



# Programming Fundamentals 1

---

Produced by Mr. Dave Drohan ([david.drohan@setu.ie](mailto:david.drohan@setu.ie))  
Dr. Siobhán Drohan

Ms. Mairead Meagher

Department of Computing & Mathematics  
South East Technological University  
Waterford, Ireland

setu.ie





# SHOP V2.0

## An Array of Products





# Agenda

---

☐ RECAP : SHOP V1.0

☐ SHOP V2.0



---

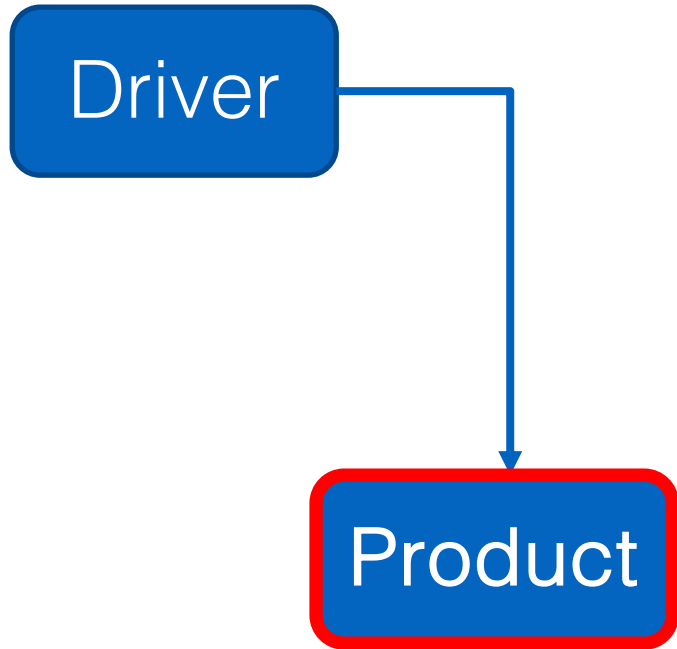
# RECAP : SHOP V1.0





# Recap: Shop V1.0 - Product

---



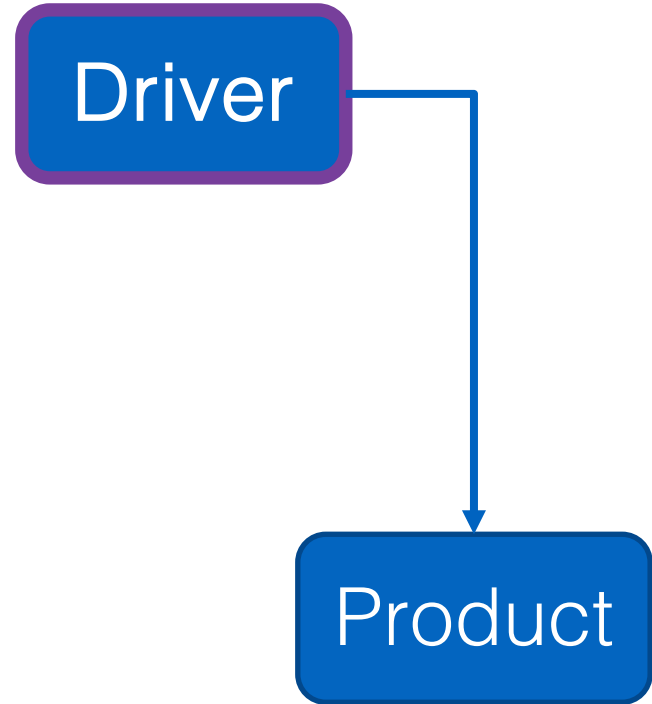
- ❑ The **Product** class stores details about a product
  - name
  - code
  - unit cost
  - in the current product line or not?



