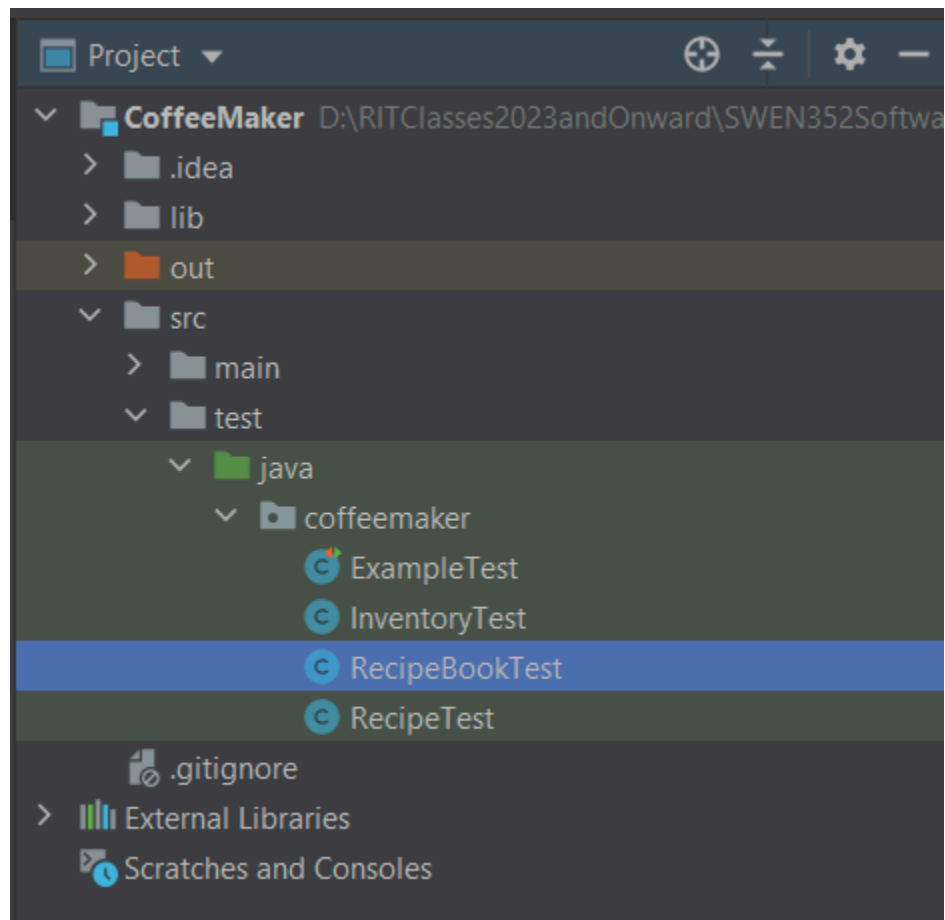
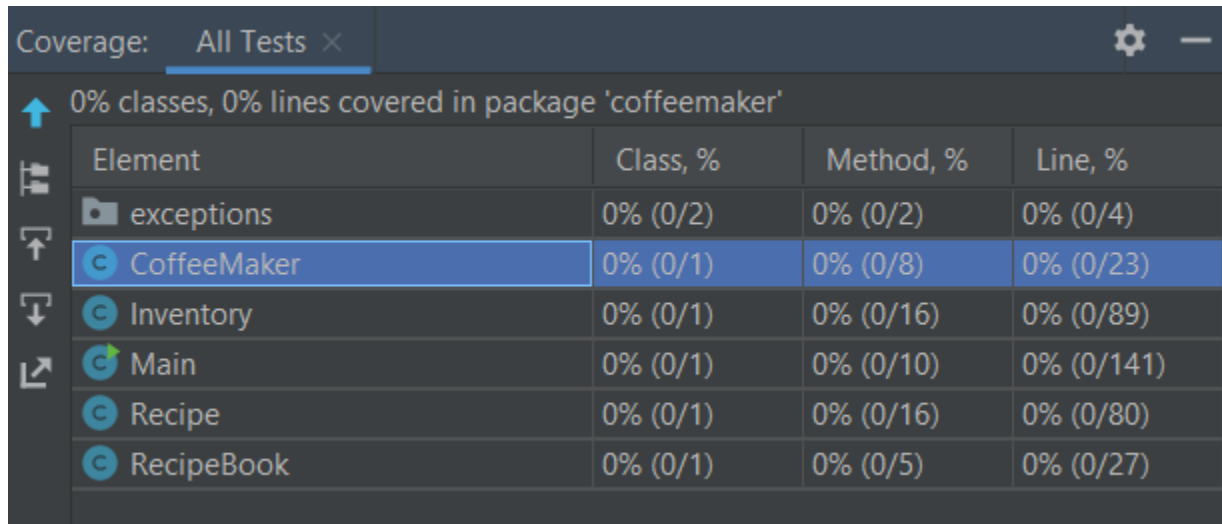


