Recap of OO concepts

Objects, classes, methods and more.

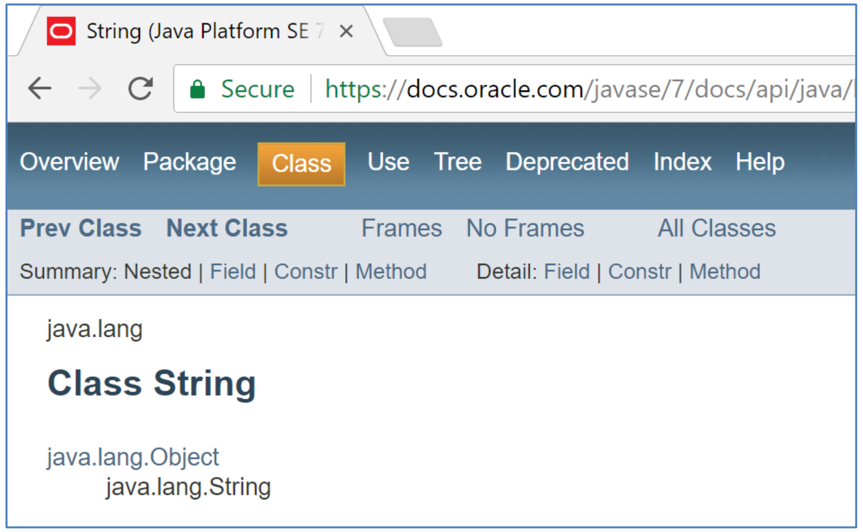
Produced by: Ms. Mairead Meagher
Ms Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.setu.ie/

Classes and Objects

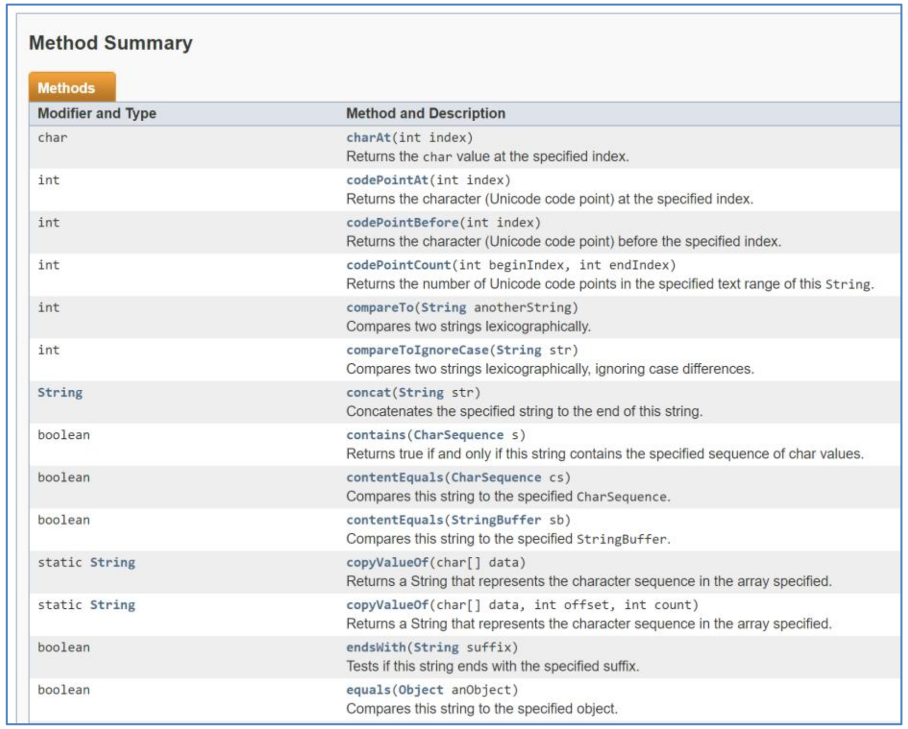
- A **class**
  - defines a group of related **methods** (functions) and **fields** (variables / properties).


String (Java Platform SE 7
Secure
https://docs.oracle.com/javase/7/docs/api/java/
Overview
Package
Class
Use
Tree
Deprecated
Index
Help
Prev Class
Next Class
Frames
No Frames
All Classes
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

java.lang

Class String

java.lang.Object
    java.lang.String



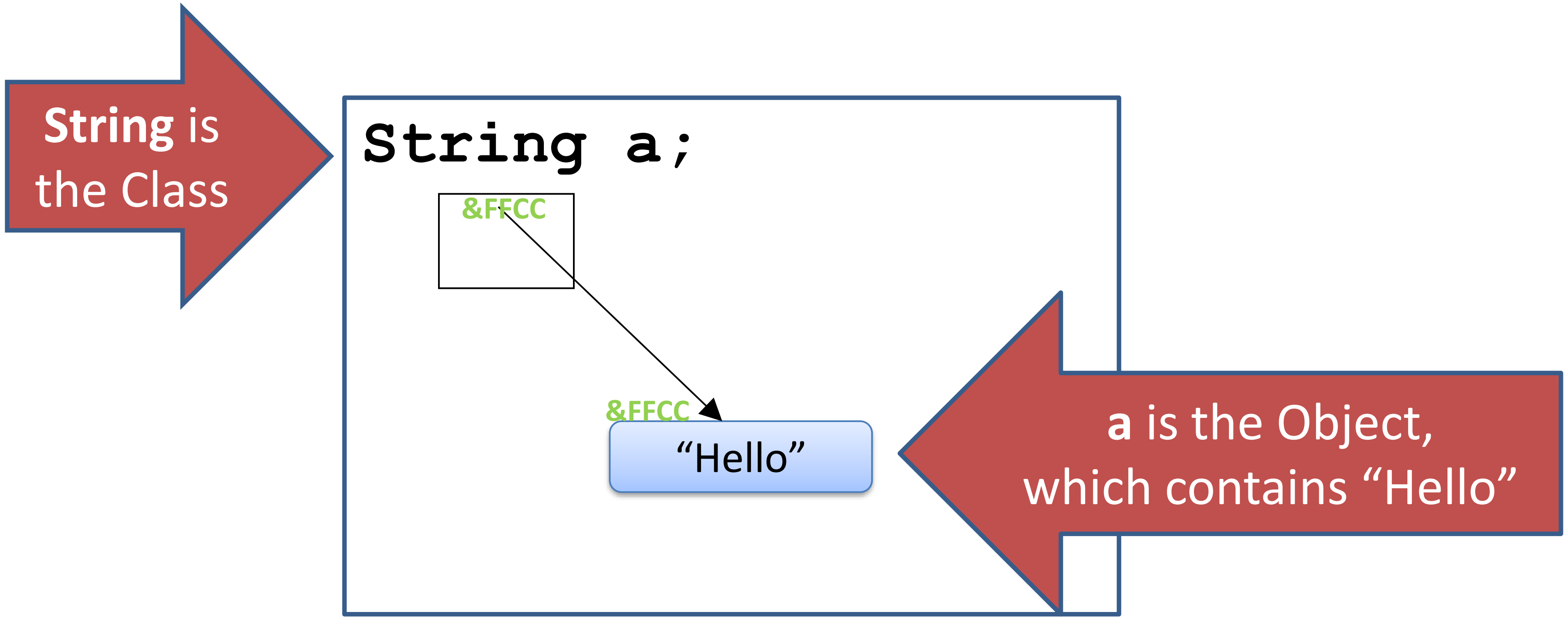
Method Summary

Methods

Modifier and Type	Method and Description
`char`	`charAt(int index)` Returns the `char` value at the specified index.
`int`	`codePointAt(int index)` Returns the character (Unicode code point) at the specified index.
`int`	`codePointBefore(int index)` Returns the character (Unicode code point) before the specified index.
`int`	`codePointCount(int beginIndex, int endIndex)` Returns the number of Unicode code points in the specified text range of this `String`.
`int`	`compareTo(String anotherString)` Compares two strings lexicographically.
`int`	`compareToIgnoreCase(String str)` Compares two strings lexicographically, ignoring case differences.
`String`	`concat(String str)` Concatenates the specified string to the end of this string.
`boolean`	`contains(CharSequence s)` Returns true if and only if this string contains the specified sequence of char values.
`boolean`	`contentEquals(CharSequence cs)` Compares this string to the specified `CharSequence`.
`boolean`	`contentEquals(StringBuffer sb)` Compares this string to the specified `StringBuffer`.
`static String`	`copyValueOf(char[] data)` Returns a String that represents the character sequence in the array specified.
`static String`	`copyValueOf(char[] data, int offset, int count)` Returns a String that represents the character sequence in the array specified.
`boolean`	`endsWith(String suffix)` Tests if this string ends with the specified suffix.
`boolean`	`equals(Object anObject)` Compares this string to the specified object.

