2025

# Práctica 3 -CoffeMaker

USING JUNIT & MOCKITO FOR TESTING

BRIAN VALIENTE RÓDENAS

Brian Valiente Rodenas

# Contenido

Brian Valiente Rodenas

# 🚀 INFORME: Errores encontrados y correcciones

## 1. Error en el método deleteRecipe de la clase CoffeeMaker

**Error detectado:**

En el test se esperaba que, tras eliminar una receta, ésta tuviera el nombre "No data". El código original tenía:

```
public boolean deleteRecipe(Recipe r) {
    boolean canDeleteRecipe = false;
    if(r != null) {
        for(int i = 0; i < NUM_RECIPES; i++) {
            if(r.equals(recipeArray[i])) {
                recipeArray[i] = recipeArray[i]; // <<<<< ERROR
                canDeleteRecipe = true;
            }
        }
    }
    return canDeleteRecipe;
}
```

**Corrección realizada:**

Se cambió el método para que, al encontrar la receta, se reemplace por una nueva instancia de Recipe (que por defecto tendrá el nombre "No data") y se actualice el indicador del espacio ocupado (recipeFull):

```
public boolean deleteRecipe(Recipe r) {
    boolean canDeleteRecipe = false;
    if(!"No data".equals(r.getName())) {
        for(int i = 0; i < NUM_RECIPES; i++) {
            if(r.equals(recipeArray[i])) {
                recipeArray[i] = new Recipe();
                recipeFull[i] = false;
                canDeleteRecipe = true;
            }
        }
    }
    return canDeleteRecipe;
}
```

Brian Valiente Rodenas

## 2. Simplificación del método equals en la clase Recipe

**Código original (con redundancia):**

```
public boolean equals(Recipe r) {
    if(r.getName() == null) {
        return false;
    }
    if(this.name == null) {
        return false;
    }
    if((this.name).equals(r.getName())) {
        return true;
    }
    return false;
}
```

**Corrección realizada:**

Se simplifica directamente a esto ya que ahora las recetas vacías son “No data”:

```
public boolean equals(Recipe r) {
    return (this.name).equals(r.getName());
}
```

## 3. Error en el método setAmtSugar de la clase Recipe

**Error detectado:**

En el test se esperaba que, tras comprar café, se actualizara correctamente el inventario, pero no fue así. El método actual asigna el valor de azúcar a amtMilk en lugar de a amtSugar:

```
public void setAmtSugar(int amtSugar) {
    if(amtSugar >= 0) {
        this.amtMilk = amtSugar;  // >>>>> ERROR: actualizando amtMilk en Sugar
    }
    else {
        this.amtSugar = 0;
    }
}
```

**Corrección realizada:**

Se debe asignar correctamente a amtSugar:

```java
public void setAmtSugar(int amtSugar) {
    if(amtSugar >= 0) {
        this.amtSugar = amtSugar;
    }
    else {
        this.amtSugar = 0;
    }
}
```

## 4. Constructores faltantes en la clase Recipe

**Corrección realizada:**

Se añadieron los constructores siguientes:

```java
public Recipe() {
    this.name = "No data";
    this.price = 0;
    this.amtCoffee = 0;
    this.amtMilk = 0;
    this.amtSugar = 0;
    this.amtChocolate = 0;
}

public Recipe(String name, int price, int amtCoffee, int amtMilk, int amtSugar, int amtChoco
    this.name = name;
    this.price = price;
    this.amtCoffee = amtCoffee;
    this.amtMilk = amtMilk;
    this.amtSugar = amtSugar;
    this.amtChocolate = amtChocolate;
}
```

## 5. Error en el método editRecipe de la clase CoffeeMaker

**Error detectado:**

El método original no editaba correctamente la receta ya que usaba newRecipe.equals(...) en lugar de oldRecipe.equals(...) y contenía redundancia al llamar a addRecipe(newRecipe).

**Código original:**

```
public boolean editRecipe(Recipe oldRecipe, Recipe newRecipe) {
    boolean canEditRecipe = false;
    for(int i = 0; i < NUM_RECIPES; i++) {
        if(recipeArray[i].getName() != null) {
            if(newRecipe.equals(recipeArray[i])) {  // <<<<< ERROR: debería ser oldRecipe
                recipeArray[i] = new Recipe();
                if(addRecipe(newRecipe)) {  // <<<<< REDUNDANCIA
                    canEditRecipe = true;
                } else {
                    canEditRecipe = false;
                }
            }
        }
    }
    return canEditRecipe;
}
```