Deliverable 1:



Deliverable 2:



Coverage: All Tests ×

0% classes, 0% lines covered in package 'coffeemaker'

Element	Class, %	Method, %	Line, %
exceptions	0% (0/2)	0% (0/2)	0% (0/4)
CoffeeMaker	0% (0/1)	0% (0/8)	0% (0/23)
Inventory	0% (0/1)	0% (0/16)	0% (0/89)
Main	0% (0/1)	0% (0/10)	0% (0/141)
Recipe	0% (0/1)	0% (0/16)	0% (0/80)
RecipeBook	0% (0/1)	0% (0/5)	0% (0/27)

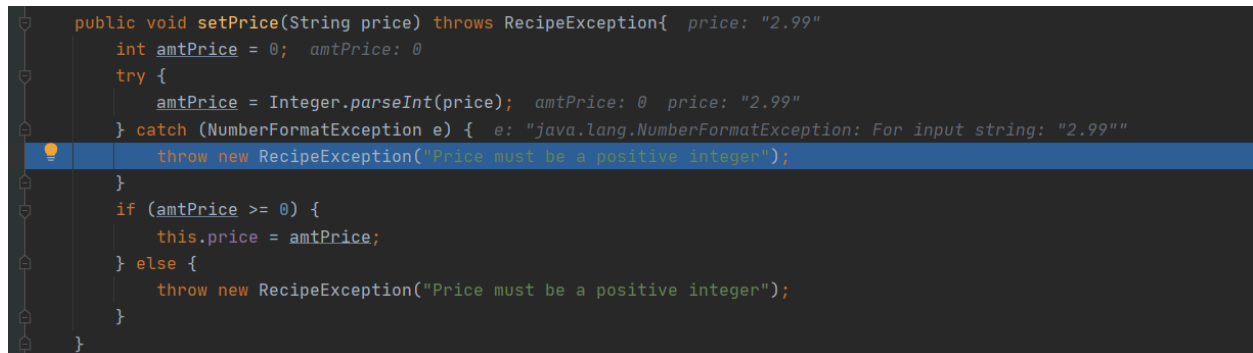
Deliverable 3:

Defect 1: When adding a recipe and inputting price, using a decimal will give an error and say that it needs a positive integer

Class Defect Is In: Recipe

Function/Method Defect is in: setPrice(String price)

ScreenShot Of Offending Source Code:



```
public void setPrice(String price) throws RecipeException{ price: "2.99"
    int amtPrice = 0; amtPrice: 0
    try {
        amtPrice = Integer.parseInt(price); amtPrice: 0 price: "2.99"
    } catch (NumberFormatException e) { e: "java.lang.NumberFormatException: For input string: "2.99""
        throw new RecipeException("Price must be a positive integer");
    }
    if (amtPrice >= 0) {
        this.price = amtPrice;
    } else {
        throw new RecipeException("Price must be a positive integer");
    }
}
```

Explanation Of Defect:

As seen when running the debugger above, inputting a decimal fails because `Integer.parseInt()` is being used when we should be using `Double.parseDouble()`. This defect goes against expected input. Prices in the real world are decimals and not integers. Also, it doesn't prompt the user to enter an integer when entering price.

