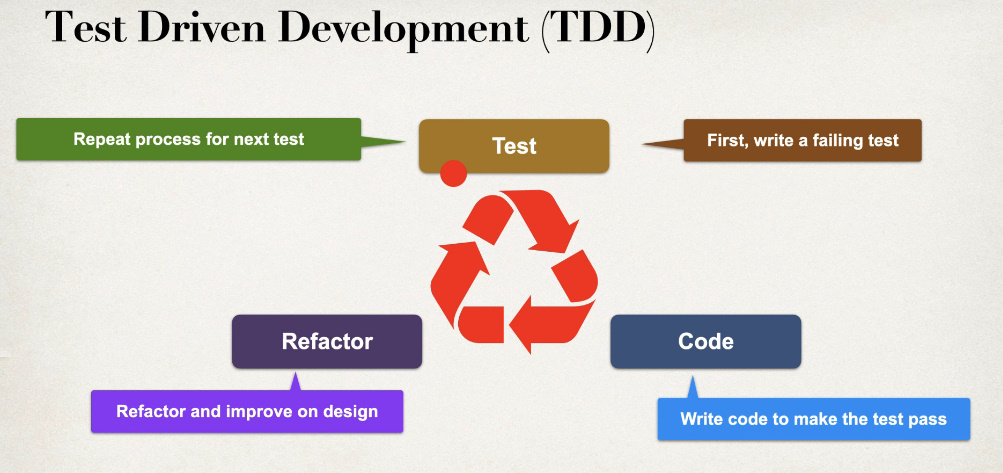
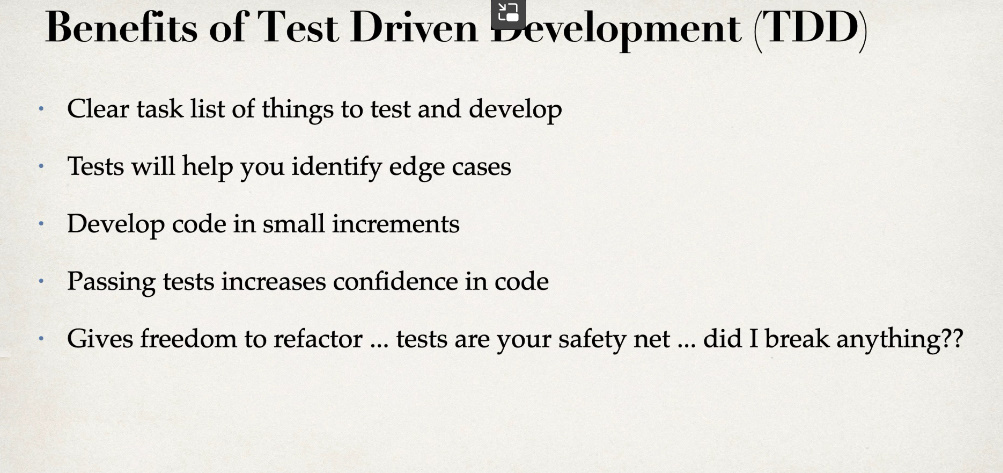


**Test Driven Development** – este o practica de dezvoltare software in care testele sunt scrise inainte ca codul aplicatiei sa fie complet scris, bazandu-se pe elementele test,code si refactor