Classes and **Objects**

- An **object**
  - is a single instance of a class
  - i.e. an object is created (instantiated) from a class.



Classes and Objects – **Many Objects**

- Many **objects** can be constructed from a single **class** definition.
- Each **object** must have a unique name within the program.

Ver 1.0

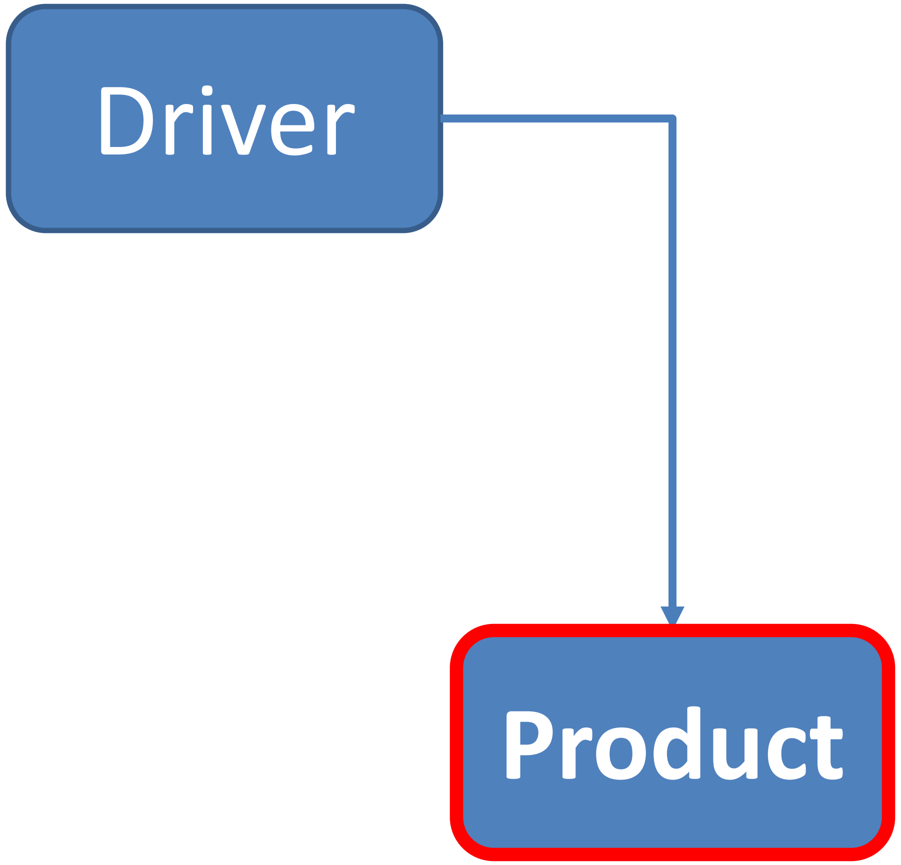
SHOP

Shop V1.0 - **Product**

- We will recap object oriented concepts through the study of a new class called **Product**.

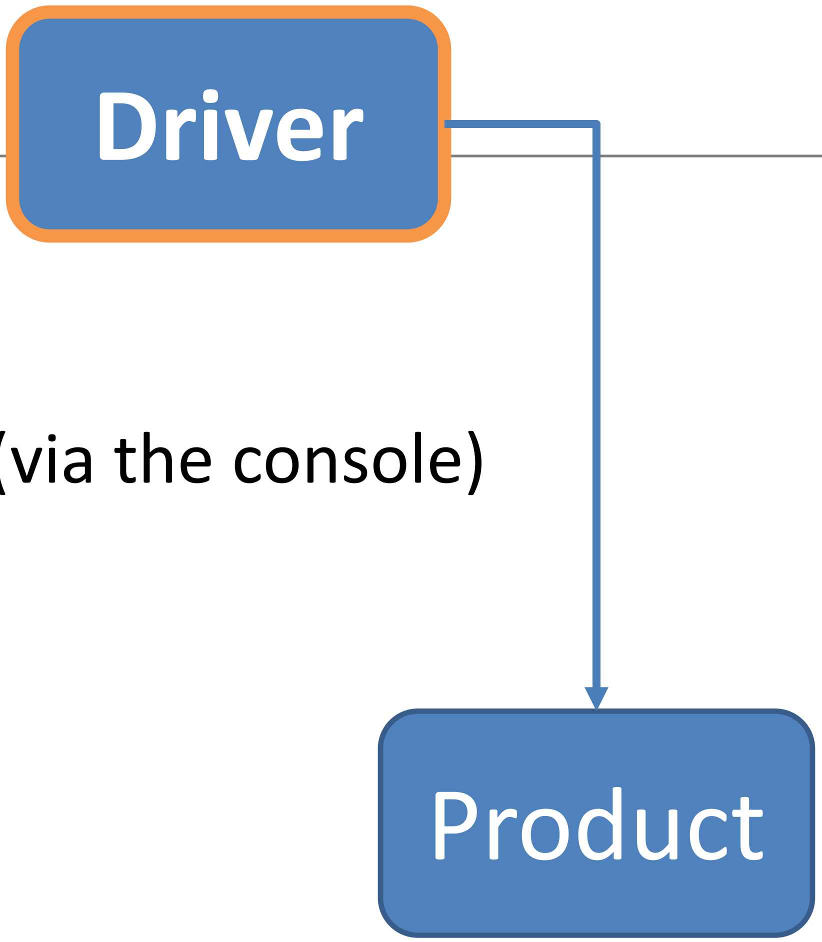
$$\begin{tikzcd}\boxed{\text{Driver}} \arrow[rd] & \\ & \boxed{\textbf{Product}}\end{tikzcd}$$

Shop V1.0 - **Product**



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

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)

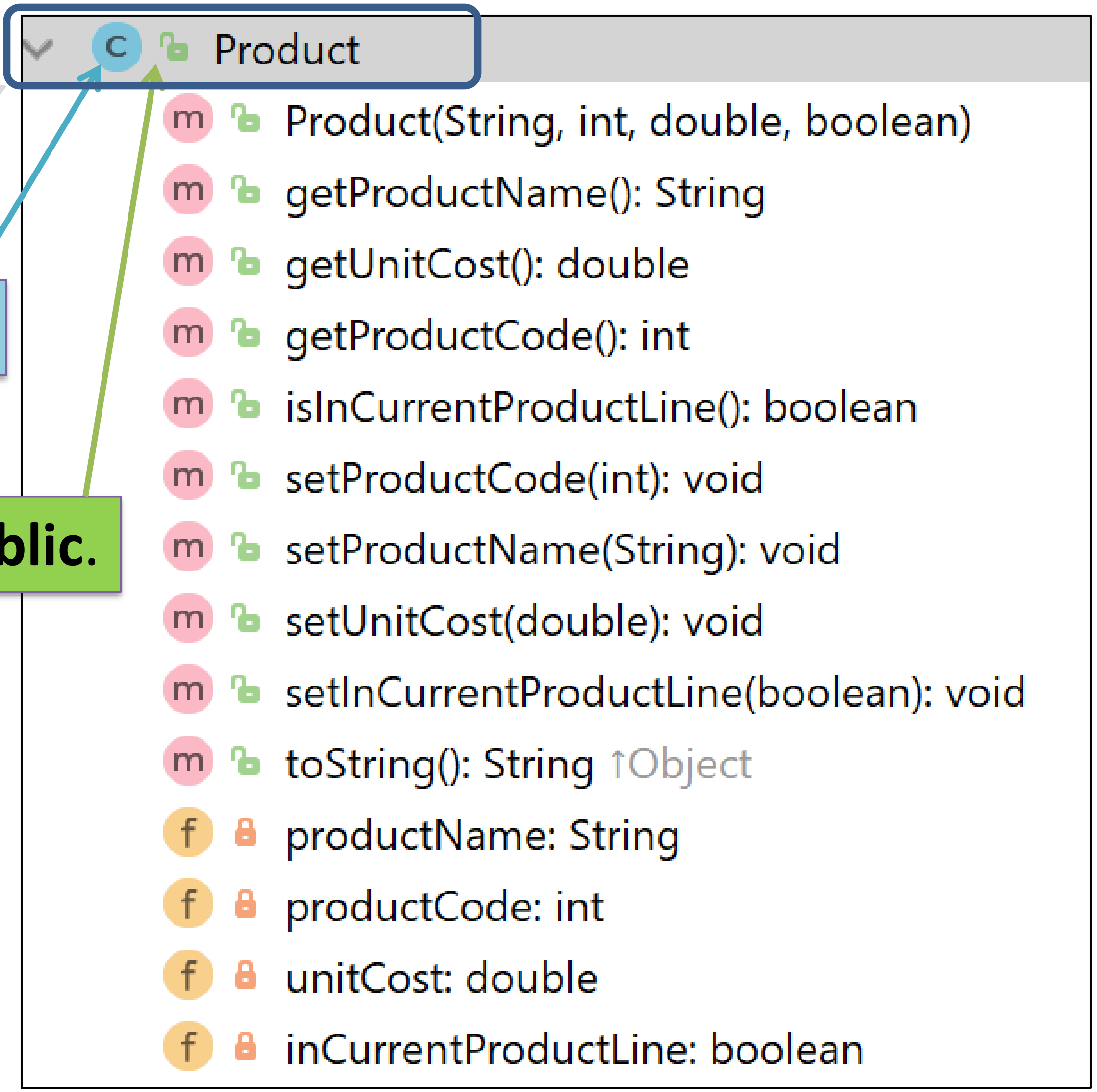
- Driver is covered in the next lecture.