JUnit code catching defect:

```
public class RecipeTest {

    @Test
    void testSetPrice() throws RecipeException {
        Recipe recipe = new Recipe();
        recipe.setPrice("3.99");
        assertEquals("expected: 3.99, recipe.getPrice()");
    }
}
```

Tests failed: 1 of 1 test – 84 ms

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ...

coffeemaker.exceptions.RecipeException: Price must be a positive integer

at coffeemaker.Recipe.setPrice(Recipe.java:144)
at coffeemaker.RecipeTest.testSetPrice(RecipeTest.java:13) <1 internal call>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Process finished with exit code -1

Updated source code with defect fixed:

```
/**
 * @param price The price to set.
 */
public void setPrice(String price) throws RecipeException{
    double amtPrice = 0.0;
    try {
        amtPrice = Double.parseDouble(price);
    } catch (NumberFormatException e) {
        throw new RecipeException("Price must be a positive integer");
    }
    if (amtPrice >= 0.0) {
        this.price = amtPrice;
    } else {
        throw new RecipeException("Price must be a positive integer");
    }
}
```

Test Method Passing:

The screenshot shows an IDE with a project structure on the left including CoffeeMaker, Inventory, Main, Recipe, and RecipeBook. A test file 'RecipeTest.java' is open, showing a test method 'testSetPrice()' that creates a Recipe object, sets its price to '3.99', and asserts that the price is 3.99. The test passes. The bottom panel shows the test results: 'Test Results' (52 ms), 'RecipeTest' (52 ms), and 'testSetPrice()' (52 ms). The process finished with exit code 0.

```
@Test
void testSetPrice() throws RecipeException {
    Recipe recipe = new Recipe();
    recipe.setPrice("3.99");
    assertEquals("expected: 3.99, recipe.getPrice()");
}
```

Tests passed: 1 of 1 test – 52 ms

Test Results 52 ms

RecipeTest 52 ms

testSetPrice() 52 ms

Process finished with exit code 0

Defect 2: Inventory, line 220

When trying to make a recipe with an inventory, the amount of coffee required is added to the inventory instead of subtracted from. It can easily be fixed just by changing the addition sign to a subtraction sign.

Before:

The screenshot shows the 'Before' state of the InventoryTest.java file. The test method 'testRemoveIngredients()' is shown, which creates a Recipe object, sets its ingredients, and asserts that the inventory has the correct amounts of ingredients. The test fails because the inventory has the wrong amounts of ingredients.

```
public class InventoryTest {
    @Test
    void testRemoveIngredients() {
        Recipe r = new Recipe();
        Inventory inventory = new Inventory();
        try {
            r.setAmtChocolate("5");
            r.setAmtCoffee("5");
            r.setAmtMilk("5");
            r.setAmtSugar("5");
        } catch (Exception e) {
            assert(false);
        }
        assert(inventory.useIngredients(r));
        assert(inventory.getChocolate() == 10);
        assert(inventory.getSugar() == 10);
        assert(inventory.getCoffee() == 10);
        assert(inventory.getMilk() == 10);
    }
}
```

Tests failed: 1 of 1 test – 32 ms

java.lang.AssertionError: (true) expected: false

at coffeeMaker.InventoryTest.testRemoveIngredients(InventoryTest.java:26) <1 internal lines

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

```
public synchronized boolean useIngredients(Recipe r) {
    if (enoughIngredients(r)) {
        Inventory.coffee += r.getAmtCoffee();
        Inventory.milk -= r.getAmtMilk();
        Inventory.sugar -= r.getAmtSugar();
        Inventory.chocolate -= r.getAmtChocolate();
        return true;
    } else {
        return false;
    }
}
```

After:

The screenshot shows the 'After' state of the InventoryTest.java file. The test method 'testRemoveIngredients()' is shown, which creates a Recipe object, sets its ingredients, and asserts that the inventory has the correct amounts of ingredients. The test passes because the inventory has the correct amounts of ingredients.

```
public class InventoryTest {
    @Test
    void testRemoveIngredients() {
        Recipe r = new Recipe();
        Inventory inventory = new Inventory();
        try {
            r.setAmtChocolate("5");
            r.setAmtCoffee("5");
            r.setAmtMilk("5");
            r.setAmtSugar("5");
        } catch (Exception e) {
            assert(false);
        }
        assert(inventory.useIngredients(r));
        assert(inventory.getChocolate() == 10);
        assert(inventory.getSugar() == 10);
        assert(inventory.getCoffee() == 10);
        assert(inventory.getMilk() == 10);
    }
}
```

