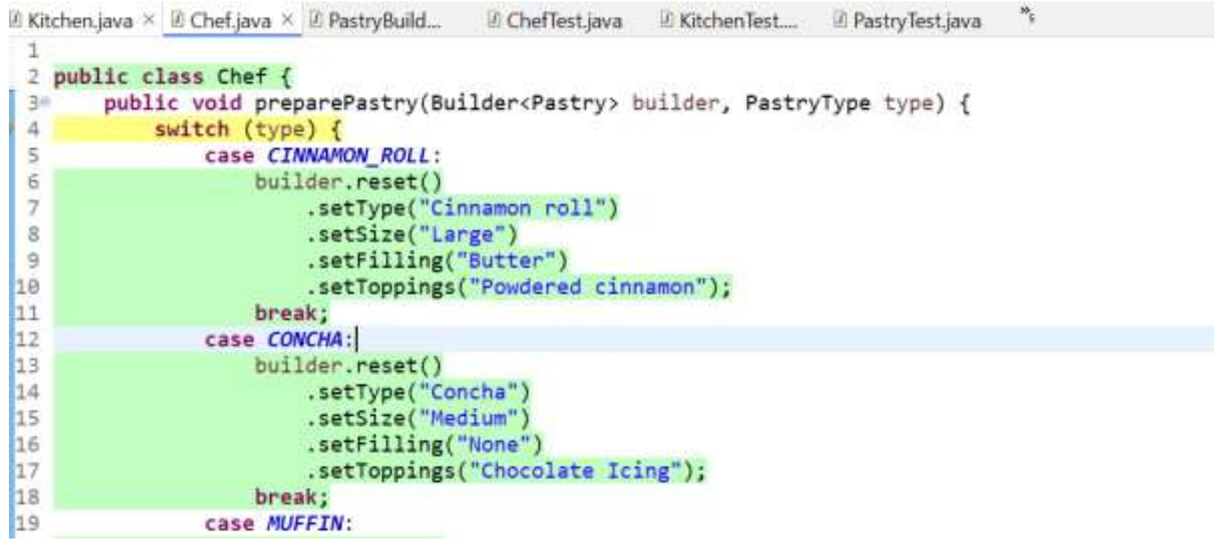
BUILDER

Ever Hinojosa Aguirre

THE PASTRY SHOP

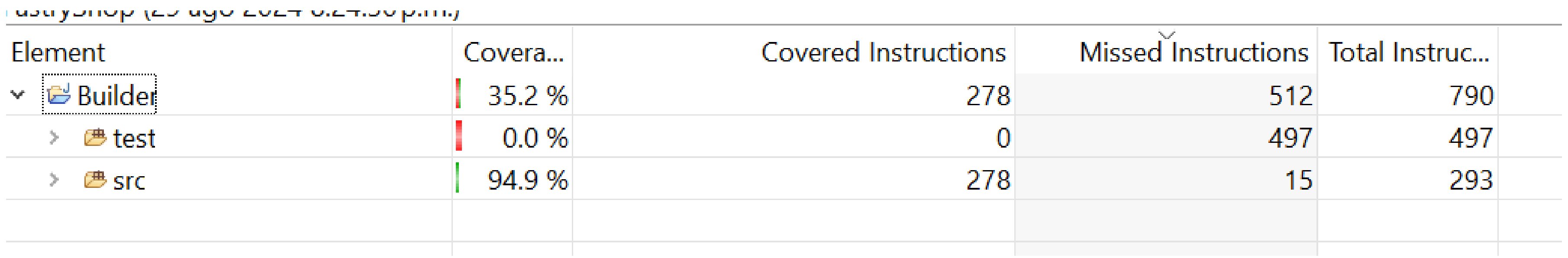
From the construction nature of the pattern, i gravitated to the construction of food, seeing as pastries can be produced kinda from the same mold and change according to the specifications of the client, It sounds like a great way to apply the builder pattern, changing the director for a chef class that "constructs" the pastries.

More pastries can be added in the enum and then give the build to the chef.



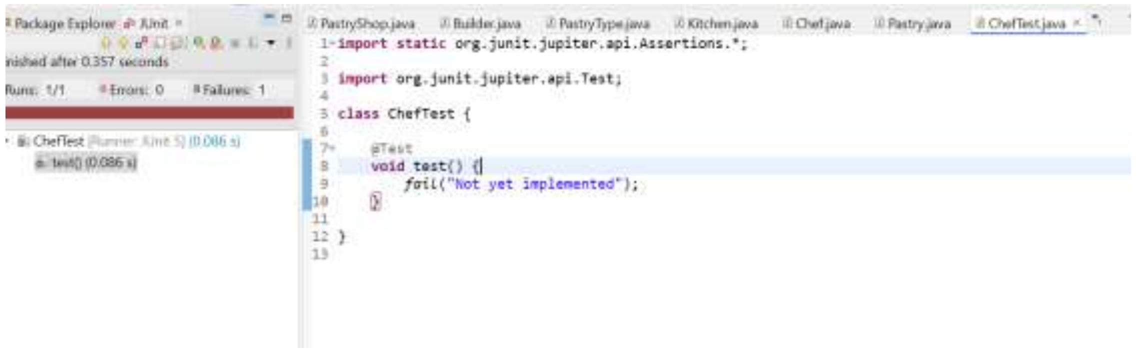
```

public class Chef {
    public void preparePastry(Builder<Pastry> builder, PastryType type) {
        switch (type) {
            case CINNAMON_ROLL:
                builder.reset()
                    .setType("Cinnamon roll")
                    .setSize("Large")
                    .setFilling("Butter")
                    .setToppings("Powdered cinnamon");
                break;
            case CONCHA:
                builder.reset()
                    .setType("Concha")
                    .setSize("Medium")
                    .setFilling("None")
                    .setToppings("Chocolate Icing");
                break;
            case MUFFIN:
```