# Recap: Shop V1.0 - Driver

---

- The **Driver** class
  - has the **main()** method.
  - **reads** the product details from the user (via the console)
  - **creates** a new Product object.
  - **prints** the product object (to the console)





---

# SHOP V2.0

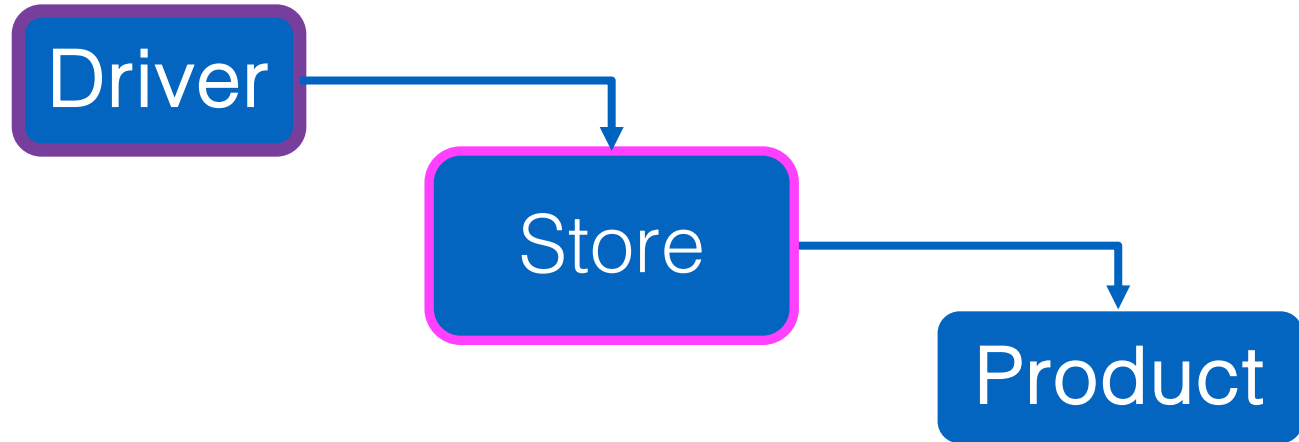




# Shop V2.0

---

- ❑ New **Store** class is responsible for maintaining a collection of Products
  - i.e. an array of Products.
- ❑ **Driver** will now allow the user to decide how many product details they want to store.