Tests passed: 1 of 1 test – 27 ms

Process finished with exit code 0

```

public synchronized boolean useIngredients(Recipe r) {
    if (enoughIngredients(r)) {
        Inventory.coffee -= r.getAmtCoffee();
        Inventory.milk -= r.getAmtMilk();
        Inventory.sugar -= r.getAmtSugar();
        Inventory.chocolate -= r.getAmtChocolate();
        return true;
    } else {
        return false;
    }
}

```

Defect 3: When trying to add to the inventory, the command line interface tells us “Inventory was not added” no matter what integers we provide which is incorrect because checking the inventory afterwards shows that the amount of coffee and milk I’ve added was added correctly. It’s just Sugar and Chocolate that is not being successfully added. This defect will focus on fixing the problem with **adding chocolate** to the inventory.

Class Defect Is In: Inventory

Function/Method Defect is in: setPrice(String price)

Defect 4: When utilizing the addSugar function, the function will always throw an InventoryException because the incorrect inequality is used.

Class Defect Is In: Inventory

Function/Method Defect is in: addSugar(String sugar)

Before:

```
171  * to the current amount of sugar units.
172  * @param sugar
173  * @throws InventoryException
174  */
175  public synchronized void addSugar(String sugar) throws InventoryException {
176      int amtSugar = 0;
177      try {
178          amtSugar = Integer.parseInt(sugar);
179      } catch (NumberFormatException e) {
180          throw new InventoryException("Units of sugar must be a positive integer");
181      }
182      if (amtSugar <= 0) {
183          throw new InventoryException("Units of sugar must be a positive integer");
184      } else {
185          Inventory.sugar += amtSugar;
186      }
187  }
188  /**
189  */
190  }
191  }
192  }
193  }
194  }
195  }
196  }
197  }
198  }
199  }
200  }
201  }
202  }
203  }
204  }
205  }
206  }
207  }
208  }
209  }
210  }
211  }
212  }
213  }
214  }
215  }
216  }
217  }
218  }
219  }
220  }
221  }
222  }
223  }
224  }
225  }
226  }
227  }
228  }
229  }
230  }
231  }
232  }
233  }
234  }
235  }
236  }
237  }
238  }
239  }
240  }
241  }
242  }
243  }
244  }
245  }
246  }
247  }
248  }
249  }
250  }
251  }
252  }
253  }
254  }
255  }
256  }
257  }
258  }
259  }
260  }
261  }
262  }
263  }
264  }
265  }
266  }
267  }
268  }
269  }
270  }
271  }
272  }
273  }
274  }
275  }
276  }
277  }
278  }
279  }
280  }
281  }
282  }
283  }
284  }
285  }
286  }
287  }
288  }
289  }
290  }
291  }
292  }
293  }
294  }
295  }
296  }
297  }
298  }
299  }
300  }
301  }
302  }
303  }
304  }
305  }
306  }
307  }
308  }
309  }
310  }
311  }
312  }
313  }
314  }
315  }
316  }
317  }
318  }
319  }
320  }
321  }
322  }
323  }
324  }
325  }
326  }
327  }
328  }
329  }
330  }
331  }
332  }
333  }
334  }
335  }
336  }
337  }
338  }
339  }
340  }
341  }
342  }
343  }
344  }
345  }
346  }
347  }
348  }
349  }
350  }
351  }
352  }
353  }
354  }
355  }
356  }
357  }
358  }
359  }
360  }
361  }
362  }
363  }
364  }
365  }
366  }
367  }
368  }
369  }
370  }
371  }
372  }
373  }
374  }
375  }
376  }
377  }
378  }
379  }
380  }
381  }
382  }
383  }
384  }
385  }
386  }
387  }
388  }
389  }
390  }
391  }
392  }
393  }
394  }
395  }
396  }
397  }
398  }
399  }
400  }
401  }
402  }
403  }
404  }
405  }
406  }
407  }
408  }
409  }
410  }
411  }
412  }
413  }
414  }
415  }
416  }
417  }
418  }
419  }
420  }
421  }
422  }
423  }
424  }
425  }
426  }
427  }
428  }
429  }
430  }
431  }
432  }
433  }
434  }
435  }
436  }
437  }
438  }
439  }
440  }
441  }
442  }
443  }
444  }
445  }
446  }
447  }
448  }
449  }
450  }
451  }
452  }
453  }
454  }
455  }
456  }
457  }
458  }
459  }
460  }
461  }
462  }
463  }
464  }
465  }