A **Product** Class…

Object Type/ **Class** Name
i.e. Product

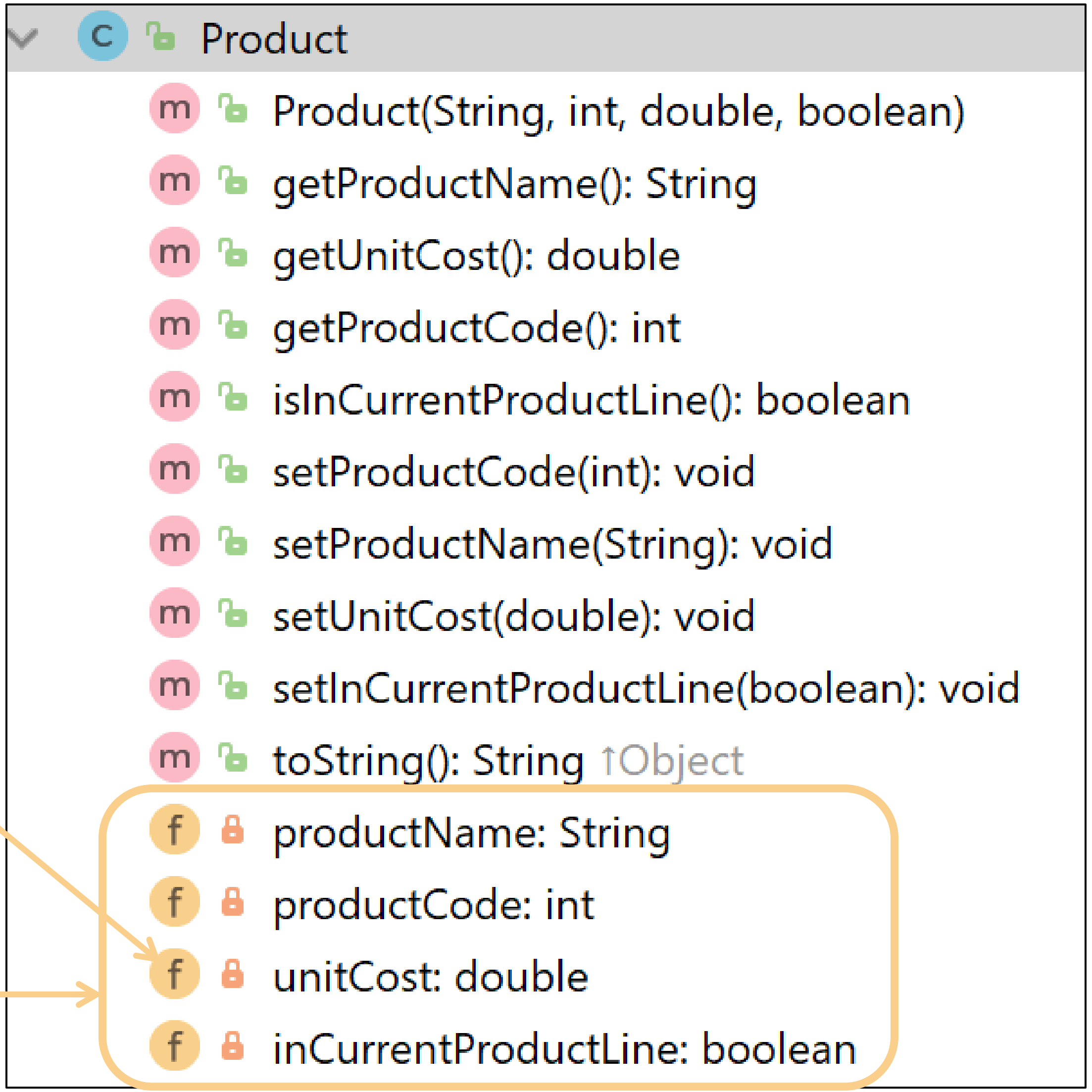
The **C** icon means it is a **Class**.

The open padlock means it is **public**.



```
Product
  Product(String, int, double, boolean)
  getProductName(): String
  getUnitCost(): double
  getProductCode(): int
  isInCurrentProductLine(): boolean
  setProductCode(int): void
  setProductName(String): void
  setUnitCost(double): void
  setInCurrentProductLine(boolean): void
  toString(): String ↑Object
  productName: String
  productCode: int
  unitCost: double
  inCurrentProductLine: boolean
```

A Product Class...**fields**

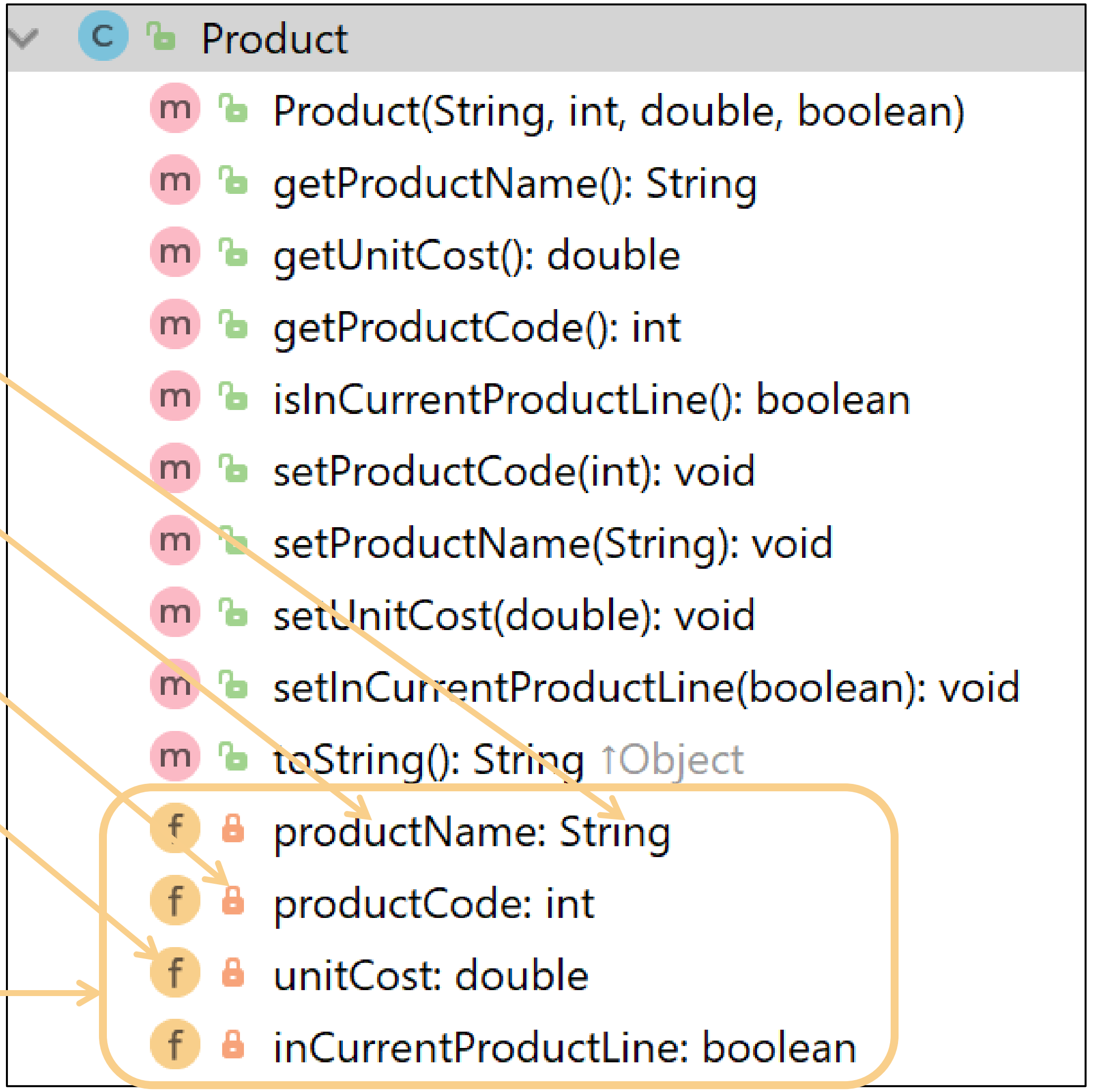


```
Product
  Product(String, int, double, boolean)
  getProductName(): String
  getUnitCost(): double
  getProductCode(): int
  isInCurrentProductLine(): boolean
  setProductCode(int): void
  setProductName(String): void
  setUnitCost(double): void
  setInCurrentProductLine(boolean): void
  toString(): String ↑Object
  productName: String
  productCode: int
  unitCost: double
  inCurrentProductLine: boolean
```