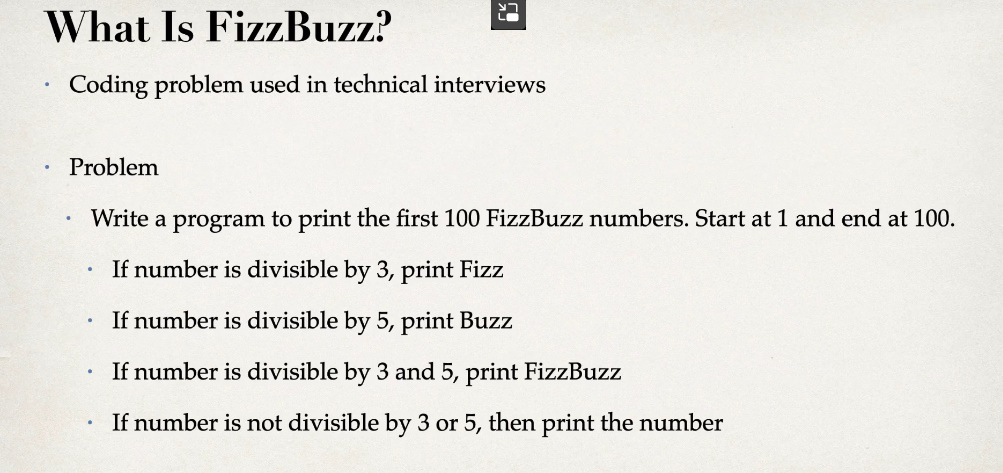


Decim Test Driven Development are urmatoarele principii:

1. Scriem testul care cade
2. Scriem un cod in asa fel ca testul sa nu cada
3. Imbunatatim designul daca e necesar, caci odata ce deja avem testele, putem verifica daca modificand codul nu am stricat ceva
4. Se repeta acelasi ciclu pentru urmatoarele teste



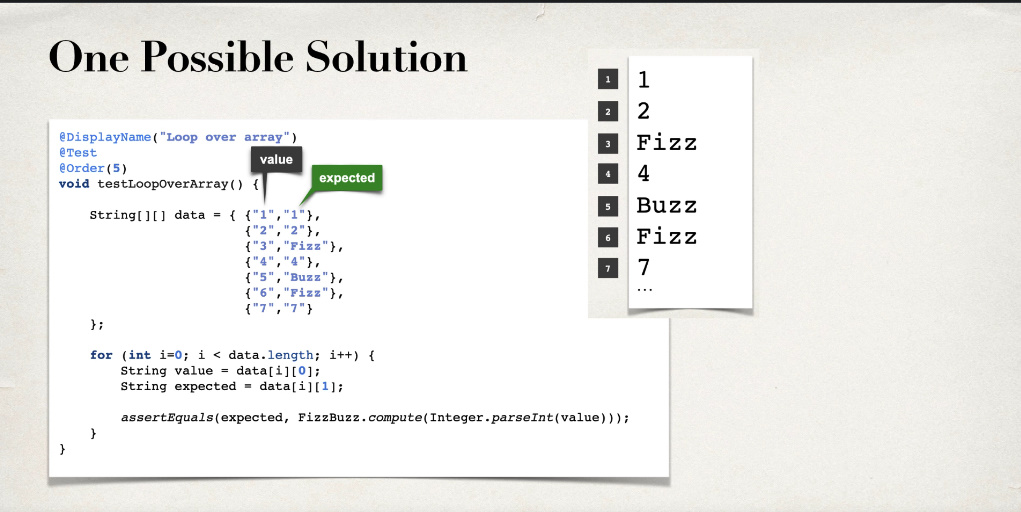
**Exemplu**



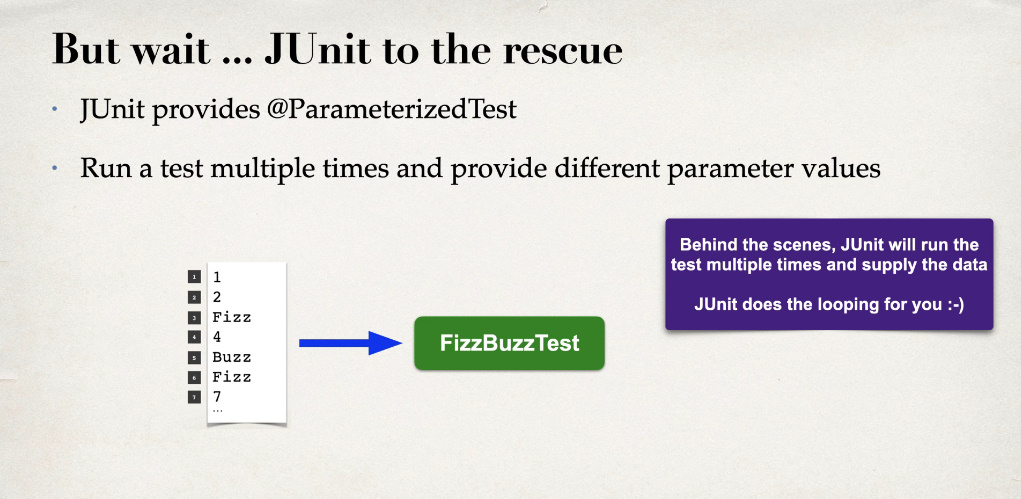
* **Assertions.fail(“mesaj”)** – este o metoda care forteaza un test sa cada