**Corrección realizada:**

Se cambió a (comprobando oldRecipe y que el nombre de la antigua coincida con la nueva como restricción del sistema al editar recetas, especificado en los casos de uso):

```
public boolean editRecipe(Recipe oldRecipe, Recipe newRecipe) {
    boolean canEditRecipe = false;
    if(!"No data".equals(oldRecipe.getName())) {
        for(int i = 0; i < NUM_RECIPES; i++) {
            if(!"No data".equals(recipeArray[i].getName())) {
                if(oldRecipe.equals(recipeArray[i]) && oldRecipe.getName().equals(newRecipe.ge
                    canEditRecipe = true;
                    if(canEditRecipe) {
                        recipeArray[i] = newRecipe;
                        return canEditRecipe;
                    }
                }
            }
        }
    }
    return canEditRecipe;
}
```

# 6. Error en el método addInventory de la clase CoffeeMaker

**Error detectado:**

El test comprueba que cantidades positivas se acepten. En el código original se cometió un error en la condición para amtSugar:

```
if(amtCoffee < 0 || amtMilk < 0 || amtSugar > 0 || amtChocolate < 0) {
    canAddInventory = false;
}
```

**Corrección realizada:**

La condición debe ser que alguna cantidad sea negativa:

```
if(amtCoffee < 0 || amtMilk < 0 || amtSugar < 0 || amtChocolate < 0) {
    canAddInventory = false;
}
```

## 7. Error en el método makeCoffee de la clase CoffeeMaker

**Error detectado:**

El método actual actualizaba incorrectamente el inventario, sumando en lugar de restar para el café, y no validaba si la receta tenía el nombre "No data". Código original con error en la actualización:

```
public int makeCoffee(Recipe r, int amtPaid) {
    boolean canMakeCoffee = true;
    if(amtPaid < r.getPrice()) {
        canMakeCoffee = false;
    }
    if(!inventory.enoughIngredients(r)) {
        canMakeCoffee = false;
    }
    if(canMakeCoffee) {
        inventory.setCoffee(inventory.getCoffee() + r.getAmtCoffee()); // <<<<< ERROR: se suma
        inventory.setMilk(inventory.getMilk() - r.getAmtMilk());
        inventory.setSugar(inventory.getSugar() - r.getAmtSugar());
        inventory.setChocolate(inventory.getChocolate() - r.getAmtChocolate());
        return amtPaid - r.getPrice();
    }
    else {
        return amtPaid;
    }
}
```

**Corrección realizada:**

Se modificó el método para que valide el nombre de la receta, imprima mensajes informativos y realice la actualización correcta (resta)

```
inventory.setCoffee(inventory.getCoffee() - r.getAmtCoffee());
```

Brian Valiente Rodenas

# 8. Error en el método setChocolate de la clase Inventory

**Error detectado:**

El método usaba el operador += en lugar de una asignación directa:

```
public void setChocolate(int chocolate) {
    if(chocolate >= 0) {
        Inventory.chocolate += chocolate; // >>>>> ERROR: no es acumulativo
    }
    else {
        Inventory.chocolate = 0;
    }
}
```

**Corrección realizada:**

Se cambia a:

```
        Inventory.chocolate = chocolate;
```

# 9. Error en el método deleteRecipe de la clase Main

**Error detectado:**

Al eliminar una receta, se utiliza el método getName() sobre la receta ya borrada, por lo que no se muestra el nombre correcto.

**Código original:**

```
public static void deleteRecipe() {
    Recipe [] recipes = coffeeMaker.getRecipes();
    for(int i = 0; i < recipes.length; i++) {
        System.out.println((i+1) + ". " + recipes[i].getName());
    }
    String recipeToDeleteString = inputOutput("Please select the number of the recipe t
    int recipeToDelete = stringToInt(recipeToDeleteString) - 1;
    if(recipeToDelete < 0) {
        mainMenu();
    }

    boolean recipeDeleted = coffeeMaker.deleteRecipe(recipes[recipeToDelete]);

    if(recipeDeleted) System.out.println(recipes[recipeToDelete].getName() + " successf
    else System.out.println(recipes[recipeToDelete].getName() + " could not be deleted.

    mainMenu();
}
```

**Corrección realizada:**

Se guarda el nombre de la receta antes de eliminarla:

```
String name = recipes[recipeToDelete].getName();
boolean recipeDeleted = coffeeMaker.deleteRecipe(recipes[recipeToDelete]);

if(recipeDeleted) System.out.println(name + " successfully deleted.");
else System.out.println(name + " could not be deleted.");
```

# 10. Otros cambios menores

- **Uso de la anotación @Override**: Se añadió @Override en los métodos toString() tanto en la clase **Inventory** como en la clase **Recipe**.
- **Eliminación de código no utilizado**: Se eliminó la variable boolean recipeAdded = false; en la clase **Main** y se declaró correctamente en la línea donde se asigna el resultado de coffeeMaker.addRecipe(r).
- **Optimización en el método addRecipe de la clase CoffeeMaker**: Se añadió un break; en el bucle que busca el primer hueco libre para almacenar la receta, para evitar iteraciones innecesarias.

- **Restricción en el número de recetas**: Según el enunciado se indica que solo se pueden añadir 3 recetas, por lo que se utiliza la constante private final int NUM_RECIPES = 3; en la clase **CoffeeMaker**.
- **En la clase CoffeMaker**: El método getRecipeForName() lo he editado para que sea con "No data" consecuente con los cambios realizados:

```
public Recipe getRecipeForName(String name) {
    Recipe r = new Recipe();
    for(int i = 0; i < NUM_RECIPES; i++) {
        if(!"No data".equals(recipeArray[i].getName())) {
            if((recipeArray[i].getName()).equals(name)) {
                r = recipeArray[i];
            }
        }
    }
    return r;
}
```

# 11. Adaptación de la clase Inventory para leer datos desde un fichero JSON

Se modificó la clase **Inventory** para que los datos iniciales de los 4 ingredientes se obtengan desde un fichero JSON. Durante el proceso de pruebas, surgieron varios problemas al intentar utilizar *Mockito* para crear *mocks* o *spies* de la clase *Inventory*. Inicialmente, se intentó interceptar y modificar la clase mediante *mocking* o *spying*, pero esto generó errores.

El problema radica en que *Mockito*, al usar el *inline mock maker* (añadido previamente en las dependencias del proyecto), intenta modificar la clase *Inventory* (y en algunos casos, también *java.lang.Object*) para poder interceptar y "stubear" llamadas a sus métodos. Sin embargo, esta técnica requiere la instrumentación del bytecode, y ahí es donde se presentó una limitación.

El error específico reportado fue:

"Mockito cannot mock this class: class com.coffeemaker.Inventory"

Analizando la traza de error, se encontró que el problema estaba relacionado con *Byte Buddy*, la biblioteca que *Mockito* usa internamente para instrumentar clases en *mocks* en línea (*inline mocks*). La versión actual de *Byte Buddy* no soporta oficialmente *Java 23*, ya que su compatibilidad se extiende solo hasta *Java 22*. Esto impide la instrumentación de clases en la JVM más reciente y provoca que al intentar crear un *spy* o *mock* de *Inventory*, se lance una excepción.

Se exploraron varias soluciones, entre ellas:

- **Actualizar Byte Buddy** a una versión más reciente que pudiera soportar *Java 23*.

- **Revertir a una versión anterior de Java (Java 21)** para evitar la incompatibilidad. Sin embargo, esto no resolvió el problema.
- **Configurar la JVM con la propiedad net.bytebuddy.experimental** para forzar la instrumentación en versiones no soportadas.

Ninguna de estas soluciones resultó efectiva en nuestro caso.

Finalmente, la estrategia adoptada fue **crear una subclase de Inventory**, llamada **InventoryTestable**, en la que sobrescribimos el método parseInventory(). En el contexto de las pruebas, esta subclase carga un JSON predefinido en memoria en lugar de leer un archivo externo. De este modo, evitamos depender de la instrumentación de *Mockito* para modificar el comportamiento de la clase y aseguramos que las pruebas sean más estables y predecibles.

Esta solución permitió continuar con las pruebas unitarias sin depender de las limitaciones impuestas por *Mockito* y *Byte Buddy* en *Java 23*.

**Solución propuesta:**

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;

/**
 *
 * @author Brian Valiente Rodenas
 *
 * Inventory for the coffee maker
 */
public class Inventory {

    private static int coffee;
    private static int milk;
    private static int sugar;
    private static int chocolate;
                    private        final        String        path        =
"src\\main\\java\\com\\coffemaker\\data\\Inventario.json";

    /**
     * Constructor por defecto, carga el inventario desde la ruta.
     */
    @SuppressWarnings("OverridableMethodCallInConstructor")
    public Inventory() {
        try {
            String content = new String(Files.readAllBytes(Paths.get(path)),
StandardCharsets.UTF_8);
```

```
            parseInventory(content);
        } catch (IOException e) {
            setDefaults();
        }
    }

    @SuppressWarnings("OverridableMethodCallInConstructor")
    public Inventory(String pathJSON) {
        try {
                                        String      content      =      new
String(Files.readAllBytes(Paths.get(pathJSON)), StandardCharsets.UTF_8);
            parseInventory(content);
        } catch (IOException e) {
            setDefaults();
        }
    }

    public void parseInventory(String jsonContent) {
        Gson gson = new Gson();
        JsonObject json = gson.fromJson(jsonContent, JsonObject.class);
        JsonArray ingredientes = json.getAsJsonArray("ingredientes");
        for (int i = 0; i < ingredientes.size(); i++) {
            JsonObject ing = ingredientes.get(i).getAsJsonObject();
            String nombre = ing.get("nombre").getAsString();
            int cantidad = ing.get("cantidad").getAsInt();
            switch (nombre.toLowerCase()) {
                case "coffee" -> setCoffee(cantidad);
                case "milk" -> setMilk(cantidad);
                case "sugar" -> setSugar(cantidad);
                case "chocolate" -> setChocolate(cantidad);
            }
        }
    }

    public void setDefaults() {
        setCoffee(15);
        setMilk(15);
        setSugar(15);
        setChocolate(15);
    }

// RESTO DEL CÓDIGO
```

**Subclase para pruebas:**

```
public class InventoryTestable extends Inventory {

    private final String jsonContent;

    public InventoryTestable(String jsonContent) {
        super("dummy");
        this.jsonContent = jsonContent;
    }

    @Override
    public void parseInventory(String ignoredContent) {
        // Se ignora el contenido recibido y se utiliza el JSON inyectado.
        super.parseInventory(jsonContent);
    }
}
```

## 12. Clase de prueba con Mock (JUnit y Mockito)

**Código de la clase de prueba:**

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
import static org.mockito.ArgumentMatchers.anyString;

public class InventoryJsonTest {

    @Test
    public void testInventoryInitializationWithMock() {
        String jsonMock = """
        {
            "ingredientes": [
                {"nombre": "coffee", "cantidad": 12},
                {"nombre": "milk", "cantidad": 14},
                {"nombre": "sugar", "cantidad": 10},
                {"nombre": "chocolate", "cantidad": 12}
            ]
        }
        """;

        Inventory inventory = new InventoryTestable(jsonMock);
        inventory.parseInventory(anyString());

          assertEquals(12, inventory.getCoffee(), "El valor de coffee no
coincide.");
           assertEquals(14, inventory.getMilk(), "El valor de milk no
coincide.");
```

```
            assertEquals(10,  inventory.getSugar(),  "El  valor  de  sugar  no
coincide.");
        assertEquals(12, inventory.getChocolate(), "El valor de chocolate no
coincide.");
    }
}
```

# 13. Configuración del entorno y Maven

Para trabajar con Gson y manejar ficheros JSON se realizó lo siguiente:

1. **Instalación y configuración:**
    - Descargar Maven y JDK.
    - Configurar las variables de entorno JAVA_HOME y MAVEN_HOME.
    - Instalar los plugins necesarios en Visual Studio Code.
2. **Creación de un nuevo proyecto Maven:**
    - Se seleccionó un arquetipo (por ejemplo, maven-archetype-quickstart).
    - Se definieron los valores de groupId, artifactId y demás configuraciones.
3. **Modificación del archivo pom.xml.** Se añadieron las siguientes **dependencias**:

```
<dependencies>
    <!-- Gson para trabajar con JSON -->
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.10</version>
    </dependency>

    <!-- JUnit para pruebas unitarias -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <scope>test</scope>
    </dependency>

    <!-- Soporte para pruebas parametrizadas en JUnit -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-params</artifactId>
      <scope>test</scope>
    </dependency>
```

```xml
    <!-- Mockito para mockear objetos en pruebas unitarias -->
    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-core</artifactId>
      <version>5.8.0</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-junit-jupiter</artifactId>
      <version>5.8.0</version>
      <scope>test</scope>
    </dependency>

    <!-- Mockito para mockear las clases finales y estáticas -->
    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-inline</artifactId>
      <version>5.2.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

## Conclusión

Se han corregido los errores detectados en los métodos de las clases **CoffeeMaker**, **Recipe**, **Inventory** y **Main**, y se ha adaptado la clase **Inventory** para inicializar sus datos desde un fichero JSON. Además, se ha implementado una solución para testear la inicialización del inventario mediante una subclase (InventoryTestable) para evitar problemas con Mockito y Java 23. Se han aplicado buenas prácticas de codificación y se han eliminado redundancias y código no utilizado. Aquí adjuntamos la disposición del proyecto y el resultado satisfactorio de todos los test realizados.