The **f** icon means it is a **field**.

Fields
i.e. the **attributes / properties** of the class

A Product Class...**fields**



Product
- Product(String, int, double, boolean)
- getProductName(): String
- getUnitCost(): double
- getProductCode(): int
- isInCurrentProductLine(): boolean
- setProductCode(int): void
- setProductName(String): void
- setUnitCost(double): void
- setInCurrentProductLine(boolean): void
- toString(): String ↑Object
- productName: String
- productCode: int
- unitCost: double
- inCurrentProductLine: boolean

field **type**

field **name**

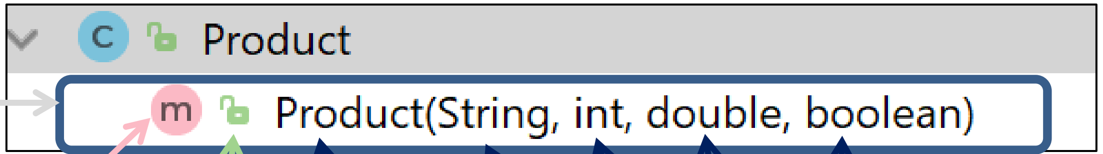
The closed padlock means it is **private**.

The **f** icon means it is a **field**.

Fields
i.e. the **attributes / properties** of the class

A **Product** Class… **constructor**

Constructor
i.e. for building objects.



```
Product
    Product(String, int, double, boolean)
```

The **m** icon means it is a **method**.

The open padlock means it is **public**.

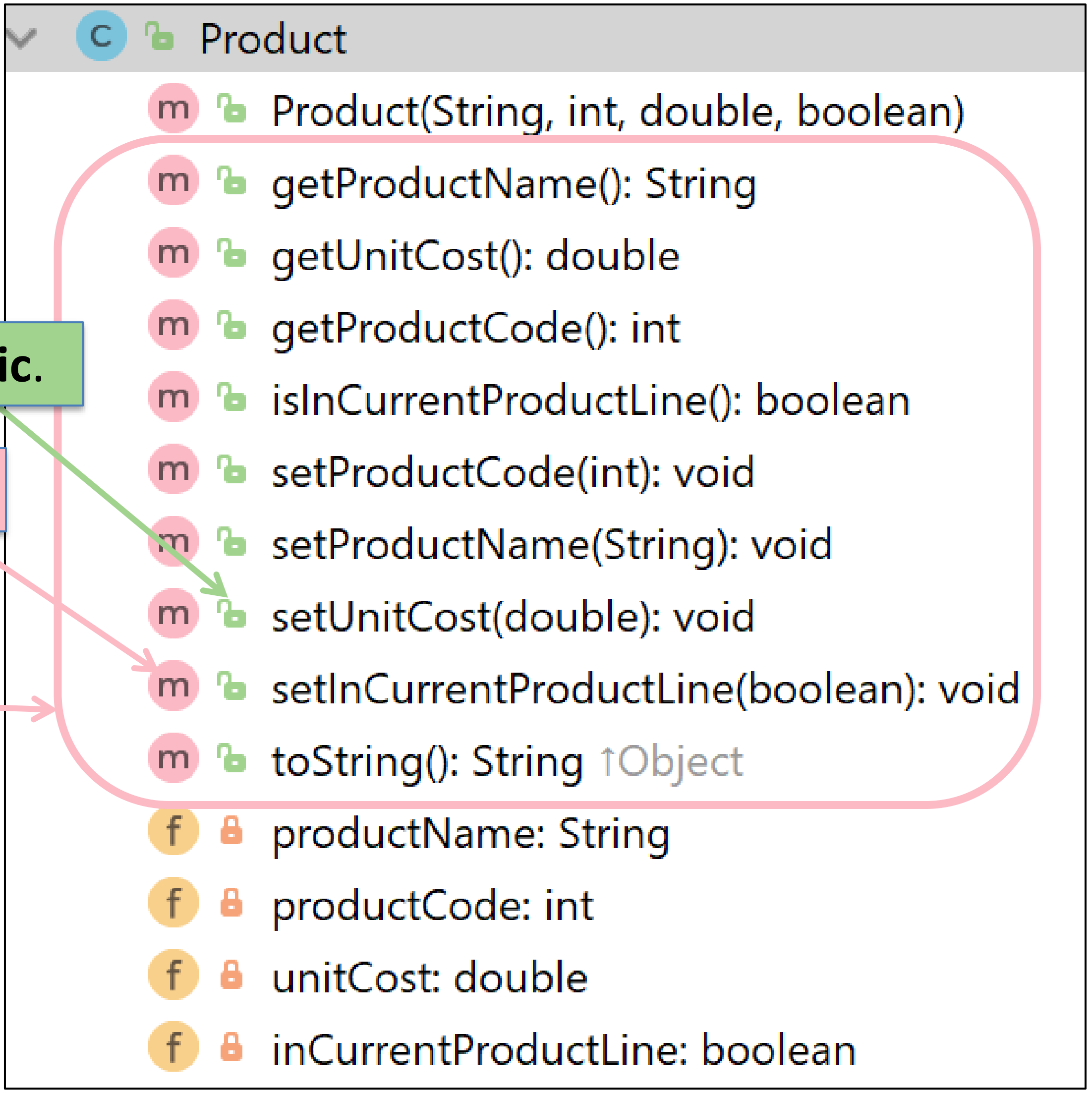
Constructors have same name as the class

Four **parameters**;
one for each field.