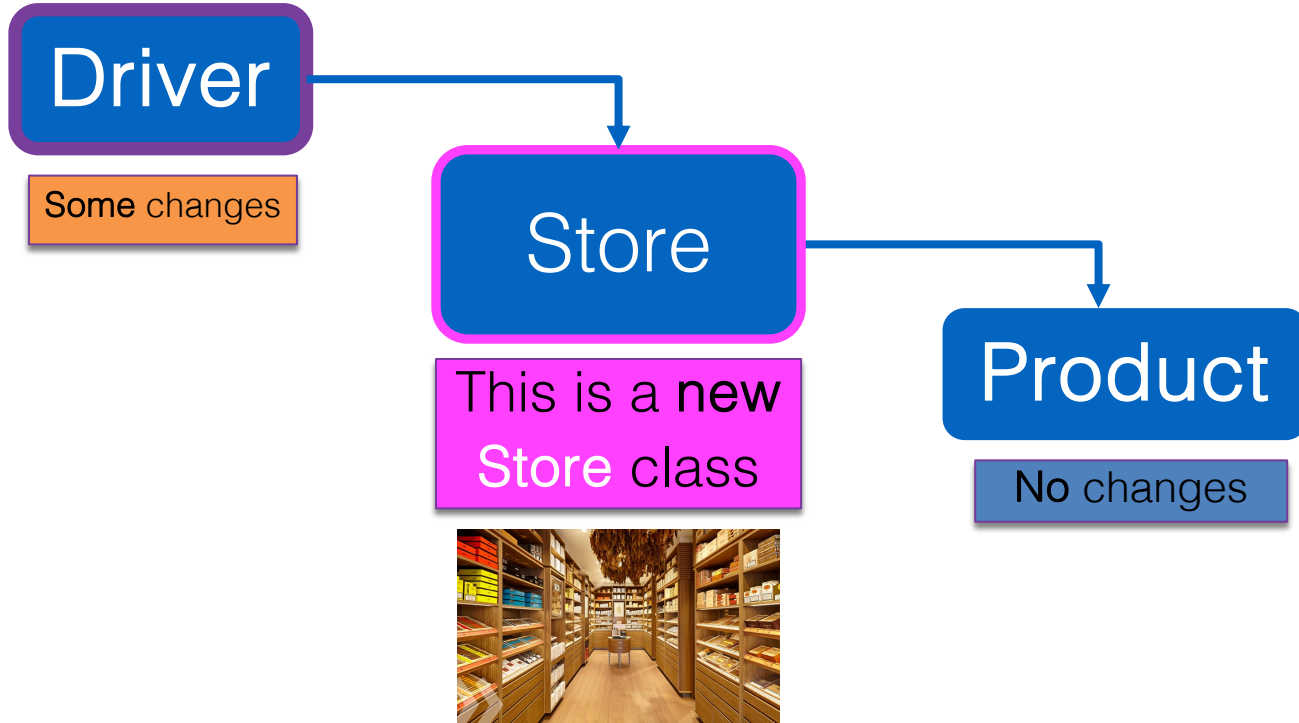






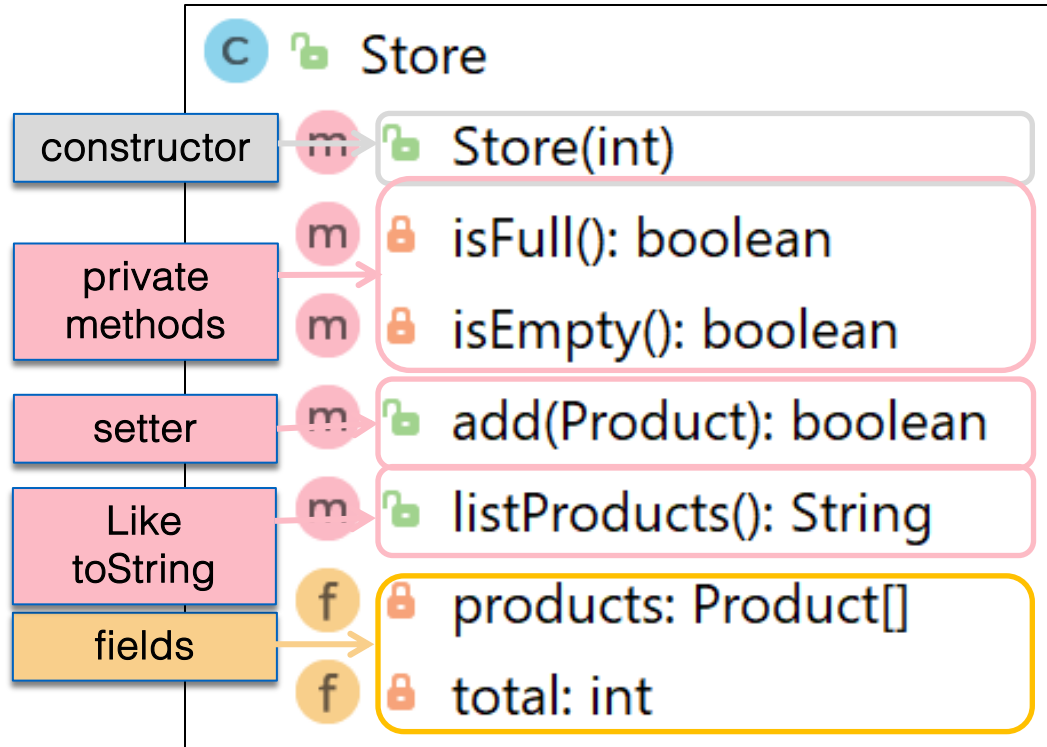
# Shop V2.0 – changes to classes

---





# Store – new class





**C** Store

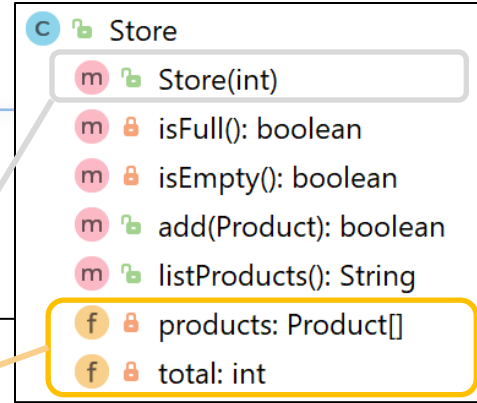
- m Store(int)
- m isFull(): boolean
- m isEmpty(): boolean
- m add(Product): boolean
- m listProducts(): String
- f products: Product[]
- f total: int

```
public class Store {  
    private Product[] products;  
    private int total = 0;  
  
    public Store(int numberItems) {  
        products = new Product[numberItems];  
    }  
  
    //other methods  
}
```