466  }
467  }
468  }
469  }
470  }
471  }
472  }
473  }
474  }
475  }
476  }
477  }
478  }
479  }
480  }
481  }
482  }
483  }
484  }
485  }
486  }
487  }
488  }
489  }
490  }
491  }
492  }
493  }
494  }
495  }
496  }
497  }
498  }
499  }
500  }
501  }
502  }
503  }
504  }
505  }
506  }
507  }
508  }
509  }
510  }
511  }
512  }
513  }
514  }
515  }
516  }
517  }
518  }
519  }
520  }
521  }
522  }
523  }
524  }
525  }
526  }
527  }
528  }
529  }
530  }
531  }
532  }
533  }
534  }
535  }
536  }
537  }
538  }
539  }
540  }
541  }
542  }
543  }
544  }
545  }
546  }
547  }
548  }
549  }
550  }
551  }
552  }
553  }
554  }
555  }
556  }
557  }
558  }
559  }
560  }
561  }
562  }
563  }
564  }
565  }
566  }
567  }
568  }
569  }
570  }
571  }
572  }
573  }
574  }
575  }
576  }
577  }
578  }
579  }
580  }
581  }
582  }
583  }
584  }
585  }
586  }
587  }
588  }
589  }
590  }
591  }
592  }
593  }
594  }
595  }
596  }
597  }
598  }
599  }
600  }
601  }
602  }
603  }
604  }
605  }
606  }
607  }
608  }
609  }
610  }
611  }
612  }
613  }
614  }
615  }
616  }
617  }
618  }
619  }
620  }
621  }
622  }
623  }
624  }
625  }
626  }
627  }
628  }
629  }
630  }
631  }
632  }
633  }
634  }
635  }
636  }
637  }
638  }
639  }
640  }
641  }
642  }
643  }
644  }
645  }
646  }
647  }
648  }
649  }
650  }
651  }
652  }
653  }
654  }
655  }
656  }
657  }
658  }
659  }
660  }
661  }
662  }
663  }
664  }
665  }
666  }
667  }
668  }
669  }
670  }
671  }
672  }
673  }
674  }
675  }
676  }
677  }
678  }
679  }
680  }
681  }
682  }
683  }
684  }
685  }
686  }
687  }
688  }
689  }
690  }
691  }
692  }
693  }
694  }
695  }
696  }
697  }
698  }
699  }
700  }
701  }
702  }
703  }
704  }
705  }
706  }
707  }
708  }
709  }
710  }
711  }
712  }
713  }
714  }
715  }
716  }
717  }
718  }
719  }
720  }
721  }
722  }
723  }
724  }
725  }
726  }
727  }
728  }
729  }
730  }
731  }
732  }
733  }
734  }
735  }
736  }
737  }
738  }
739  }
740  }
741  }
742  }
743  }
744  }
745  }
746  }
747  }
748  }
749  }
750  }
751  }
752  }
753  }
754  }
755  }
756  }
757  }
758  }
759  }
760  }
761  }
762  }
763  }
764  }
765  }
766  }
767  }
768  }
769  }
770  }
771  }
772  }
773  }
774  }
775  }
776  }
777  }
778  }
779  }
780  }
781  }
782  }
783  }
784  }
785  }
786  }
787  }
788  }
789  }
790  }
791  }
792  }
793  }
794  }
795  }
796  }
797  }
798  }
799  }
800  }
801  }
802  }
803  }
804  }
805  }
806  }
807  }
808  }
809  }
810  }
811  }
812  }
813  }
814  }
815  }
816  }
817  }
818  }
819  }
820  }
821  }
822  }
823  }
824  }
825  }
826  }
827  }
828  }
829  }
830  }
831  }
832  }
833  }
834  }
835  }
836  }
837  }
838  }
839  }
840  }
841  }
842  }
843  }
844  }
845  }
846  }
847  }
848  }
849  }
850  }
851  }
852  }
853  }
854  }
855  }
856  }
857  }
858  }
859  }
860  }
861  }
862  }
863  }
864  }
865  }
866  }
867  }
868  }
869  }
870  }
871  }
872  }
873  }
874  }
875  }
876  }
877  }
878  }
879  }
880  }
881  }
882  }
883  }
884  }
885  }
886  }
887  }
888  }
889  }
890  }
891  }
892  }
893  }
894  }
895  }
896  }
897  }
898  }
899  }
900  }
901  }
902  }
903  }
904  }
905  }
906  }
907  }
908  }
909  }
910  }
911  }
912  }
913  }
914  }
915  }
916  }
917  }
918  }
919  }
920  }
921  }
922  }
923  }
924  }
925  }
926  }
927  }
928  }
929  }
930  }
931  }
932  }
933  }
934  }
935  }
936  }
937  }
938  }
939  }
940  }
941  }
942  }
943  }
944  }
945  }
946  }
947  }
948  }
949  }
950  }
951  }
952  }
953  }
954  }
955  }
956  }
957  }
958  }
959  }
960  }
961  }
962  }
963  }
964  }
965  }
966  }
967  }
968  }
969  }
970  }
971  }
972  }
973  }
974  }
975  }
976  }
977  }
978  }
979  }
980  }
981  }
982  }
983  }
984  }
985  }
986  }
987  }
988  }
989  }
990  }
991  }
992  }
993  }
994  }
995  }
996  }
997  }
998  }
999  }
1000  }
```

