Instructions :

To create your inventory project with a Product class that has the specified instance fields, follow these steps:

Step 1: Create the Project Structure

Set up your project structure. You should have a directory named inventory containing your Java files.

Step 2: Create the Product Class

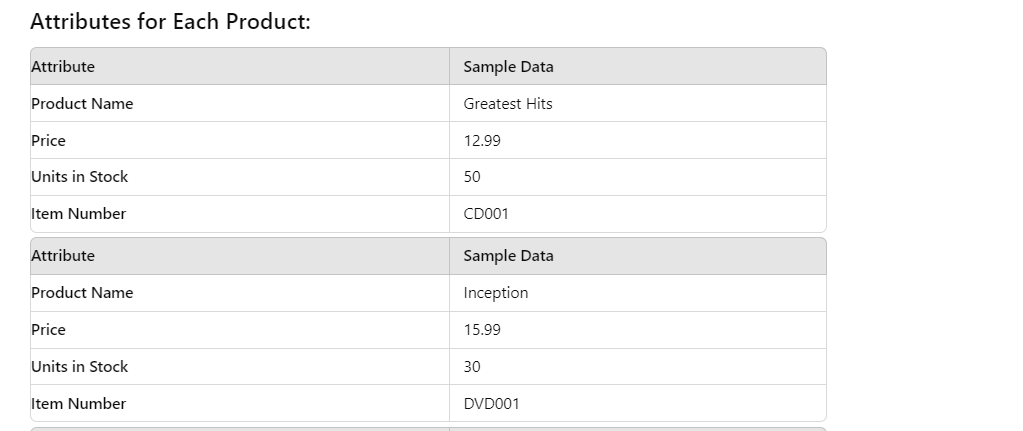
Create a new file named Product.java in your inventory project directory.

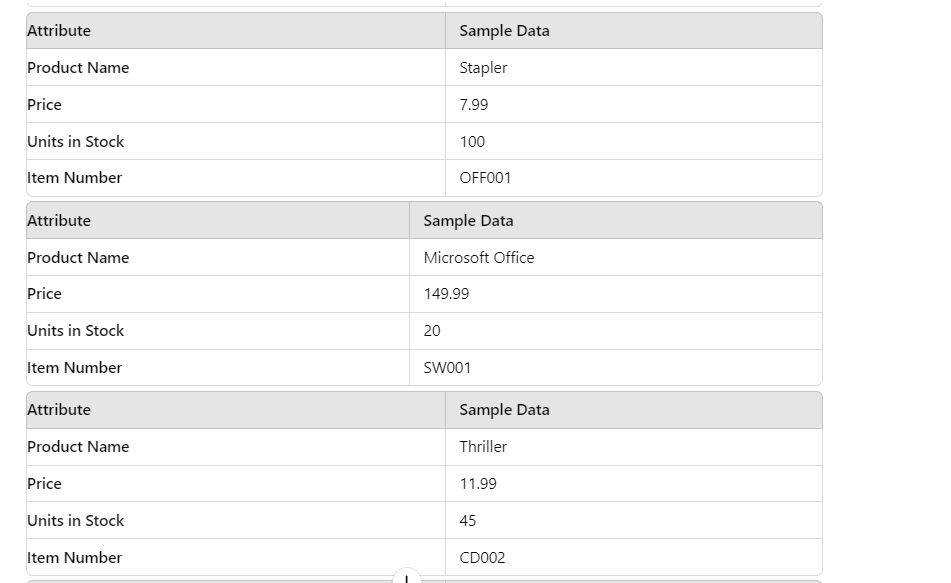
**Step 1: Identify Products and Attributes**