fields



Why private?



```
public class Store {
```

```
    private Product[] products;  
    private int total = 0;
```

fields

```
    public Store(int numberItems) {  
        products = new Product[numberItems];  
    }
```

constructor

Why private?

```
    //other methods
```

```
}
```



```
private boolean isFull() {  
    return (total == products.length);  
}  
  
private boolean isEmpty() {  
    return (total == 0);  
}  
  
public boolean add(Product product) {  
    if (isFull()) {  
        return false;  
    }  
    else {  
        products[total] = product;  
        total++;  
        return true;  
    }  
}
```

c	Store
m	Store(int)
m	isFull(): boolean
m	isEmpty(): boolean
m	add(Product): boolean
m	listProducts(): String
f	products: Product[]
f	total: int

**getters**  
isFull() & isEmpty()  
return state of fields.

They are private member methods

**setter**  
add() makes use of private method isFull()



```
public String listProducts() {  
    if (isEmpty()) {  
        return "No products";  
    }  
    else {  
        String listOfProducts = "";  
        for (int i = 0; i < total; i++) {  
            listOfProducts += i + ": " + products[i]  
+ "\n";  
        }  
        return listOfProducts;  
    }  
}
```

C Store

- m Store(int)
- m isFull(): boolean
- m isEmpty(): boolean
- m add(Product): boolean
- m listProducts(): String
- f products: Product[]
- f total: int

toString type method listProducts()  
makes use of private method  
isEmpty()



# Driver

5 changes

4) main() changed to call processOrder()

2) addProduct() changed to add the entered product to the array.

5) printProduct() changed to print out all products in the array.

3) New method, processOrder(), reads in products from the user.

```
Driver
m main(String[]): void
m addProduct(): void
m printProduct(): void
m processOrder(): void
f input: Scanner = new Scanner(...)
f store: Store
```

1) Product object removed and replaced with Store object.



# Driver

Change - 1

```
Driver
├── m main(String[]): void
├── m addProduct(): void
├── m printProduct(): void
├── m processOrder(): void
├── f input: Scanner = new Scanner(...)
└── f store: Store
```

```
import java.util.Scanner;

public class Driver{

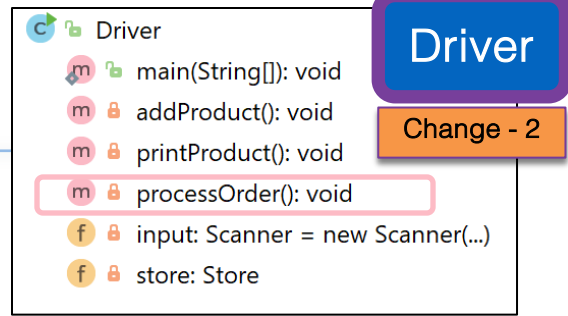
    private Scanner input = new Scanner(System.in);
    private Store store;

    //code omitted
}
```

1) Product object removed and replaced with Store object.



2) New method, `processOrder()`, reads in products from the user.



Driver

- m main(String[]): void
- m addProduct(): void
- m printProduct(): void
- m processOrder(): void
- f input: Scanner = new Scanner(...)
- f store: Store

Change - 2

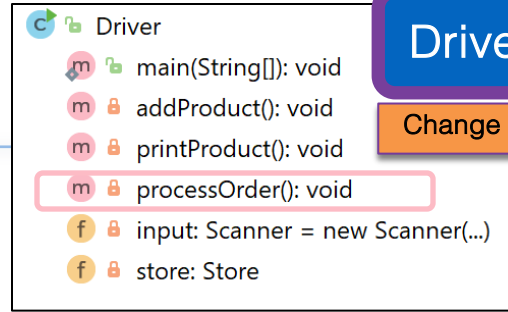
The image shows a snippet of a Java IDE. At the top right is a blue box with the word "Driver" in white. Below it is an orange box with "Change - 2". The main area shows a list of methods for the "Driver" class. Each method is preceded by an icon: a blue 'm' for methods and a yellow 'f' for fields. The method `processOrder(): void` is highlighted with a pink rectangular box.



```
private void processOrder() {  
    //find out from the user how many products they would like to order  
    System.out.print("How many Products would you like to have in your Store? ");  
    int numberProducts = input.nextInt();  
  
    store = new Store(numberProducts);  
  
    //ask the user for the details of the products and add them to the order  
    for (int i = 0; i < numberProducts; i++){  
        addProduct();  
    }  
}
```