A **Product** Class… **fields** and **constructor**

```
public class Product {

    private String productName;
    private int productCode;
    private double unitCost;
    private boolean inCurrentProductLine;

    public Product (String productName, int productCode,
                    double unitCost, boolean inCurrentProductLine){

        this.productName = productName;
        this.productCode = productCode;
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }
```

A Product Class… **methods**



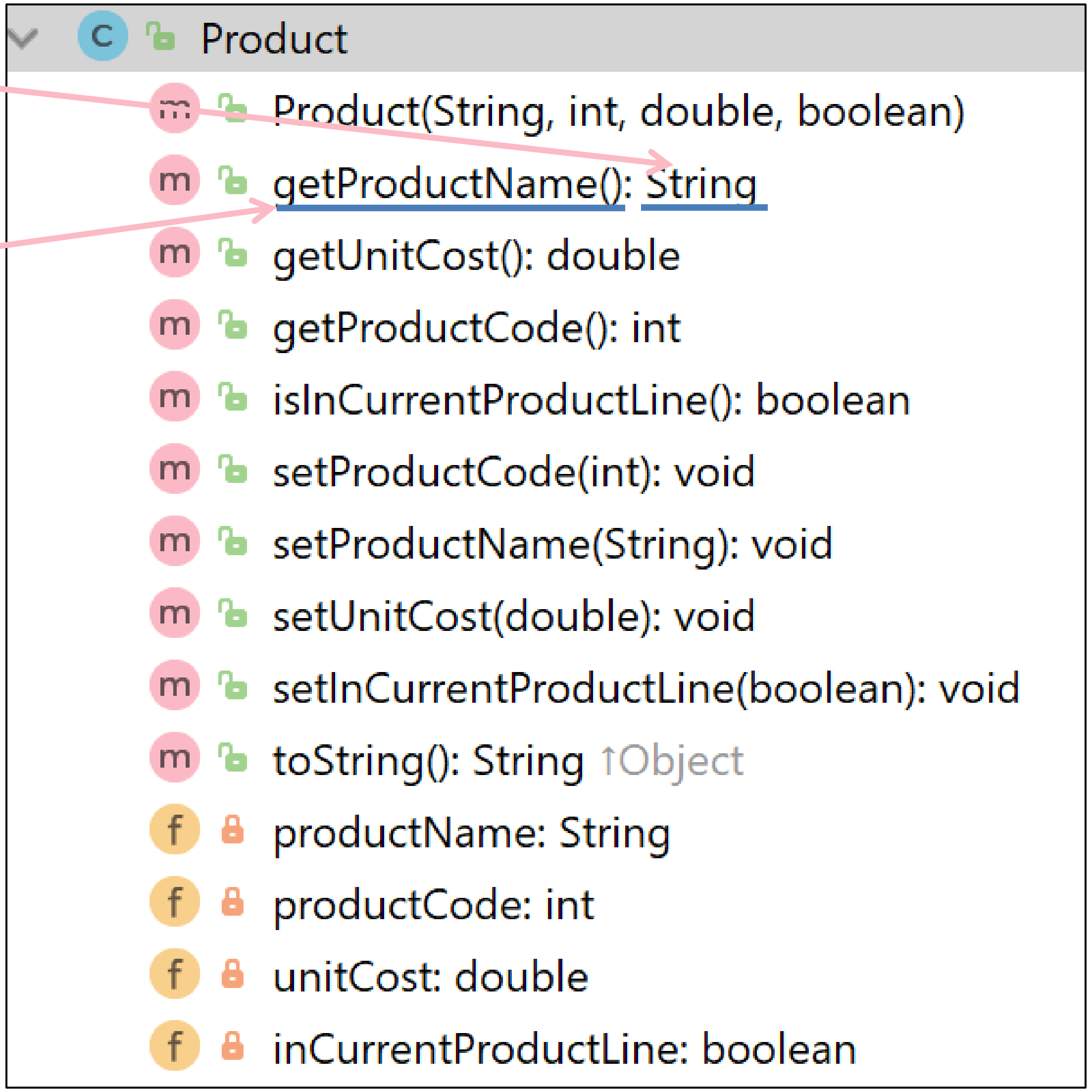
```
Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean
```

The open padlock means it is **public**.

The **m** icon means it is a **method**.

Methods
i.e. the **behaviours** of the class

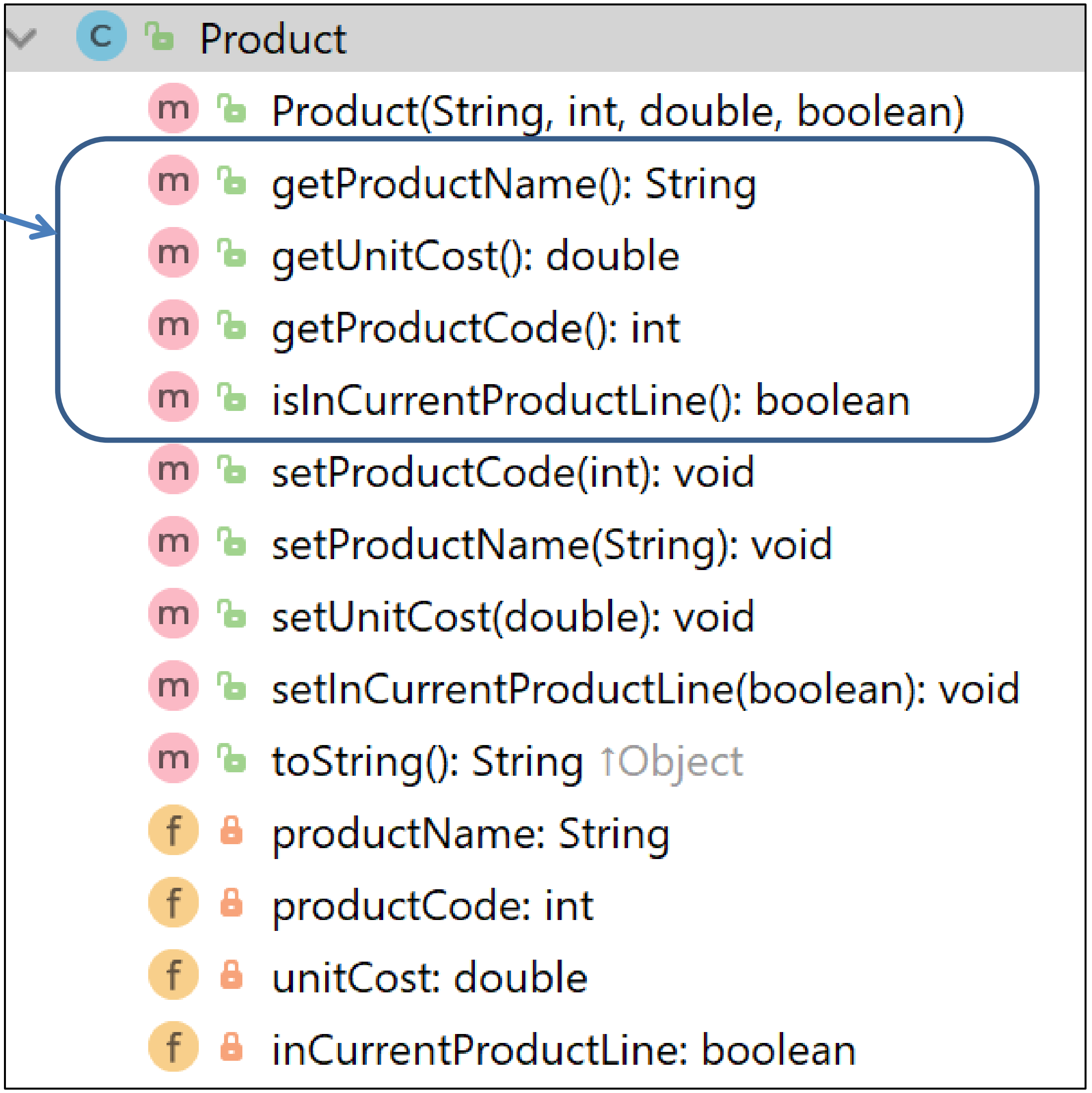
A Product Class... **methods**



Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean

A Product Class... **getters**

getters



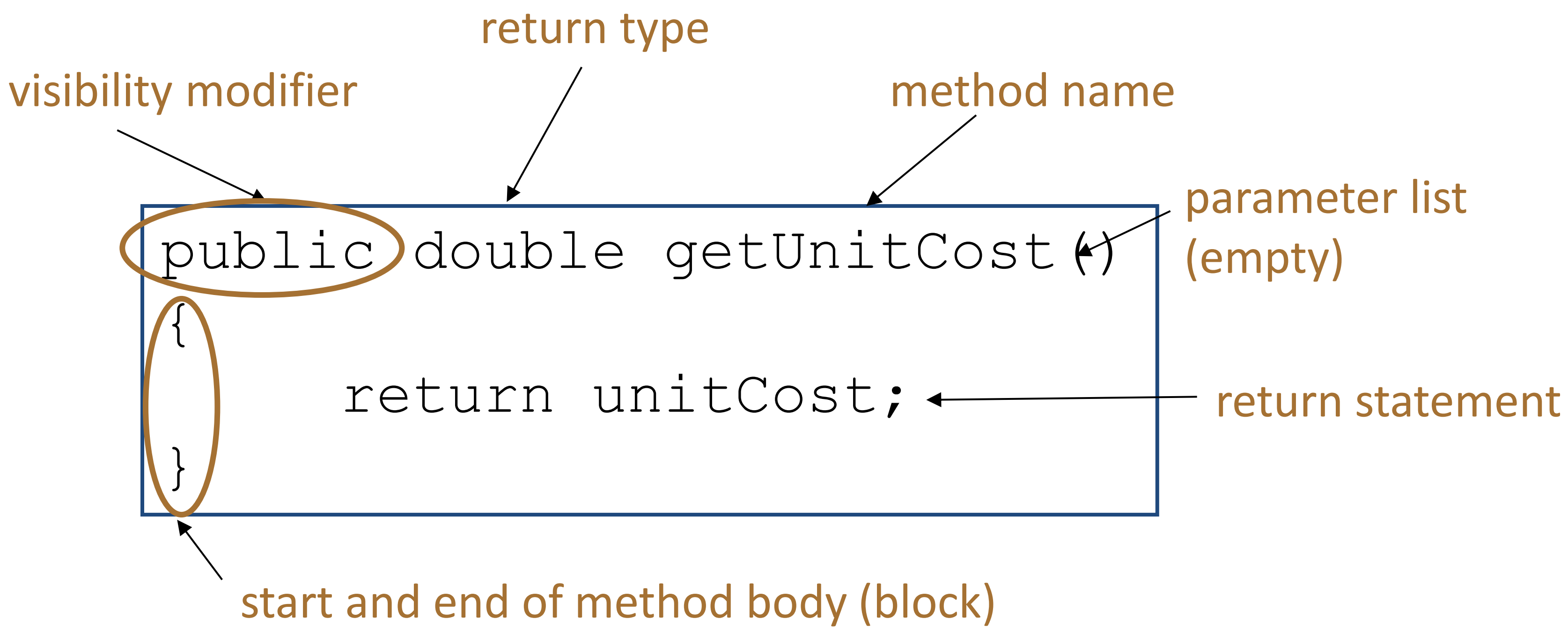
```
Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean
```

Getters (Accessor Methods)

- **Accessor** methods
  - return information about the **state** of an object
    - i.e. **the values stored in the fields**.

- A 'getter' method
  - is a specific type of **accessor** method and typically:
    - **contains a return statement**
      (as the last executable statement in the method).
    - defines a **return type**.
    - **does NOT change the object state**.

Getters

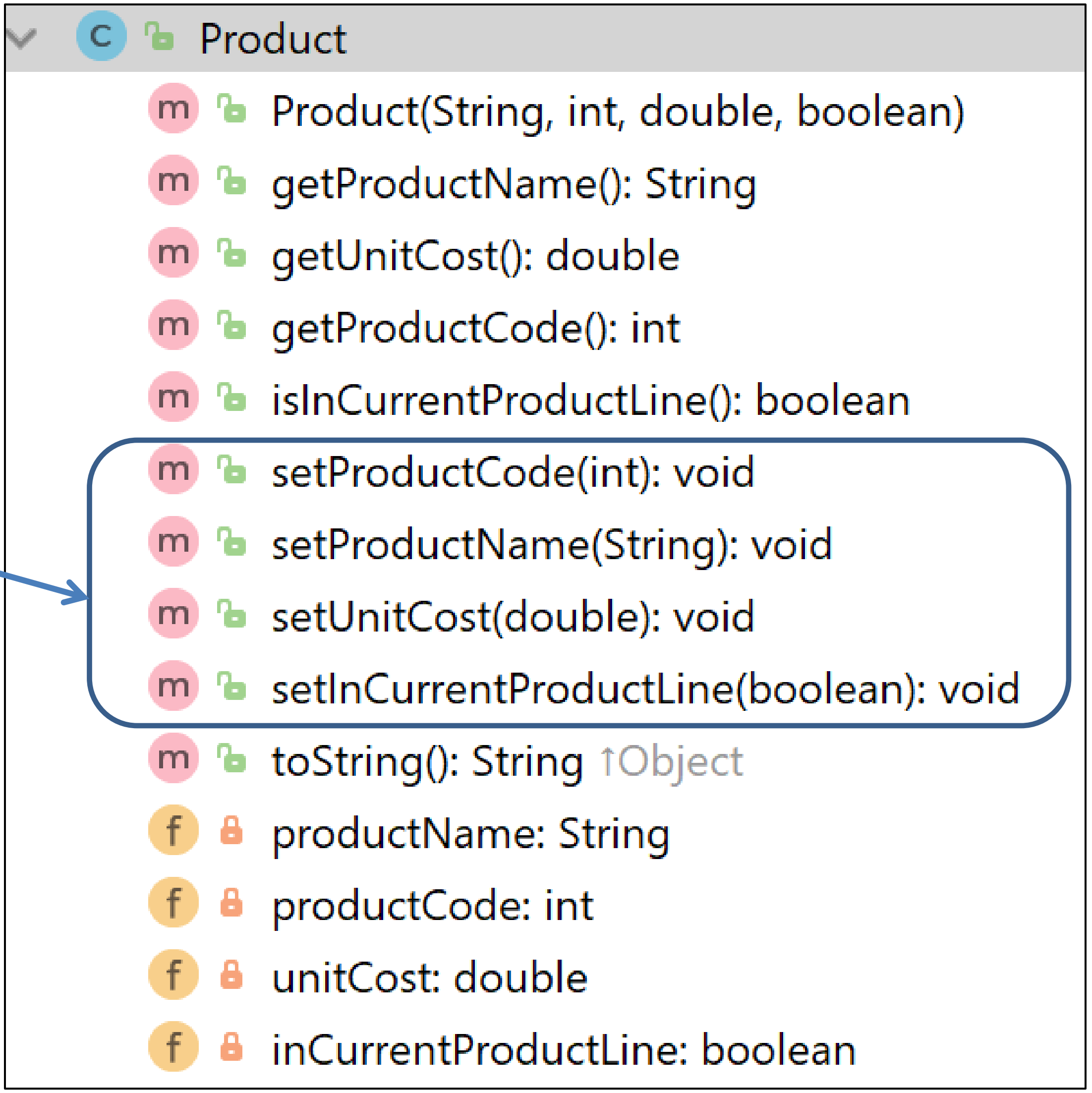

```
public double getUnitCost()
{
        return unitCost;
}
```

A Product Class…**get**ters

```
public String getProductName(){
    return productName;
}

public double getUnitCost(){
    return unitCost;
}

public int getProductCode() {
    return productCode;
}

public boolean isInCurrentProductLine() {
    return inCurrentProductLine;
}
```