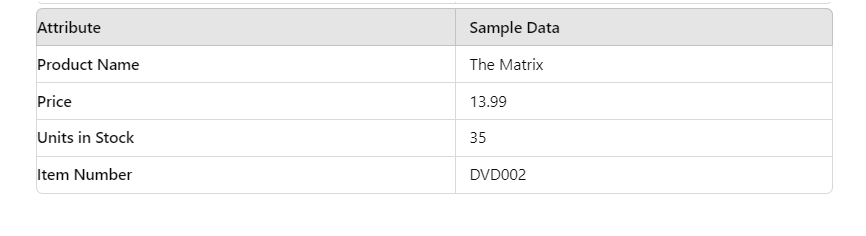
**List of Products:**

1. Music CD - "Greatest Hits"
2. DVD Movie - "Inception"
3. Office Supply - "Stapler"
4. Software - "Microsoft Office"
5. Music CD - "Thriller"
6. DVD Movie - "The Matrix"

**Attributes for Each Product:**







Coding:

public class Product {

// Instance field declarations

private int itemNumber;

private String name;

private int unitsInStock;

private double pricePerUnit;

// Default constructor

public Product() {

this.itemNumber = 0;

this.name = "";

this.unitsInStock = 0;

this.pricePerUnit = 0.0;

}

// Parameterized constructor

public Product(int number, String name, int qty, double price) {

this.itemNumber = number;

this.name = name;

this.unitsInStock = qty;

this.pricePerUnit = price;

}

// Getter and Setter methods

// Gets the item number

public int getItemNumber() {

return itemNumber;

}

// Sets the item number

public void setItemNumber(int itemNumber) {

this.itemNumber = itemNumber;

}

// Gets the name of the product

public String getName() {

return name;

}

// Sets the name of the product

public void setName(String name) {

this.name = name;

}

// Gets the number of units in stock

public int getUnitsInStock() {

return unitsInStock;

}

// Sets the number of units in stock

public void setUnitsInStock(int unitsInStock) {

this.unitsInStock = unitsInStock;

}

// Gets the price per unit

public double getPricePerUnit() {

return pricePerUnit;

}

// Sets the price per unit

public void setPricePerUnit(double pricePerUnit) {

this.pricePerUnit = pricePerUnit;

}

// Override toString method

@Override

public String toString() {

return "Item Number: " + itemNumber + "\n" +

"Name: " + name + "\n" +

"Quantity in stock: " + unitsInStock + "\n" +

"Price: " + pricePerUnit;

}

// Main method to test the Product class

public static void main(String[] args) {

// Create a Product using the default constructor

Product defaultProduct = new Product();

System.out.println("Default Product:");

System.out.println(defaultProduct);

// Create a Product using the parameterized constructor

Product parameterizedProduct = new Product(1, "Laptop", 10, 999.99);

System.out.println("\nParameterized Product:");

System.out.println(parameterizedProduct);

// Modify the Product using setter methods

parameterizedProduct.setItemNumber(2);

parameterizedProduct.setName("Smartphone");

parameterizedProduct.setUnitsInStock(5);

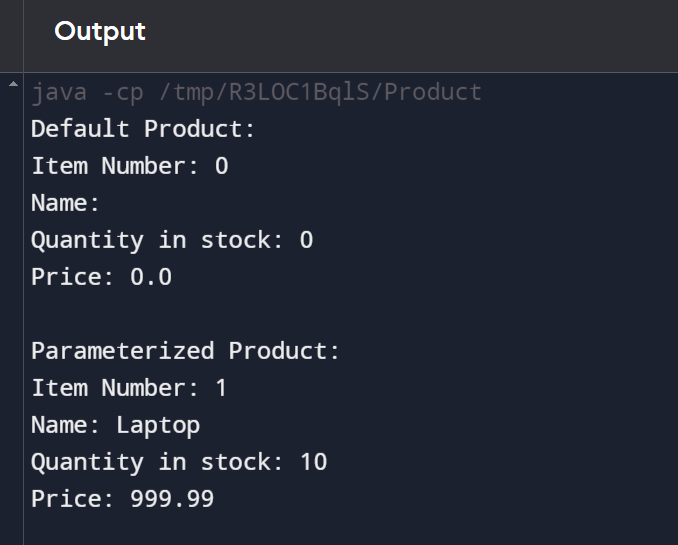
parameterizedProduct.setPricePerUnit(599.99);