**Parameterized Tests**

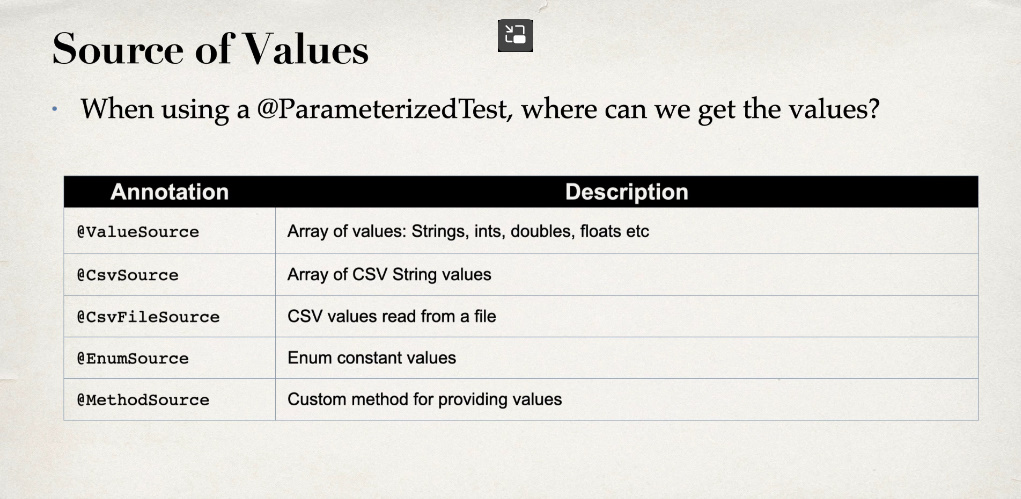
* Nu e prea comod sa tot cream cate un test pentru fiecare valoare dorita.
* Am putea folosi o collection cu valorile oferite si cele asteptate
* Asa, putem rula acelasi test intr-un loop
* assertMethod() poate fi pus si intr-un loop:



* Dar,nu vom avea raport pentru fiecare valoare,deci daca una cade, cade tot testul, dar Junit ofera o solutie mai buna:
* Putem folosi mai bine **@ParameterizedTest**
* **ParameterizedTest se bazeaza pe aceea ca dam o sursa de valori si rezultate ce se trimit parametrilor din metoda si se testeaza intr-un loop creat de Junit, fiecare pereche fiind chiar un test separat.**
* **Nu folosim @Test cu @ParameterizedTest, caci ea inlocuieste @Test**
* Asa, Junit ruleaza un test de mai multe ori si ofera diferite valori parametrizate
* In spatele scenei, Junit va rula testul de mai multe ori pentru noi, exact asa ca mai sus



