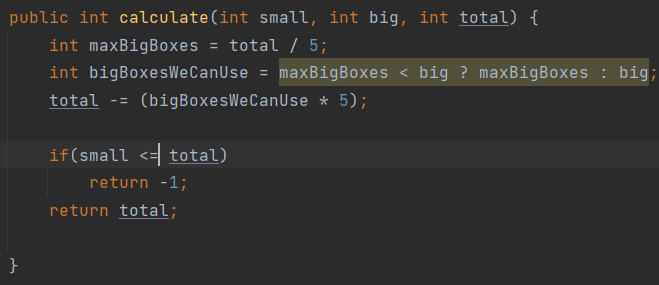
Category Partitioning:

Find the characteristics(from requirement eg.Num is +ve or intuitive input can be null) of all the parameters in the system. Find the all possible Cartesian combinations of them. But this can grow big , so what we can do is to test smart and test same type of things once.



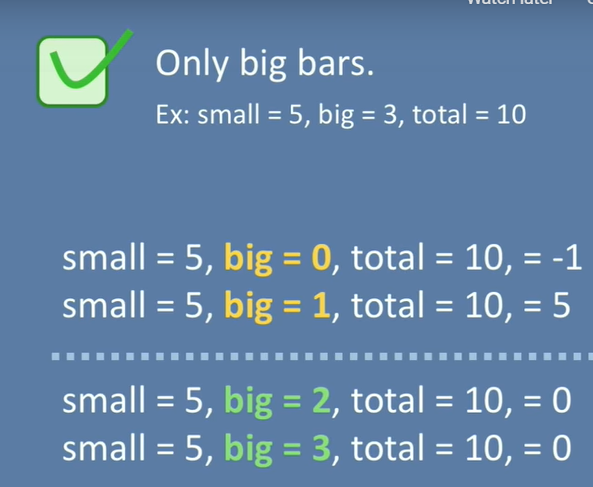
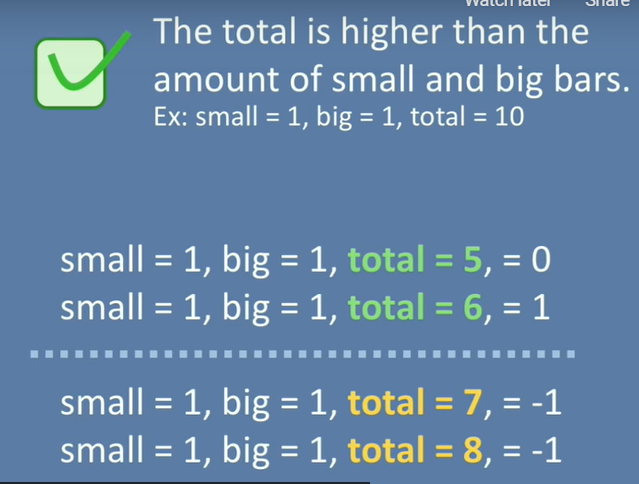
Testing (Christmas,exception) will be enough to know whether raw item can take negative number or not.

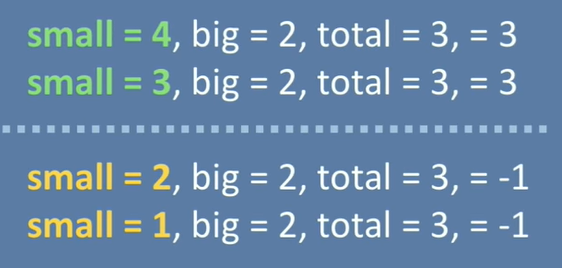
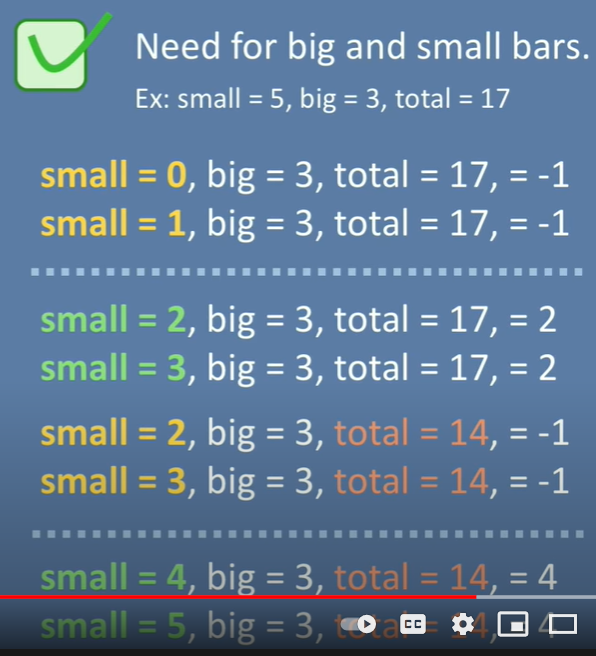
Chocolate bar example test:



But the partition is not enough only, eg the code will give error for (2,3,17) though it is in the partition :

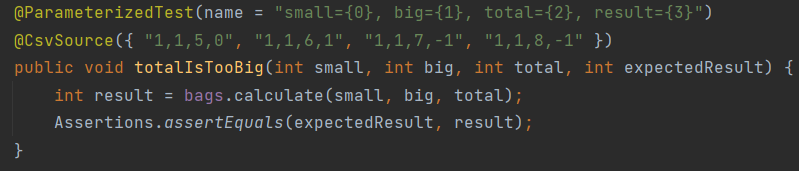
That means checking only (5,3,17) is not enough here. Thus we should check the boundaries since the programmers tend to do more errors in these places.





We’ll use parameterized test in junit to test some inputs in one partition with one test method.

To do this we’ll use @parameterizedtest and @csvsource



Boundary analysis:

For if(x>=0) return true; else return false;

On point: Point of split in the condition. Eg. X=100

Off point: Closest value near one point that alters the result. Here x=99

In point : values that evaluate true.

Out point: values that evaluate false.

So every software system we have to test for all these boundary points.

Now how to pick the boundaries for multiple conditions?

A research paper called “A simplified domain testing strategy” says : test every parameter independently.

While testing one parameters for it’s boundaries we’ll use varying “in points” in the other parameters.

This can be done using a domain matrix:

