**What is the Singleton Pattern?**

**Task 1 - IMPLEMENTATION OF SINGLETON PATTERN**

The Singleton Pattern ensures that a class has only one instance, and provides a global access point to that instance.

**Real-world Use Cases:**

* Database connection pool
* Application configuration settings
* Logger service
* Scheduler system

**How This Code Works**

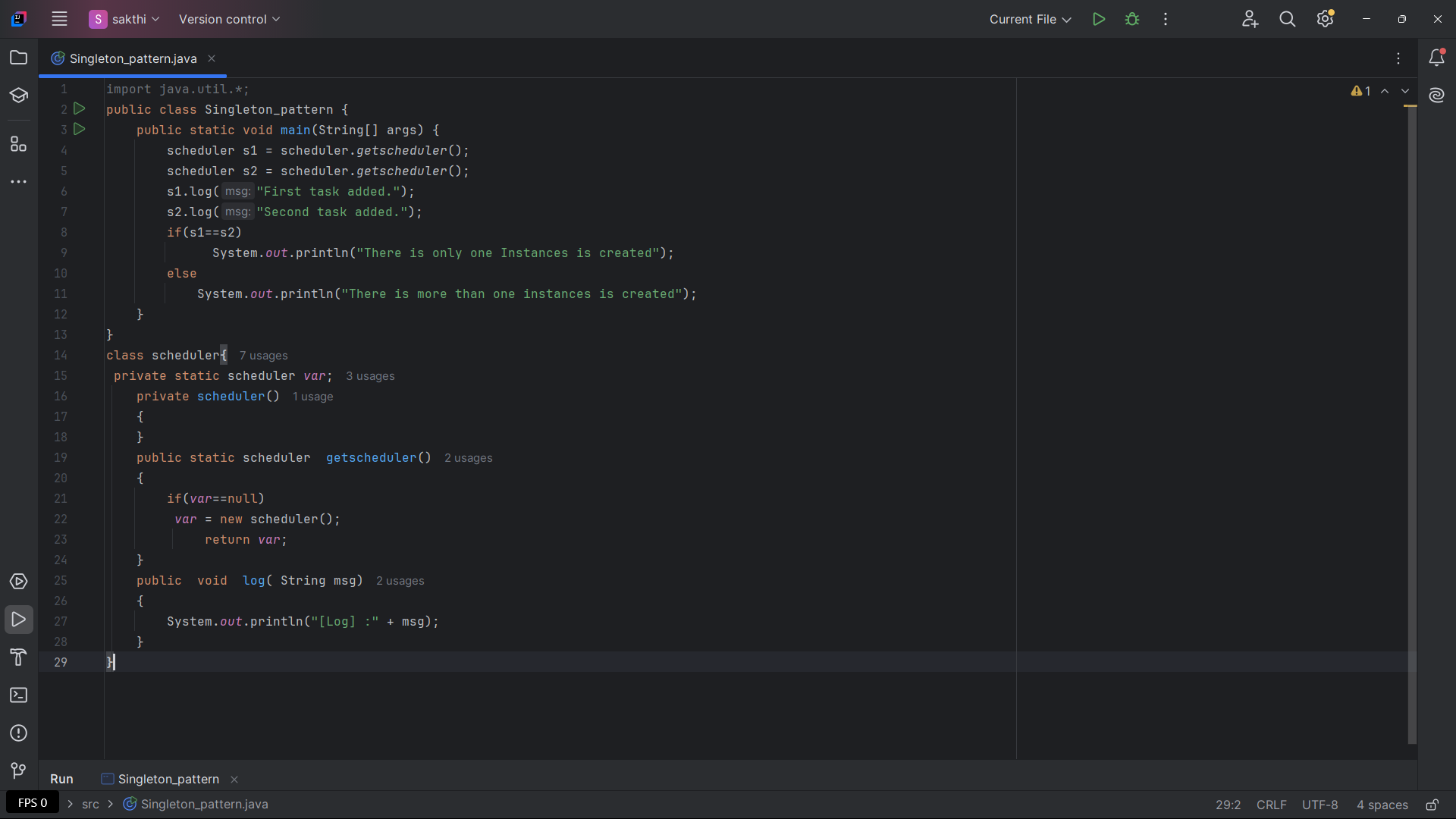
- Scheduler class:

* Has a private constructor.
* Contains a private static instance.
* Offers a public static method (`getscheduler()`) to access the single object.
* Includes a `log(String msg)` method to simulate task logging.