Tests failed: 1, passed: 14 of 15 tests - 95 ms

java.lang.RuntimeException: coffeemaker.exceptions.InventoryException: Units of sugar must be a positive integer

at coffeemaker.InventoryTest.testAddSugar(InventoryTest.java:99) <1 internal line>

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Caused by: coffeemaker.exceptions.InventoryException: Units of sugar must be a positive integer

at coffeemaker.Inventory.addSugar(Inventory.java:185)

at coffeemaker.InventoryTest.testAddSugar(InventoryTest.java:96)

After:

The screenshot shows an IDE with two tabs: `Inventory.java` and `InventoryTest.java`. The `Inventory.java` tab is active, showing a method `addSugar` that takes a `String sugar` parameter and throws an `InventoryException`. The method increments the `sugar` amount by 15 units. The `InventoryTest.java` tab shows three test methods: `testGetSugar`, `testSetSugar`, and `testAddSugar`. The `testAddSugar` method is highlighted, showing it calls `inventory.addSugar("15")` and asserts that the `sugar` value is 30. Below the code, the 'Cover' tab is open, displaying a test coverage report. The report shows that 15 of 15 tests passed, with a total execution time of 69 ms. The tests listed include `testGetSugar`, `testAddMilk`, `testSetChocolate`, `testAddSugar`, `testGetMilk`, `testAddCoffee`, `testSetCoffee`, `testGetCoffee`, `testAddChocolate`, and `testSetMilk`.

```
171 * to the current amount of sugar units.
172 * @param sugar
173 * @throws InventoryException
174 */
175 2 usages
176 public synchronized void addSugar(String sugar) throws InventoryException {
177     int amtSugar = 0;
178     try {
179         amtSugar = Integer.parseInt(sugar);
180     } catch (NumberFormatException e) {
181         throw new InventoryException("Units of sugar must be a number");
182     }
183     if (amtSugar >= 0) {
184         Inventory.sugar += amtSugar;
185     } else {
186         throw new InventoryException("Units of sugar must be non-negative");
187     }
188 }
189 /**
```

```
80 }
81
82 @Test
83 void testGetSugar() { assertEquals( expected: 15, inventory.getSugar()); }
84
85 @Test
86 void testSetSugar() {
87     inventory.setSugar(0);
88     assertEquals( expected: 0, inventory.getSugar());
89 }
90
91 @Test
92 void testAddSugar() {
93     try {
94         inventory.addSugar("15");
95         assertEquals( expected: 30, inventory.getSugar());
96     } catch (InventoryException e) {
97         throw new RuntimeException(e);
98     }
99 }
100
101 }
102 }
```