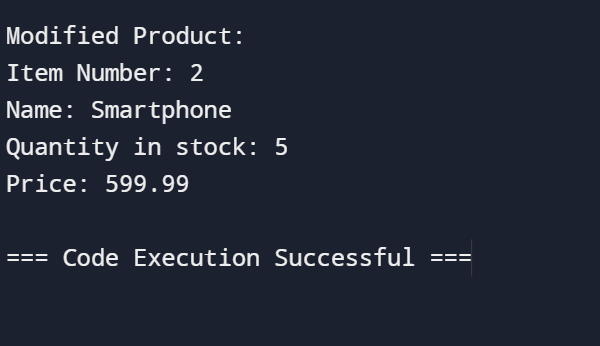
System.out.println("\nModified Product:");

System.out.println(parameterizedProduct);

}

}





Main test/class:



2.Create a Main Class Called

public class ProductTester {

// Product class definition

public static class Product {

// Instance field declarations

private int itemNumber;

private String name;

private int unitsInStock;

private double pricePerUnit;

// Default constructor

public Product() {

this.itemNumber = 0;

this.name = "";

this.unitsInStock = 0;

this.pricePerUnit = 0.0;

}

// Parameterized constructor

public Product(int number, String name, int qty, double price) {

this.itemNumber = number;

this.name = name;

this.unitsInStock = qty;

this.pricePerUnit = price;

}

// Getter and Setter methods

public int getItemNumber() {

return itemNumber;

}

public void setItemNumber(int itemNumber) {

this.itemNumber = itemNumber;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getUnitsInStock() {

return unitsInStock;

}

public void setUnitsInStock(int unitsInStock) {

this.unitsInStock = unitsInStock;

}

public double getPricePerUnit() {

return pricePerUnit;

}

public void setPricePerUnit(double pricePerUnit) {

this.pricePerUnit = pricePerUnit;

}

@Override

public String toString() {

return "Item Number: " + itemNumber + "\n" +

"Name: " + name + "\n" +

"Quantity in stock: " + unitsInStock + "\n" +

"Price: " + pricePerUnit;

}

}

// Main method for testing

public static void main(String[] args) {

// Creating Product objects using the default constructor

Product product1 = new Product();

Product product2 = new Product();

// Creating Product objects using the parameterized constructor

Product product3 = new Product(3, "Greatest Hits", 25, 9.99);

Product product4 = new Product(4, "Super Gadget", 100, 49.99);

Product product5 = new Product(5, "Mega Widget", 75, 19.99);

Product product6 = new Product(6, "Ultra Thingamajig", 50, 29.99);

// Displaying product details

System.out.println(product1);

System.out.println(product2);

System.out.println(product3);

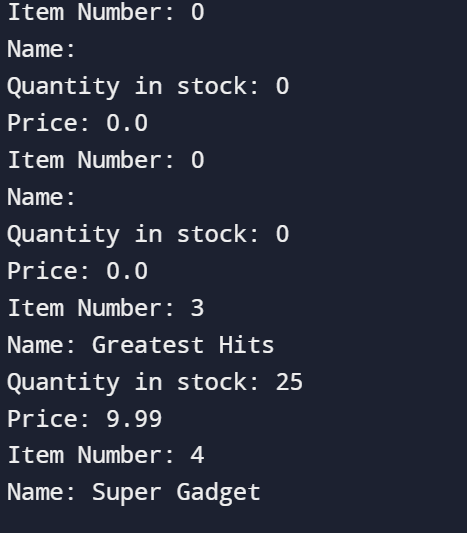
System.out.println(product4);