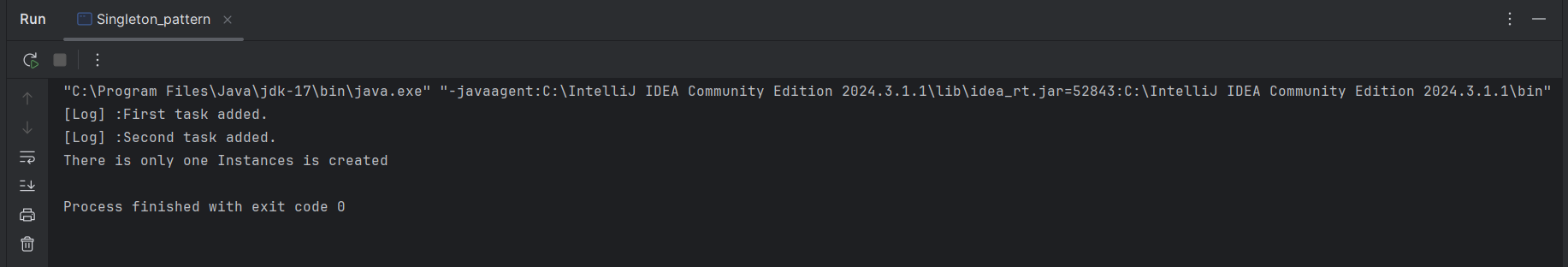
- In “main()”:

* Two references (`s1`, `s2`) are created using `getscheduler()`
* Both call the `log()` method
* A comparison (`s1 == s2`) checks whether both references point to the same object

**SAMPLE CODE :**



**OUTPUT:**



**Task 2 - IMPLEMENTATION OF FACTORY METHOD PATTERN**

**What is Factory Design Pattern?**

* The Factory Design Pattern is a creational pattern that provides a way to create objects without exposing the object creation logic to the client. It delegates the instantiation to subclasses (factories).

**Real-Life EXAMPLE**

* Imagine an app that takes an order for Pizza or Burger. Instead of manually creating the object using `new`, we delegate the task to the factory, which returns the correct object.

**HOW THIS CODE WORKS:**

**1. Food Interface:**

* Acts as a **common blueprint** for all food items
* Declares a method prepare() that must be implemented by all food types (like pizza, burger)

2. **pizza and burger Classes:**

* Both **implement the Food interface**
* Each one gives its own version of the prepare() method:
  + pizza returns: "Pizza is preparing"
  + burger returns: "burger is preparing"

3. **foodfactory - Abstract Class:**

* Defines an **abstract method** createfood()
* This method is a **placeholder** — actual logic will be written by child classes
* Acts as a **parent factory class** — it doesn't create food directly

