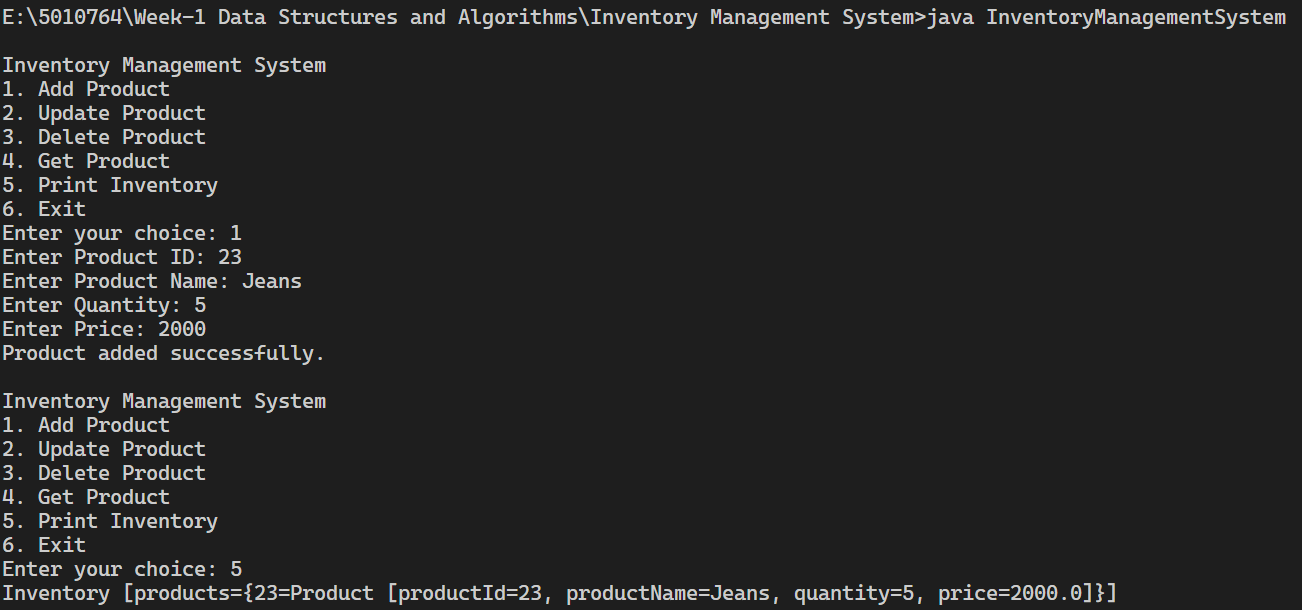
**INVENTORY MANAGEMENT SYSTEM:**

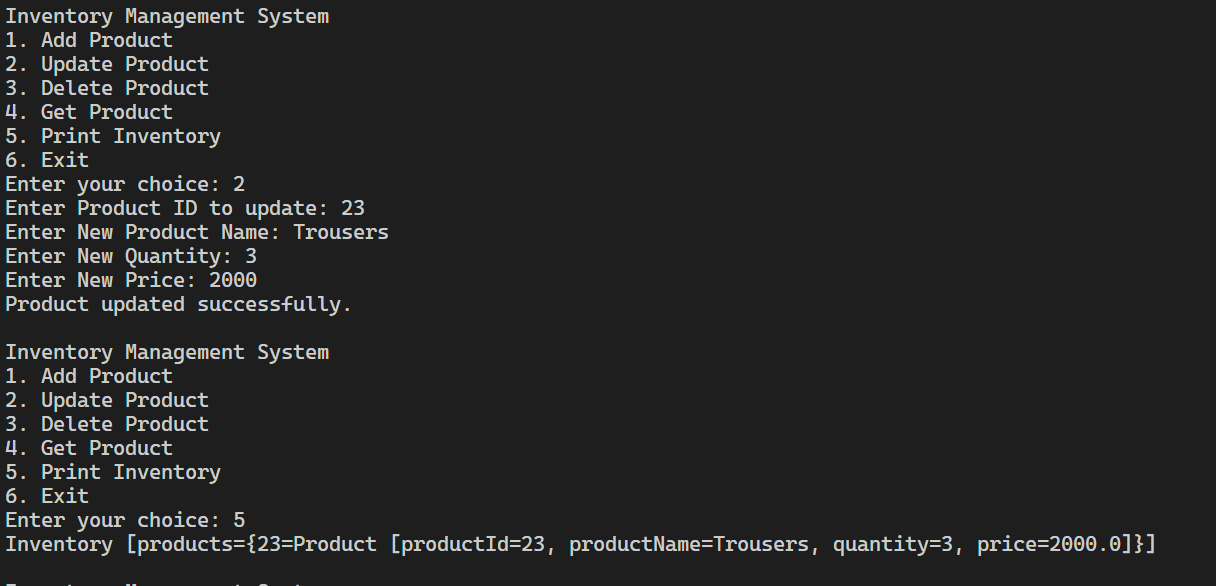
1. **Understanding the Problem Requirements**
   * **Efficiency: Choosing suitable data structures ensures efficient data handling, including storing, retrieving, and manipulating information. In an inventory system, it's important that operations such as adding, updating, and deleting products are performed swiftly and effectively.**
   * **Scalability: Effective algorithms and data structures allow the system to manage increasing amounts of data without significant slowdowns in performance.**
   * **Data Integrity: Appropriate data structures help maintain consistency and reliability in the inventory data.**

**Types of data structures suitable for this problem:**

* + **ArrayList: Offers quick access to elements and is useful when the inventory size frequently changes.**
  + **HashMap: Provides an average time complexity of O(1) for insertion, deletion, and lookup, making it ideal for large inventories where rapid access to product information is necessary.**

1. **Setup**
   * **A new Java project titled InventoryManagementSystem was established.**
2. **Implementation**
   * **A Product class was defined with attributes like productId, productName, quantity, and price.**
   * **A suitable data structure was selected to store the products:**
     + **In this case, a HashMap was used, where the key is the productId and the value is the Product object.**
   * **Methods were implemented for managing products in the inventory:**
     + **addProduct(Product product): Adds a product to the inventory.**
     + **updateProduct(String productId, Product updatedProduct): Updates existing product details.**
     + **deleteProduct(String productId): Removes a product from the inventory.**
     + **getProduct(String productId): Fetches a product from the inventory.**
3. **Analysis**
   * **Time complexity of operations in the HashMap data structure:**
     + **Add Product (addProduct): O(1) on average.**
     + **Update Product (updateProduct): O(1) on average.**
     + **Delete Product (deleteProduct): O(1) on average.**
     + **Get Product (getProduct): O(1) on average.**
   * **Optimizing Operations:**
     + **Indexing: Ensuring that each productId is unique helps avoid collisions in the HashMap.**
     + **Batch Operations: For bulk updates or deletions, batch processing can minimize the overhead of performing multiple operations.**
4. **Output**

****

****