Cover Main x All Tests x

Tests passed: 15 of 15 tests - 69 ms

Process finished with exit code 0

Test Method	Time
<default package>	69 ms
InventoryTest	62 ms
testGetSugar()	49 ms
testAddMilk()	1 ms
testSetChocolate()	1 ms
testAddSugar()	1 ms
testGetMilk()	1 ms
testAddCoffee()	1 ms
testSetCoffee()	1 ms
testGetCoffee()	1 ms
testAddChocolate()	3 ms
testSetMilk()	1 ms

For Gabe for deliverable 4: satisfy code coverage for recipe

Coverage for Inventory

IDE screenshot showing the CoffeeMaker project. The main editor displays the `InventoryTest.java` file, which contains a `testUseIngredientsMilk()` method. The `Inventory.java` file is also visible, showing the `useIngredients()` method. The `Recipe.java` and `RecipeBook.java` files are also open.

The `testUseIngredientsMilk()` method in `InventoryTest.java` is as follows:

```
void testUseIngredientsMilk() throws RecipeException {
    Recipe recipe = new Recipe();
    recipe.setAmMilk("9");
    Inventory inventory = new Inventory();
    inventory.useIngredients(recipe);
    int expected = 6;
    int actual = inventory.getMilk();
    assertEquals(expected, actual);
}
```

The `useIngredients()` method in `Inventory.java` is as follows:

```
void useIngredients(Recipe recipe) {
    amtChocolate = Integer.parseInt(recipe.getAmChocolate());
    catch (NumberFormatException e) {
        throw new RecipeException("Units of chocolate are not a number");
    }
    if (amtChocolate >= 0) {
        this.amtChocolate = amtChocolate;
    } else {
        throw new RecipeException("Units of chocolate are not a number");
    }
}
```

The `Recipe.java` file shows the `getAmChocolate()` method:

```
String getAmChocolate() {
    return amChocolate;
}
```

The `RecipeBook.java` file shows the `addRecipe()` method:

```
void addRecipe(Recipe recipe) {
    recipes.put(recipe.getName(), recipe);
}
```

The `Coverage` tab shows the following coverage data:

Element	Class...	Meth...	Line...
coffeemaker	42% (1... 45% (1... 33% (1...		
exceptions	50% (1... 50% (1... 50% (1...		
CoffeeMaker	0% (0/1) 0% (0/8) 0% (0/...		
Inventory	100% (1... 100% (1... 100% (1...		
Main	0% (0/1) 0% (0/...) 0% (0/...		
Recipe	100% (1... 56% (1... 42% (1...		
RecipeBook	0% (0/1) 0% (0/5) 0% (0/...		

The `Tests` tab shows the following test results:

Test	Time
testEnoughIngredientsMilk	48 ms
testAddCoffeeNegativeNum	3 ms
testAddSugarNumberFormat	1 ms
testEnoughIngredientsMilkTrue	1 ms
testAddMilkNumberFormatEx	1 ms
testGetSugar()	1 ms
testAddMilk()	1 ms
testUseIngredientsCoffee()	1 ms
testUseIngredientsChocolate()	1 ms
testToString()	1 ms
testSetChocolate()	1 ms
testAddMilkNegativeNumber()	1 ms
testAddSugar()	1 ms
testEnoughIngredientsChocolate()	1 ms
testGetMilk()	1 ms
testAddChocolateNegativeN	2 ms
testAddChocolateNumberFor	1 ms
testUseIngredientsSugar()	1 ms
testAddCoffee()	1 ms
testAddSugarNegativeNumb	1 ms
testEnoughIngredientsSugar()	1 ms
testSetCoffee()	1 ms
testUseIngredientsMilk()	1 ms
testEnoughIngredientsCoffee()	1 ms
testAddCoffeeNumberForma	1 ms
testAddChocolate()	1 ms
testSetMilk()	1 ms
testGetChocolate()	1 ms
testSetSugar()	1 ms

The `Process finished with exit code 0`.

IDE screenshot showing the CoffeeMaker project. The main editor displays the `InventoryTest.java` file, which contains a `testUseIngredientsMilk()` method. The `Inventory.java` file is also visible, showing the `useIngredients()` method. The `Recipe.java` and `RecipeBook.java` files are also open.