Dar, mai trebuie o anotatie pentru sursa si rezultate

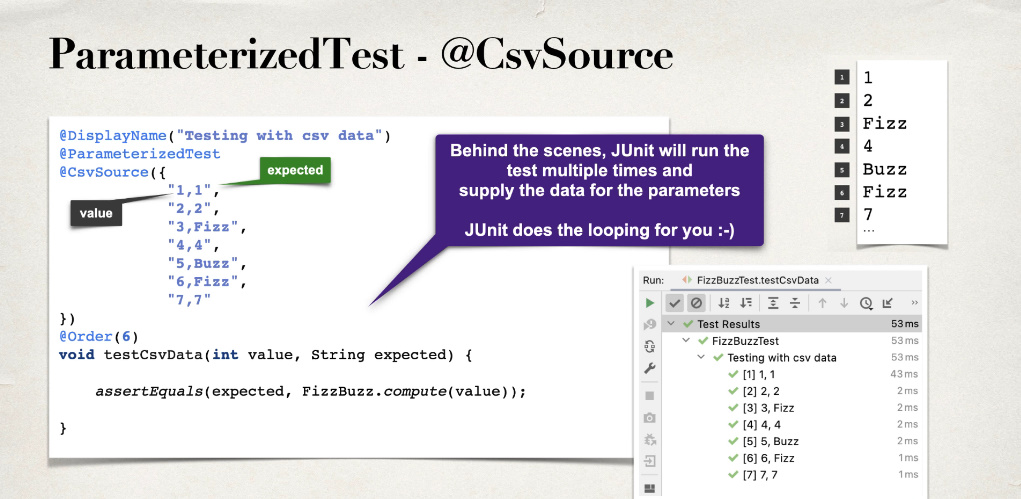


CSV – comma separated values, De ex:

String[] array = {“1,2”,”2,2”,”5,10”}

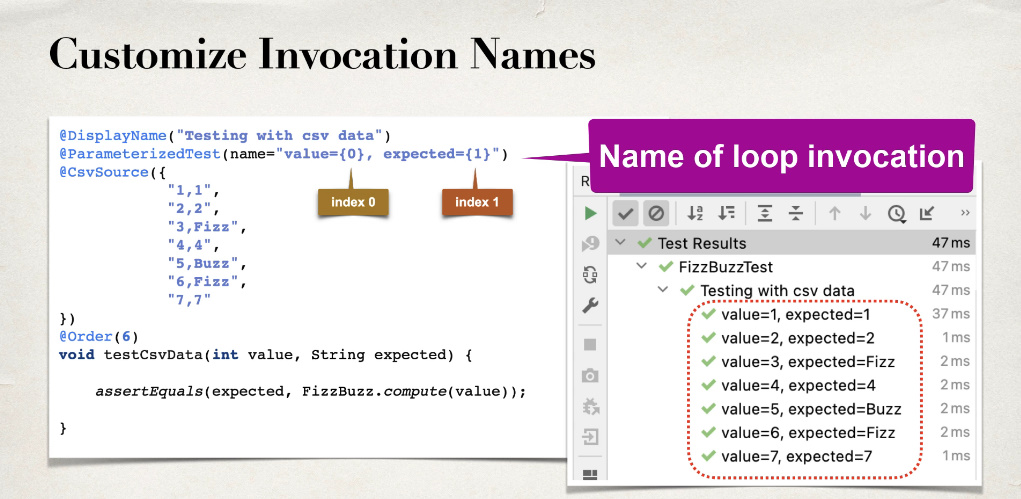
Anume virgula va delimita valoarea data de rezultat