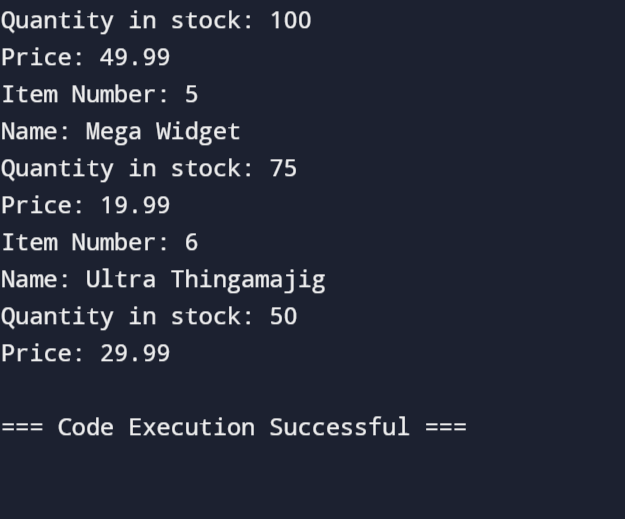
System.out.println(product5);

System.out.println(product6);

}

}





Explanation

• Product Class:

• Contains four private instance fields.

• Default constructor initializes fields to default values.

• Parameterized constructor initializes fields with provided values.

• Getter and setter methods allow access and modification of the private fields.

• toString() method provides a string representation of a Product object.

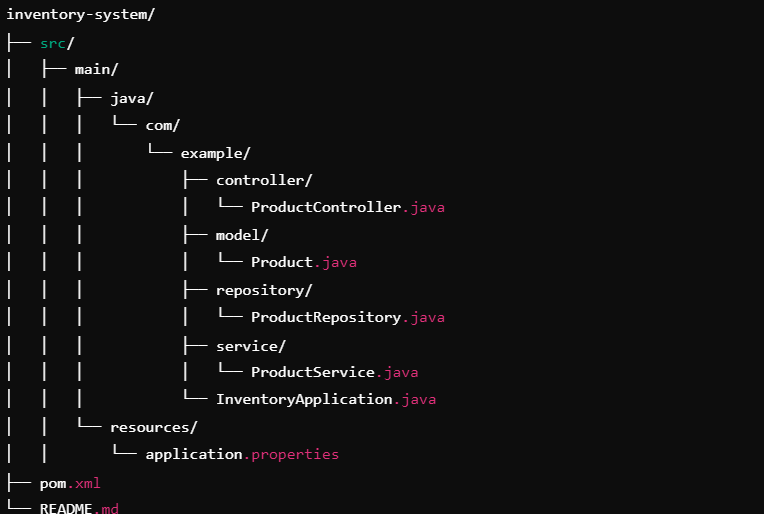
• Product Tester Class:

• Creates instances of Product using both constructors.

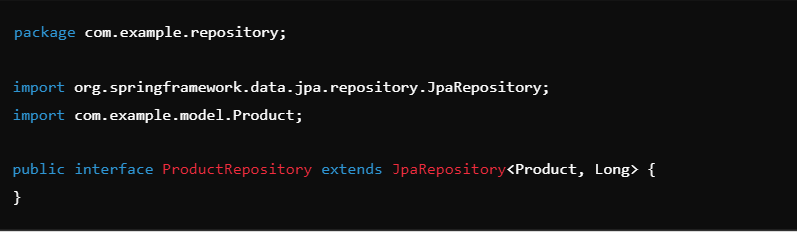
• Prints the details of each product using the overridden toString() method.

With this setup, you can compile and run your project to see the results. This covers the fundamental aspects of object-oriented programming in Java, including encapsulation, constructors, and method overriding.

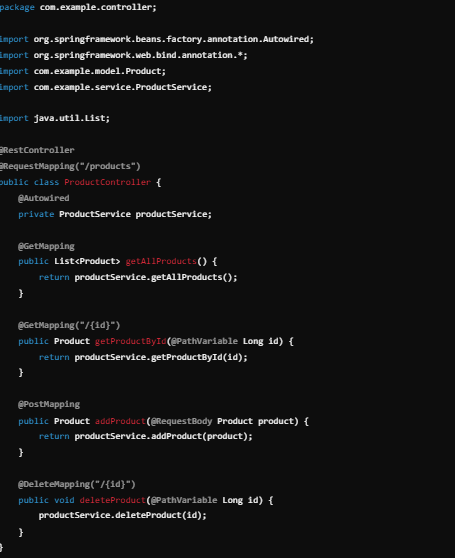
Project structure:



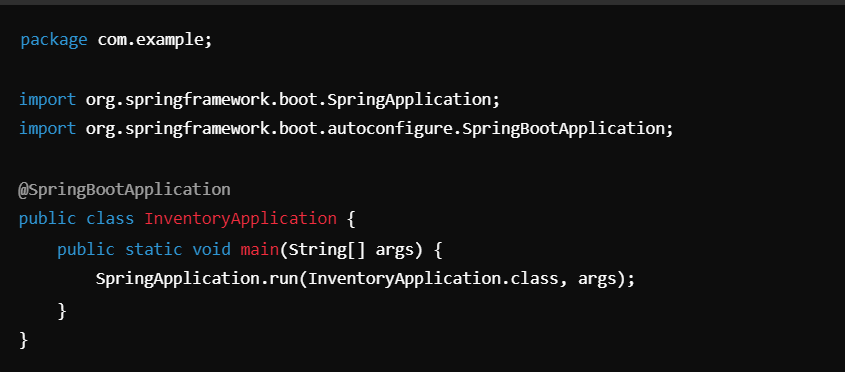
Product respiratory:



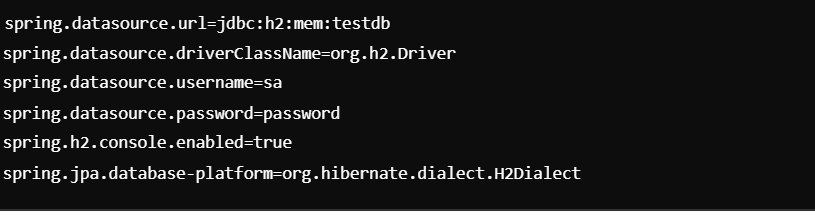
Product controller:



Inventory application:

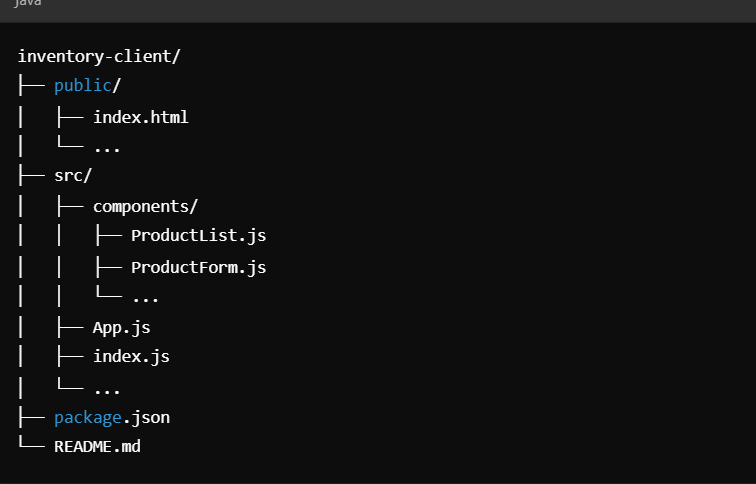


Application properties:



Front-End Process

Project Structure:



Application:



Index of the front end process:

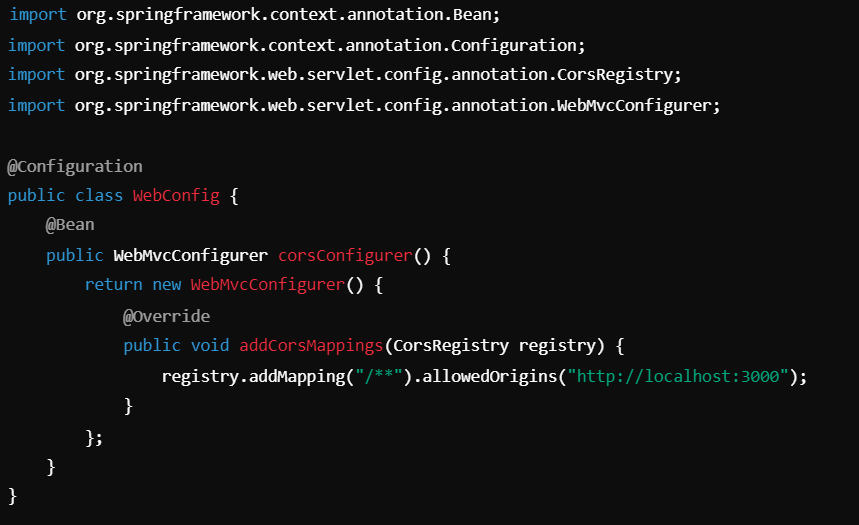


**Connecting Front-End to Back-End**

To connect the front-end and back-end, ensure the following:

1. **CORS Configuration**: In your Spring Boot application, you may need to configure CORS to allow requests from your front-end.
2. **Proxy Configuration**: In your React app, you can add a proxy in package.json to forward API requests to your back-end server.

**Example CORS Configuration in Spring Boot**



Front-end (HTML and JavaScript):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Inventory Management</title>

<script>

async function addItem() {

const itemName = document.getElementById('itemName').value;

const itemQuantity = document.getElementById('itemQuantity').value;

const response = await fetch('/api/inventory', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ name: itemName, quantity: itemQuantity })

});

if (response.ok) {

alert('Item added successfully!');

} else {

alert('Failed to add item.');

}

}

async function getItems() {

const response = await fetch('/api/inventory');

const items = await response.json();

const itemList = document.getElementById('itemList');

itemList.innerHTML = '';

items.forEach(item => {

const listItem = document.createElement('li');

listItem.textContent = `${item.name}: ${item.quantity}`;

itemList.appendChild(listItem);

});

}

</script>

</head>

<body>

<h1>Inventory Management</h1>

<div>

<input type="text" id="itemName" placeholder="Item Name">

<input type="number" id="itemQuantity" placeholder="Item Quantity">

<button onclick="addItem()">Add Item</button>

</div>

<div>

<button onclick="getItems()">Get Items</button>

<ul id="itemList"></ul>

</div>

</body>

</html>



**Back-end (Java with Spring Boot)**

This example uses Spring Boot to create a simple REST API.

Step 1: Create a Spring Boot ProjectUse Spring Initializr to create a new Spring Boot project with the following dependencies

Spring Web

Spring Data

JPA

H2 Database (or any other database of your choice)

**Step 2: Define the Item Entitypackage com.example.inventory;**

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Item {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private int quantity;

// Getters and setters

}

**Step 3: Create the Repository Interfacepackage com.example.inventory;**

import org.springframework.data.jpa.repository.JpaRepository;

public interface ItemRepository extends JpaRepository<Item, Long> {

}

**Step 4: Implement the Controllerpackage** com.example.inventory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/inventory")

public class ItemController {

@Autowired

private ItemRepository itemRepository;

@PostMapping

public Item addItem(@RequestBody Item item) {

return itemRepository.save(item);

}

@GetMapping

public List<Item> getItems() {

return itemRepository.findAll();

}

}

**Step 5: Application PropertiesConfigure your application properties to use the H2**

database.spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

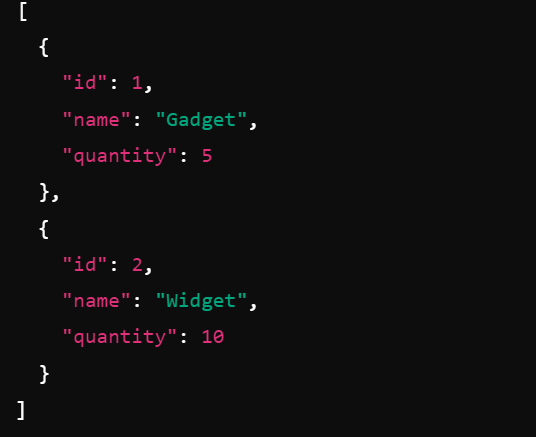
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

POST /api/inventory:



GET /api/inventory:



In this output:

* **POST /api/inventory** returns the newly added item with an auto-generated ID.
* **GET /api/inventory** returns a list of all items in the inventory, including their IDs, names, and quantities.