The `testUseIngredientsMilk()` method in `InventoryTest.java` is as follows:

```
void testUseIngredientsMilk() throws RecipeException {
    Recipe recipe = new Recipe();
    recipe.setAmMilk("9");
    Inventory inventory = new Inventory();
    inventory.useIngredients(recipe);
    int expected = 6;
    int actual = inventory.getMilk();
    assertEquals(expected, actual);
}
```

The `useIngredients()` method in `Inventory.java` is as follows:

```
void useIngredients(Recipe recipe) {
    amtChocolate = Integer.parseInt(recipe.getAmChocolate());
    catch (NumberFormatException e) {
        throw new RecipeException("Units of chocolate are not a number");
    }
    if (amtChocolate >= 0) {
        this.amtChocolate = amtChocolate;
    } else {
        throw new RecipeException("Units of chocolate are not a number");
    }
}
```

The `Recipe.java` file shows the `getAmChocolate()` method:

```
String getAmChocolate() {
    return amChocolate;
}
```

The `RecipeBook.java` file shows the `addRecipe()` method:

```
void addRecipe(Recipe recipe) {
    recipes.put(recipe.getName(), recipe);
}
```

The `Coverage` tab shows the following coverage data:

Element	Class...	Meth...	Line...
coffeemaker	42% (1... 45% (1... 33% (1...		
exceptions	50% (1... 50% (1... 50% (1...		
CoffeeMaker	0% (0/1) 0% (0/8) 0% (0/...		
Inventory	100% (1... 100% (1... 100% (1...		
Main	0% (0/1) 0% (0/...) 0% (0/...		
Recipe	100% (1... 56% (1... 42% (1...		
RecipeBook	0% (0/1) 0% (0/5) 0% (0/...		

The `Tests` tab shows the following test results:

Test	Time
testUseIngredientsCoffee()	1 ms
testUseIngredientsChocolate()	1 ms
testToString()	1 ms
testSetChocolate()	1 ms
testAddMilkNegativeNumber()	1 ms
testAddSugar()	1 ms
testEnoughIngredientsChocolate()	1 ms
testGetMilk()	1 ms
testAddChocolateNegativeN	2 ms
testAddChocolateNumberFor	1 ms
testUseIngredientsSugar()	1 ms
testAddCoffee()	1 ms
testAddSugarNegativeNumb	1 ms
testEnoughIngredientsSugar()	1 ms
testSetCoffee()	1 ms
testUseIngredientsMilk()	1 ms
testEnoughIngredientsCoffee()	1 ms
testAddCoffeeNumberForma	1 ms
testAddChocolate()	1 ms
testSetMilk()	1 ms
testGetChocolate()	1 ms
testSetSugar()	1 ms

The `Process finished with exit code 0`.

Project view on the left shows the directory structure of the `CoffeeMaker` project, including `src`, `main`, and `java` subdirectories. The `src` directory contains `coffeemaker` (28 classes, 13% lines covered) and `exceptions` (0% classes, 0% lines covered).

The central editor displays the `RecipeBookTest.java` file, showing a test method `editRecipeOutOfBounds()` at line 137. The code includes assertions for `Recipe[]` arrays and a `RecipeBook` instance.

The right sidebar shows the **Coverage** view for `RecipeBookTest`. It displays a table of coverage data for various classes and methods.

Element	Class, %	Method, %	Line, %
coffeemaker	28% (2/7)	15% (9/57)	13% (45/33...)
exceptions	0% (0/2)	0% (0/2)	0% (0/2)
CoffeeMaker	0% (0/1)	0% (0/8)	0% (0/21)
Inventory	0% (0/1)	0% (0/16)	0% (0/80)
Main	0% (0/1)	0% (0/10)	0% (0/133)
Recipe	100% (1/1)	25% (4/16)	26% (19/73)
RecipeBook	100% (1/1)	100% (5/5)	100% (26/2...)

The bottom panel shows the **Cover** view for `RecipeBookTest`. It displays a list of test methods and their execution times, along with the command used to run the tests.

Tests passed: 12 of 12 tests - 41 ms

Process finished with exit code 0

Command: `"C:\Program Files\Java\jdk-17.0.4\bin\java.exe" -ea -javaagent:C:\Users\jense\AppData\Local\JetBrains\IntelliJ\Idea2023.3\testAgent\intelli...`