- Asks how many?
- Pass into Store constructor to initialise an array to that size
- Calls `addProduct()` for each one

3) main() changed  
to call  
processOrder()



IDE snippet of the Driver class. The class name 'Driver' is highlighted in a blue box. Below it, a list of methods is shown: main(String[]): void, addProduct(): void, printProduct(): void, processOrder(): void, input: Scanner = new Scanner(...), and store: Store. The processOrder(): void method is highlighted with a pink box. An orange box labeled 'Change - 3' is positioned to the right of the class name.

```
Driver  
m main(String[]): void  
m addProduct(): void  
m printProduct(): void  
m processOrder(): void  
f input: Scanner = new Scanner(...)  
f store: Store
```



```
public static void main(String[] args) {  
    Driver driver = new Driver();  
    driver.processOrder();  
    driver.printProduct();  
}
```



## Store

If there is space in the Array, the Product, passed as a parameter is added to the Primitive Array.

A boolean result is returned indicating whether the product was added successfully or not.

```
c Store
m Store(int)
m isFull(): boolean
m isEmpty(): boolean
m add(Product): boolean
m listProducts(): String
f products: Product[]
f total: int
```

```
public boolean add(Product product){
    if (isFull()){
        return false;
    }
    else{
        products[total] = product;
        total++;
        return true;
    }
}
```

4) `addProduct()`  
changed to add the  
entered product to  
the array.

```
Driver
├── main(String[]): void
├── addProduct(): void
├── printProduct(): void
├── processOrder(): void
├── input: Scanner = new Scanner(...)
└── store: Store
```

Change - 4



## Driver

The `addProduct()` method needs to be updated to:

1. Add the product object to the array of products in Store
2. Interrogate the Boolean result returned to let the user know if the update was successful or not.



```
private void addProduct(){
    input.nextLine(); //dummy read of String to clear the buffer - bug in Scanner class.

    System.out.print("Enter the Product Name: ");
    String productName = input.nextLine();
    System.out.print("Enter the Product Code: ");
    int productCode = input.nextInt();
    System.out.print("Enter the Unit Cost: ");
    double unitCost = input.nextDouble();

    //Ask the user to type in either a Y or an N then convert to boolean value
    System.out.print("Is this product in your current line (y/n): ");
    char currentProduct = input.next().charAt(0);
    boolean inCurrentProductLine = false;
    if ((currentProduct == 'y') || (currentProduct == 'Y'))
        inCurrentProductLine = true;

    boolean isAdded = store.add(new Product(productName, productCode, unitCost, inCurrentProductLine));
    if (isAdded){
        System.out.println("Product Added Successfully");
    }
    else{
        System.out.println("No Product Added");
    }
}
```



```
private void addProduct(){  
    input.nextLine(); //dummy read of String to clear the buffer - bug in Scanner class.
```

```
    System.out.print("Enter the Product Name: ");  
    String productName = input.nextLine();  
    System.out.print("Enter the Product Code: ");  
    int productCode = input.nextInt();  
    System.out.print("Enter the Unit Cost: ");  
    double unitCost = input.nextDouble();
```

Read in a string

Read in an int

Read in a double

*//Ask the user to type in either a Y or an N then convert to boolean value*

```
    System.out.print("Is this product in your current line (y/n): ");  
    char currentProduct = input.next().charAt(0);  
    boolean inCurrentProductLine = false;  
    if ((currentProduct == 'y') || (currentProduct == 'Y'))  
        inCurrentProductLine = true;
```