**@CsvSource({...})**

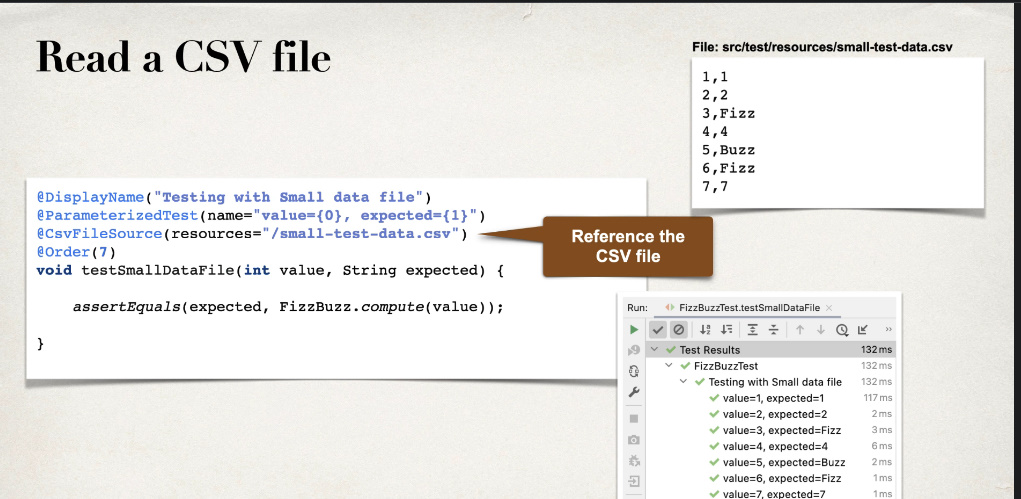


Junit face conversia din int in string si invers daca e necesar

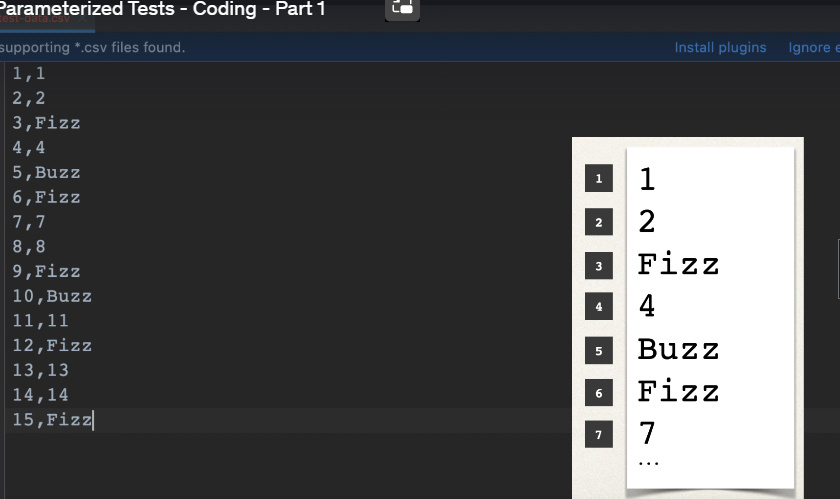
Junit automat va crea un loop ce sa execute metoda de cate ori e necesar, si va pune valorile din tabel in parametrii in ordinea aparitie lor, fiecare executie a loopului fiind un test separat.



**@CsvFileSource(resources=”adresa fisier”)**

* Avem nevoie de un fisier cu extensia .csv
* Totul se face ca si cu @CsvSource, adica Junit atribuie valorile la parametri in ordienea lor in fiecare loop creat pentru fiecare valoare
* 

In acest fisier .csv putem nu trebuie sa punem “ “



**Atentie, in folderul test nu se foloseste classpath, deci pentru a arata ca un fisier e din resources, in fata lui punem /, adica asa:**

@CsvFileSource(resources = "/test.csv")

**@ParameterizedTest(name=”val1={0}, val2={1}”)**

* prametrul **name** din anotatie face ca sa apara un mesaj pentru fiecare test din loop, exact ca @DisplayName, doar ca putem pune valorile testate in mesaj. Deci, fiecare test va avea numele: “val1 = ..., val2=...”

{pozitie} – e pozitia valorii din pereche,sau parametru, deci 0 e prima valoare, 1 e a doua si tot asa daca avem mai multe, desi 2 se folsoesc adesea: valoare si rezutat dorit

**@ValueSource(tip+s={“”,””,””})**

One of the limitations of value sources is that they only support these types:

* *short* (with the *shorts* attribute)
* *byte* (*bytes* attribute)
* *int* (*ints* attribute)
* *long* (*longs* attribute)
* *float* (*floats* attribute)
* *double* (*doubles* attribute)
* *char* (*chars* attribute)
* *java.lang.String* (*strings* attribute)
* *java.lang.Class* (*classes* attribute)
* O alta problema e ca putem crea doar un array unidimensional, nu putem crea array bidiomensional

@Order(8)  
@DisplayName("Test value source")  
@ParameterizedTest(name = "value={0}, expected={1}")  
@ValueSource(strings = {"a","B","ABC","Aabf","ddfdfd"})  
public void valueSourceTest(String s){  
 *assertTrue*(Character.*isLowerCase*(s.charAt(0)));  
}

**@MethodSource(“metoda”)**

@Order(9)  
@DisplayName(value = "Test method source")  
@ParameterizedTest(name = "value= {0}, expected={1}")  
@MethodSource("getList")  
public void methodSourceTest(int i,String e){  
 *assertEquals*(e,FizzBuzz.*compute*(i));  
}  
static String[][] getList(){  
 return new String[][]{{"1","1"},{"2","2"},{"3","Fizz"},{"4","4"},{"5","Buzz"}};  
 }

tot putem folosi un array bidimensional.