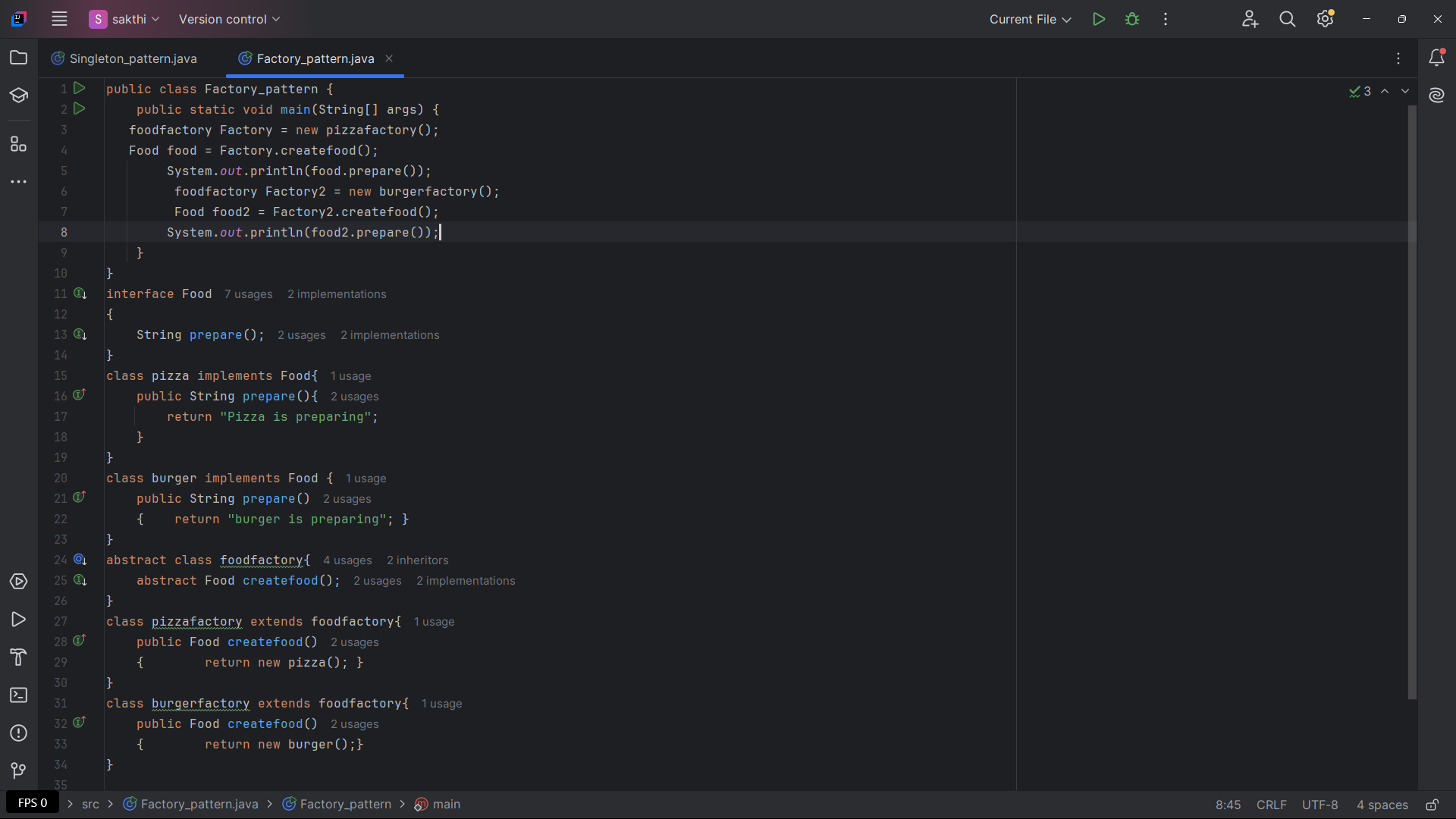
4. **pizzafactory and burgerfactory Classes:**

* These are the **concrete factory classes**
* Each one **extends foodfactory** and provides actual logic for createfood():
  + pizzafactory returns a new pizza() object
  + burgerfactory returns a new burger() object

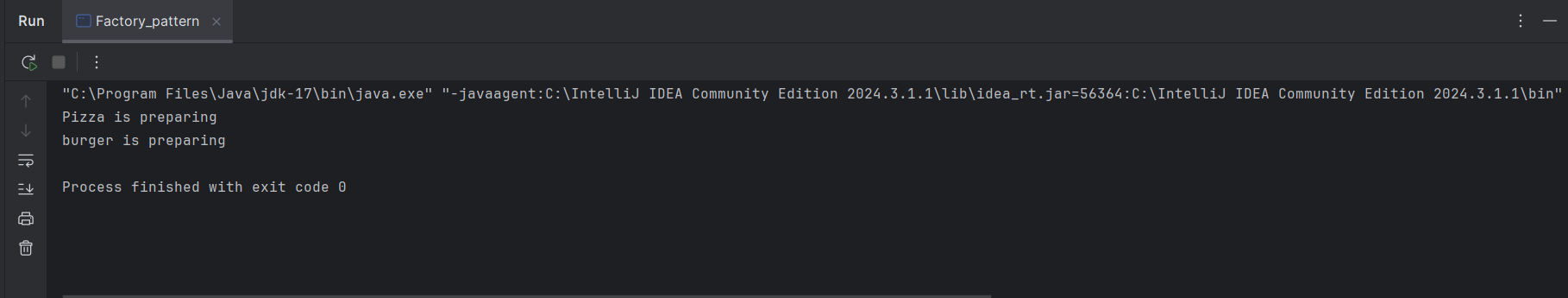
**5.In main():**

* A reference Factory of type foodfactory is created and assigned a pizzafactory object
* createfood() is called through this factory, which gives a pizza object
* prepare() is called on the pizza object to show preparation message
* Same steps are repeated using a burgerfactory:
  + A new burger object is created and prepared

**SAMPLE CODE**



**OUTPUT:**

****