JUNIT



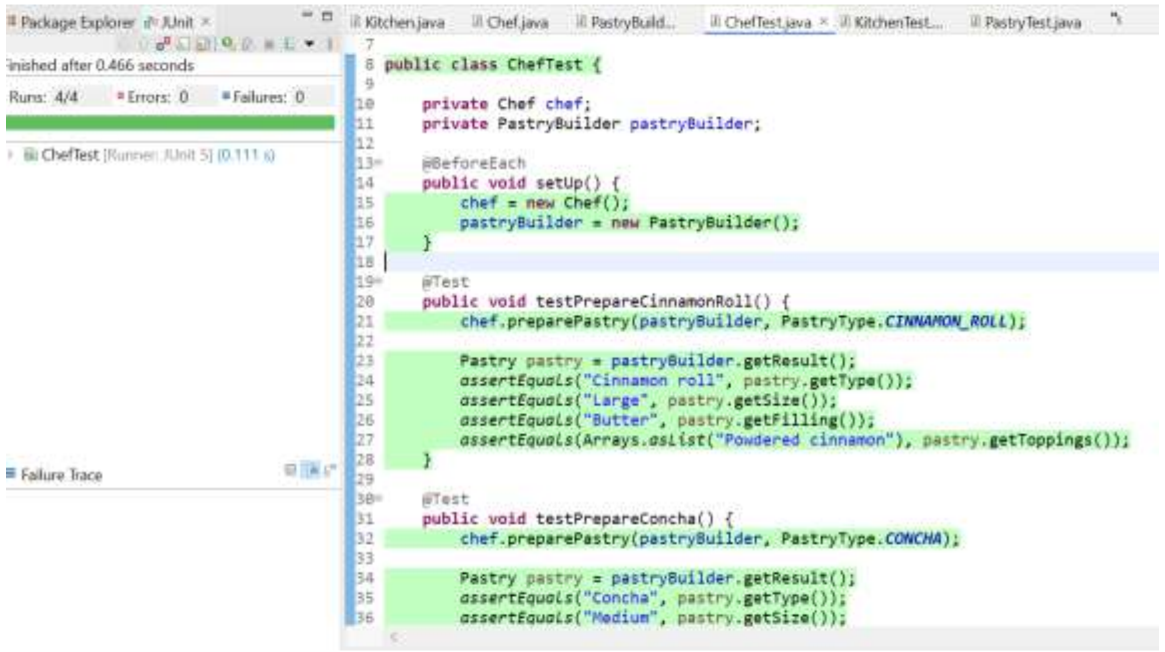
Element	Covera...	Covered Instructions	Missed Instructions	Total Instruc...
Builder	35.2 %	278	512	790
test	0.0 %	0	497	497
src	94.9 %	278	15	293

Created test for almost all of the code, The coverage of my original source is 94.9%, unless im wrong i dont take in account the 0.0% of my test directory



```
inished after 0.357 seconds
Runs: 1/1    Errors: 0    Failures: 1
ChefTest [Runner: JUnit 5] (0.086 s)
    test() (0.086 s)
```

```java
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class ChefTest {

    @Test
    void test() {
        fail("Not yet implemented");
    }

}
```



```
inished after 0.466 seconds
Runs: 4/4    Errors: 0    Failures: 0
ChefTest [Runner: JUnit 5] (0.111 s)
```

```java

public class ChefTest {

    private Chef chef;
    private PastryBuilder pastryBuilder;

    @BeforeEach
    public void setUp() {
        chef = new Chef();
        pastryBuilder = new PastryBuilder();
    }

    @Test
    public void testPrepareCinnamonRoll() {
        chef.preparePastry(pastryBuilder, PastryType.CINNAMON_ROLL);

        Pastry pastry = pastryBuilder.getResult();
        assertEquals("Cinnamon roll", pastry.getType());
        assertEquals("Large", pastry.getSize());
        assertEquals("Butter", pastry.getFilling());
        assertEquals(Arrays.asList("Powdered cinnamon"), pastry.getToppings());
    }

    @Test
    public void testPrepareConcha() {
        chef.preparePastry(pastryBuilder, PastryType.CONCHA);

        Pastry pastry = pastryBuilder.getResult();
        assertEquals("Concha", pastry.getType());
        assertEquals("Medium", pastry.getSize());
```