Read in a char

Set boolean based on char value

```
    boolean isAdded = store.add(new Product(productName, productCode, unitCost, inCurrentProductLine));  
    if (isAdded){  
        System.out.println("Product Added Successfully");  
    }  
    else{  
        System.out.println("No Product Added");  
    }  
}
```



```
private void addProduct(){  
    input.nextLine(); //dummy read of String to clear the buffer - bug in Scanner class.
```

```
    System.out.print("Enter the Product Name: ");  
    String productName = input.nextLine();  
    System.out.print("Enter the Product Code: ");  
    int productCode = input.nextInt();  
    System.out.print("Enter the Unit Cost: ");  
    double unitCost = input.nextDouble();
```

```
    //Ask the user to type in either a Y or an N then convert to boolean value  
    System.out.print("Is this product in your current line (y/n): ");  
    char currentProduct = input.next().charAt(0);  
    boolean inCurrentProductLine = false;  
    if ((currentProduct == 'y') || (currentProduct == 'Y'))  
        inCurrentProductLine = true;
```

```
    boolean isAdded = store.add(new Product(productName, productCode, unitCost, inCurrentProductLine));  
    if (isAdded){  
        System.out.println("Product Added Successfully");  
    }  
    else{  
        System.out.println("No Product Added");  
    }  
}
```

The Store add method is called to add the product to the primitive array, if space is available.

Console response if add was successful

Console response if add was unsuccessful



Driver

```
Driver
m main(String[]): void
m addProduct(): void
m printProduct(): void
m processOrder(): void
f input: Scanner = new Scanner(...)
f store: Store
```

Change - 5

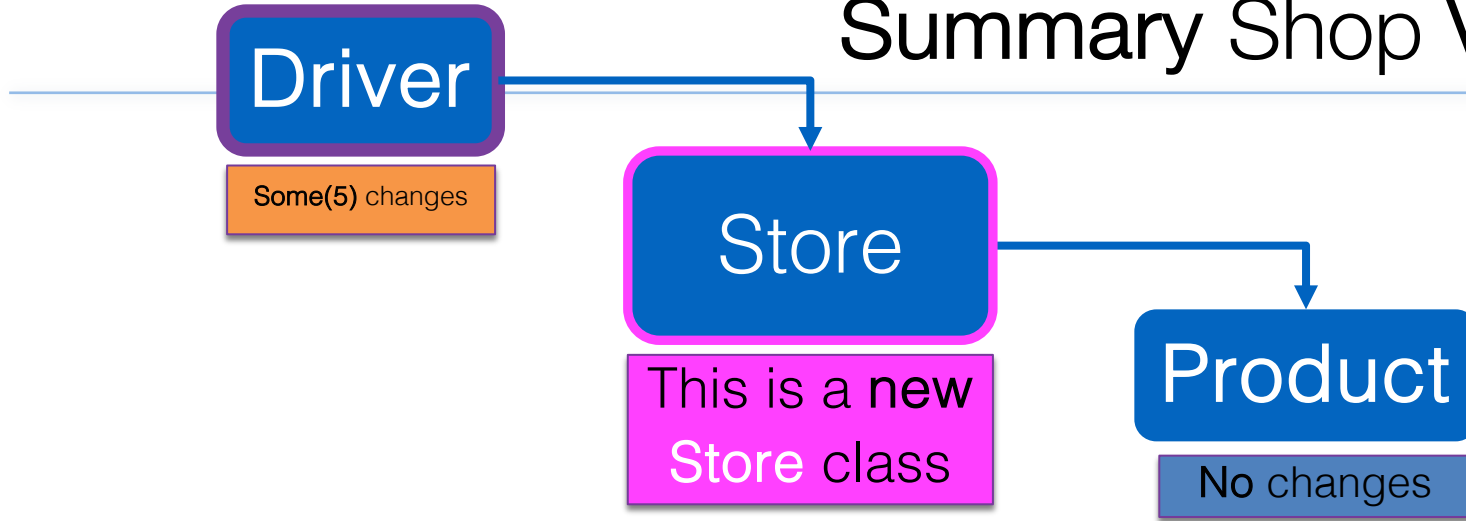
5) printProduct()  
changed to print  
out all products in  
the array.

```
private void printProduct () {
    System.out.println(store.listProducts());
}
```





# Summary Shop V2.0



- ❑ **Store** class maintains a collection of Products i.e. an **array of Products**; `store.Products[]`
- ❑ **Driver** allows the user to decide **how many product** details they want to store. Methods updated to work with this new `store.Products[]` array

# Questions?

---



Thanks.