A Product Class...**set**ters

setters



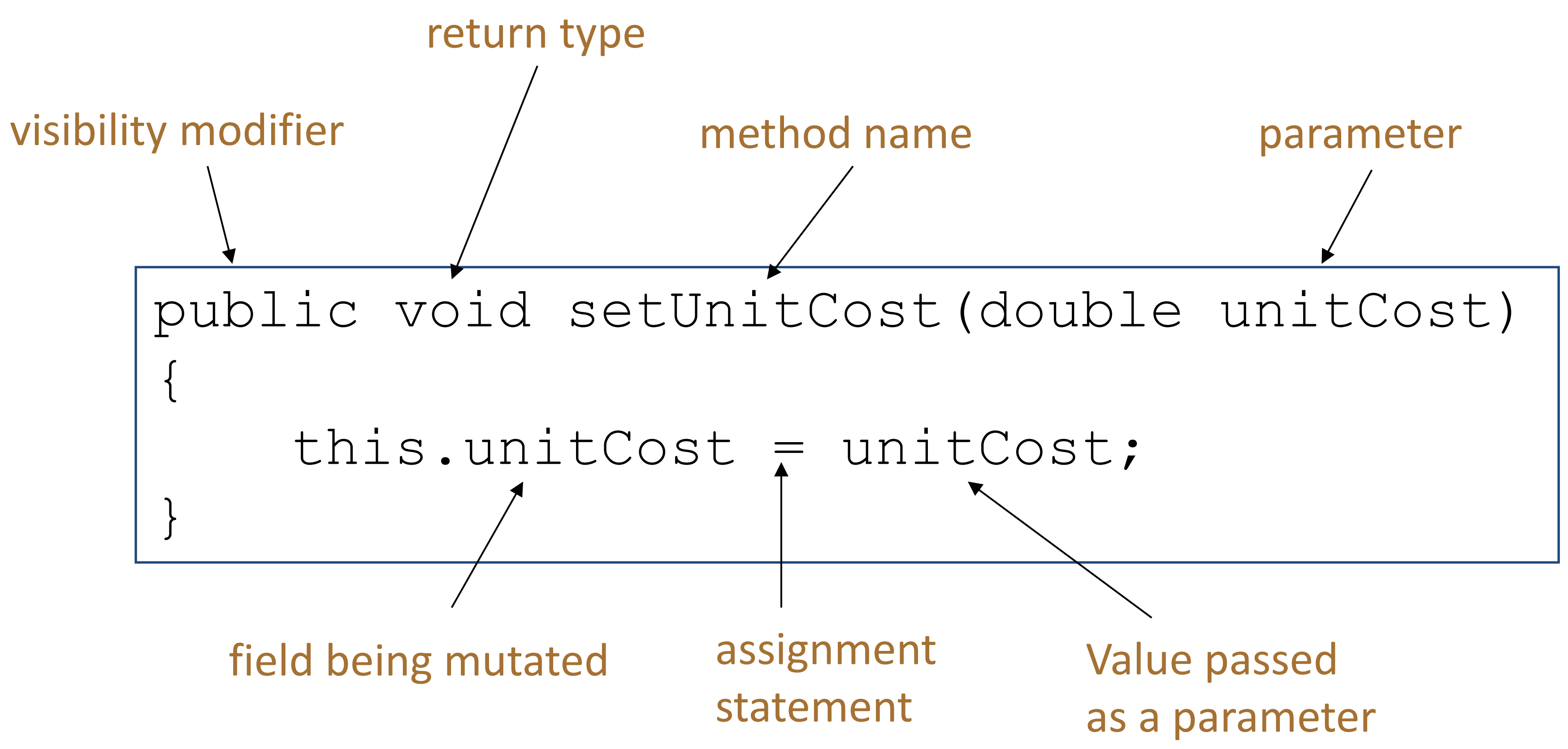
```
Product
  Product(String, int, double, boolean)
  getProductName(): String
  getUnitCost(): double
  getProductCode(): int
  isInCurrentProductLine(): boolean
  setProductCode(int): void
  setProductName(String): void
  setUnitCost(double): void
  setInCurrentProductLine(boolean): void
  toString(): String ↑Object
  productName: String
  productCode: int
  unitCost: double
  inCurrentProductLine: boolean
```

Setters (Mutator methods)

- **Mutator** methods
  - change (i.e. mutate!) an object’s state.

- A ‘setter’ method
  - is a specific type of **mutator** method and typically:
    - contains an **assignment statement**
    - takes in a **parameter**
    - **changes the object state**.

Setters



```
public void setUnitCost(double unitCost)
{
    this.unitCost = unitCost;
}
```

A Product Class…setters

```
public void setProductCode(int productCode) {
    this.productCode = productCode;
}

public void setProductName(String productName) {
    this.productName = productName;
}

public void setUnitCost(double unitCost) {
    this.unitCost = unitCost;
}

public void setInCurrentProductLine(boolean inCurrentProductLine) {
    this.inCurrentProductLine = inCurrentProductLine;
}
```

Getters/Setters

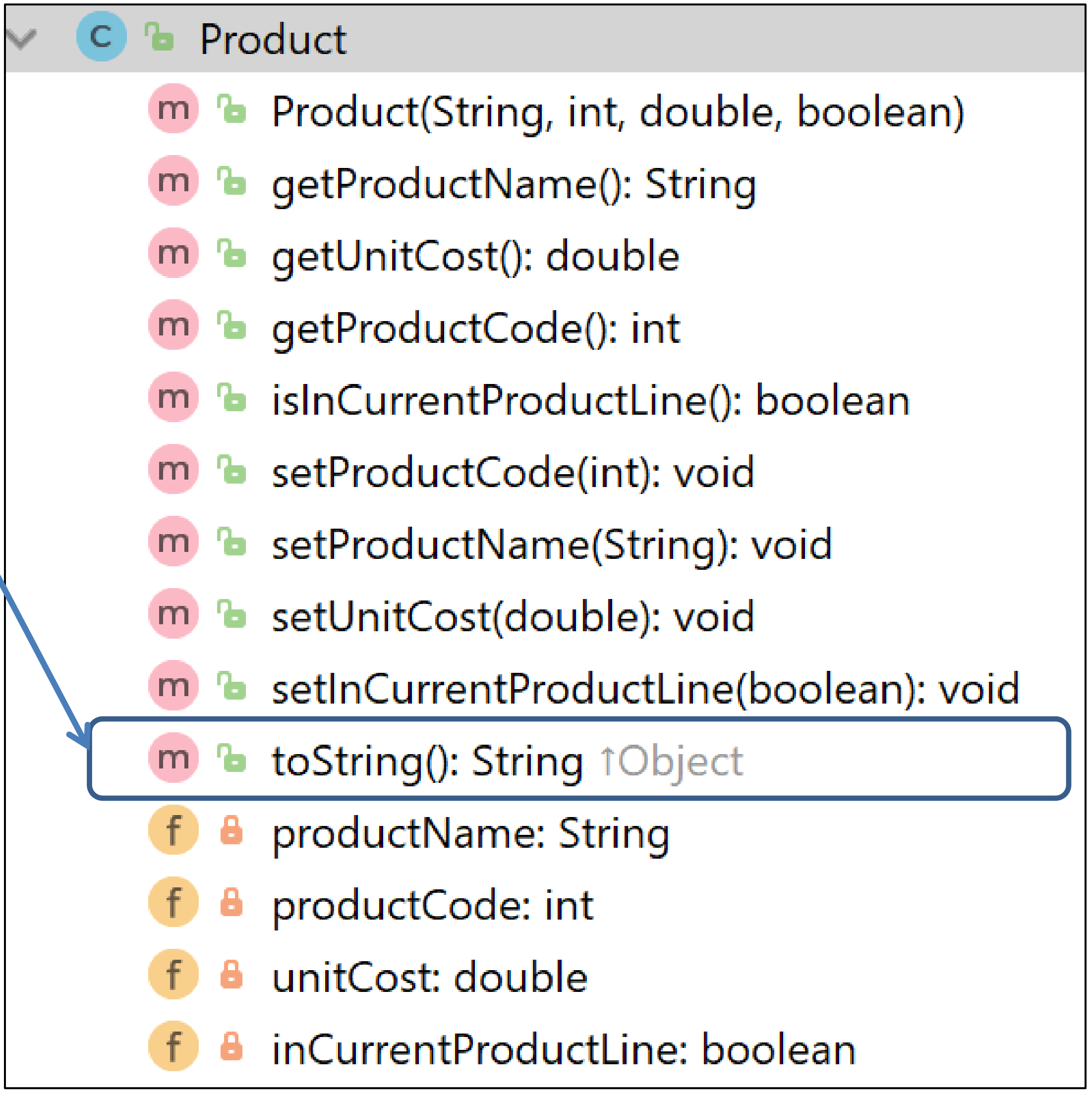
For each instance field in a class, you are normally asked to write:

- A **get**ter
  - Return statement
- A **set**ter
  - Assignment statement

A Product Class…**toString**

toString():

Builds and returns a String containing a user-friendly representation of the object state.



Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean

A Product Class…

```
public String toString()
{
    return "Product description: " + productName
         + ", product code: " + productCode
         + ", unit cost: " + unitCost
         + ", currently in product line: " + inCurrentProductLine;
}
```

Sample Console Output if we printed a Product Object:

```
Product description: 24 Inch TV, product code: 23432, unit cost: 399.99, currently in product line: true
```

toString()

- This is a useful method and you will write a **toString**() method for most of your classes.
- **When you print an object, Java automatically calls the toString() method**

  e.g.

```
Product product = new Product();

//both of these lines of code do the same thing
System.out.println(product);
System.out.println(product.toString());
```

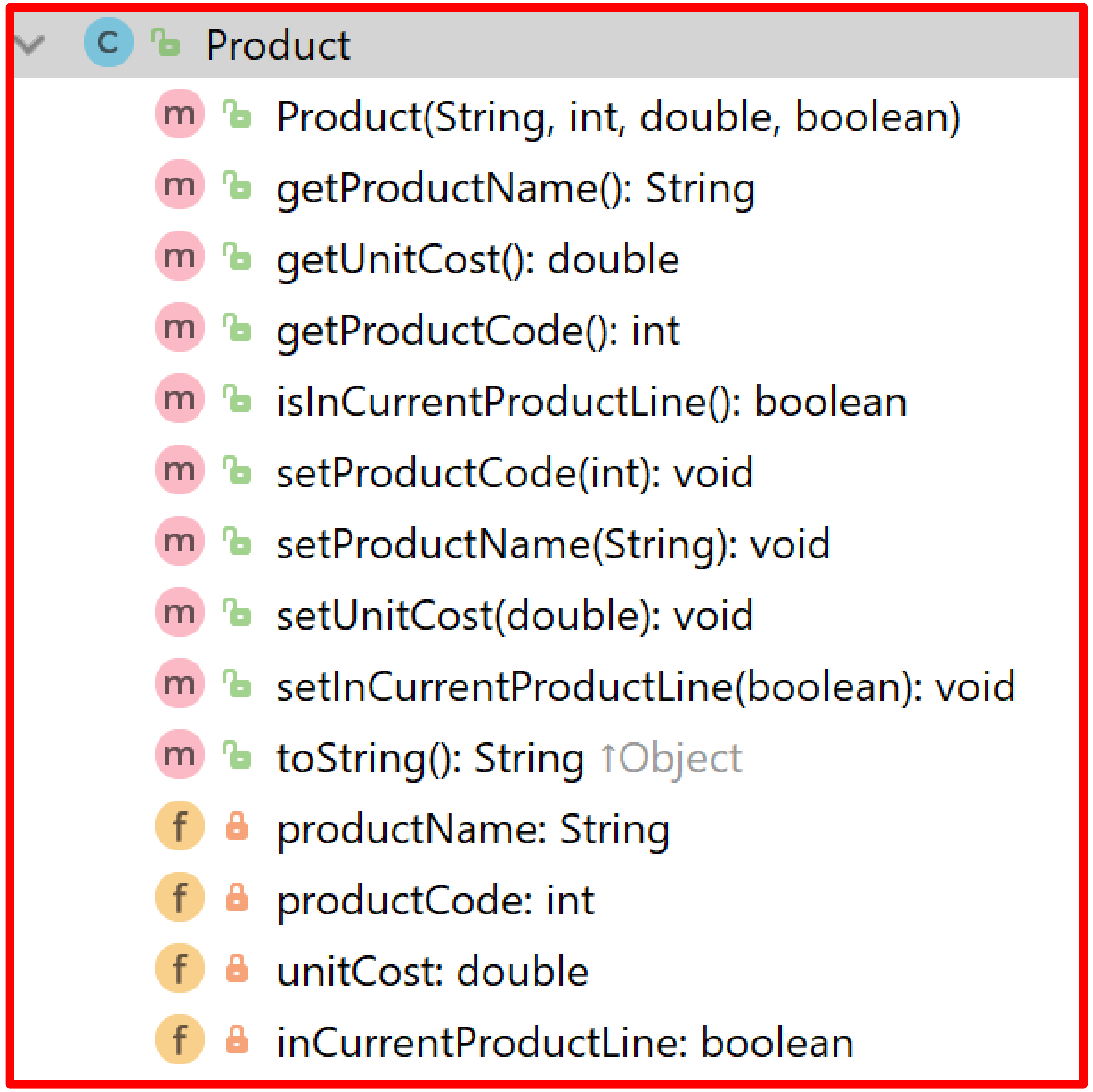
Encapsulation in Java – **steps 1-3**

Encapsulation Step	Approach in Java
1. **Wrap** the data (fields) and code acting on the data (methods) together as single unit.	`public class ClassName {  Fields  Constructors  Methods }`
2. **Hide** the fields from other classes.	**Declare the fields of a class as <u>private</u>.**
3. **Access** the fields only through the methods of their current class.	**Provide <u>public</u> set**ter **and get**ter methods to modify and view the fields values.

http://www.tutorialspoint.com/java/java_encapsulation.htm

A Product Class... **An Encapsulated Class**

1. Product class **wraps** the data (fields) and code acting on the data (methods) together as **single unit.**

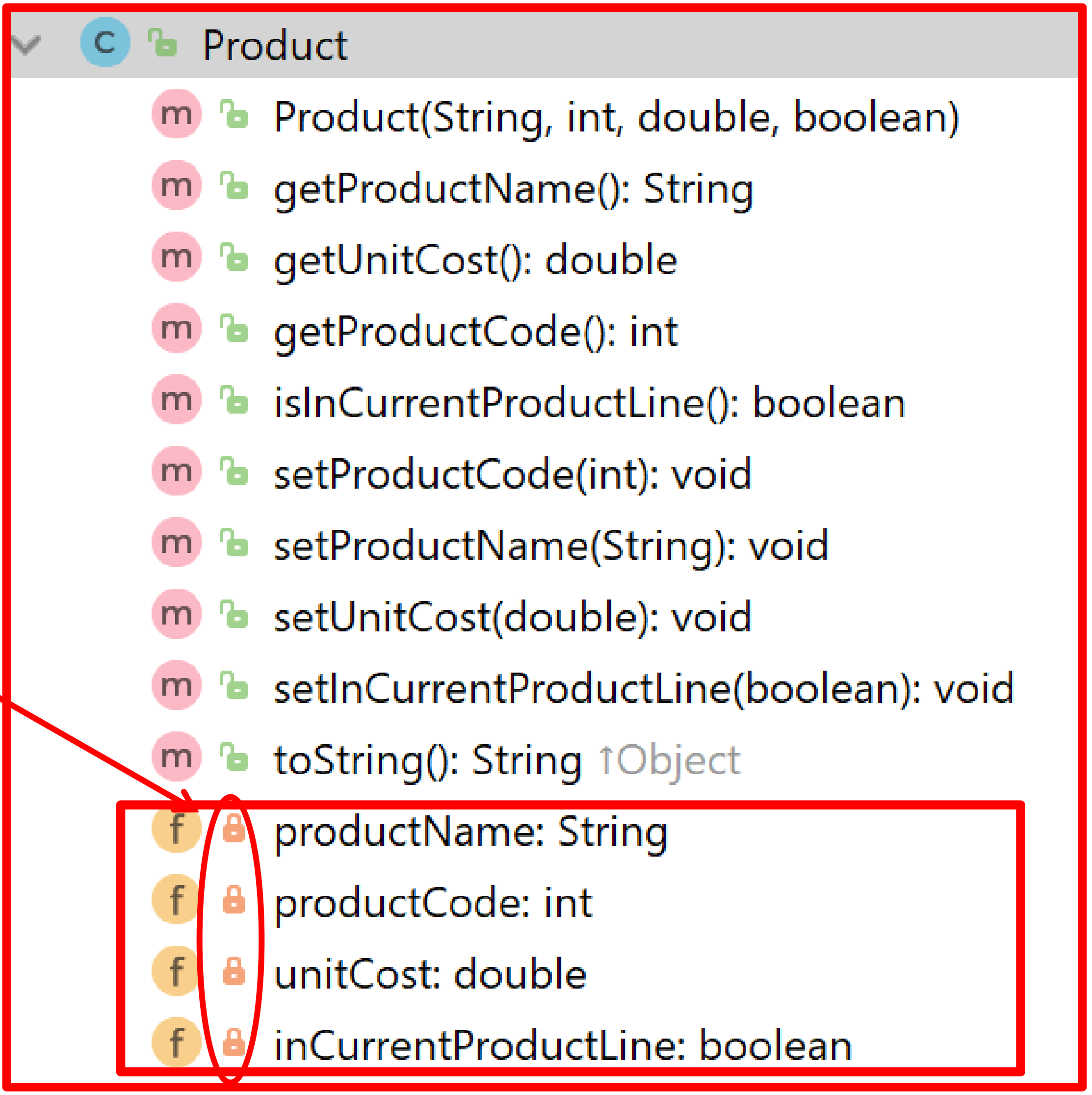


```
Product
  Product(String, int, double, boolean)
  getProductName(): String
  getUnitCost(): double
  getProductCode(): int
  isInCurrentProductLine(): boolean
  setProductCode(int): void
  setProductName(String): void
  setUnitCost(double): void
  setInCurrentProductLine(boolean): void
  toString(): String ↑Object
  productName: String
  productCode: int
  unitCost: double
  inCurrentProductLine: boolean
```

A Product Class… **An Encapsulated Class**

1. Product class **wraps** the data (fields) and code acting on the data (methods) together as **single unit.**

2. Fields are **hidden** from other classes.



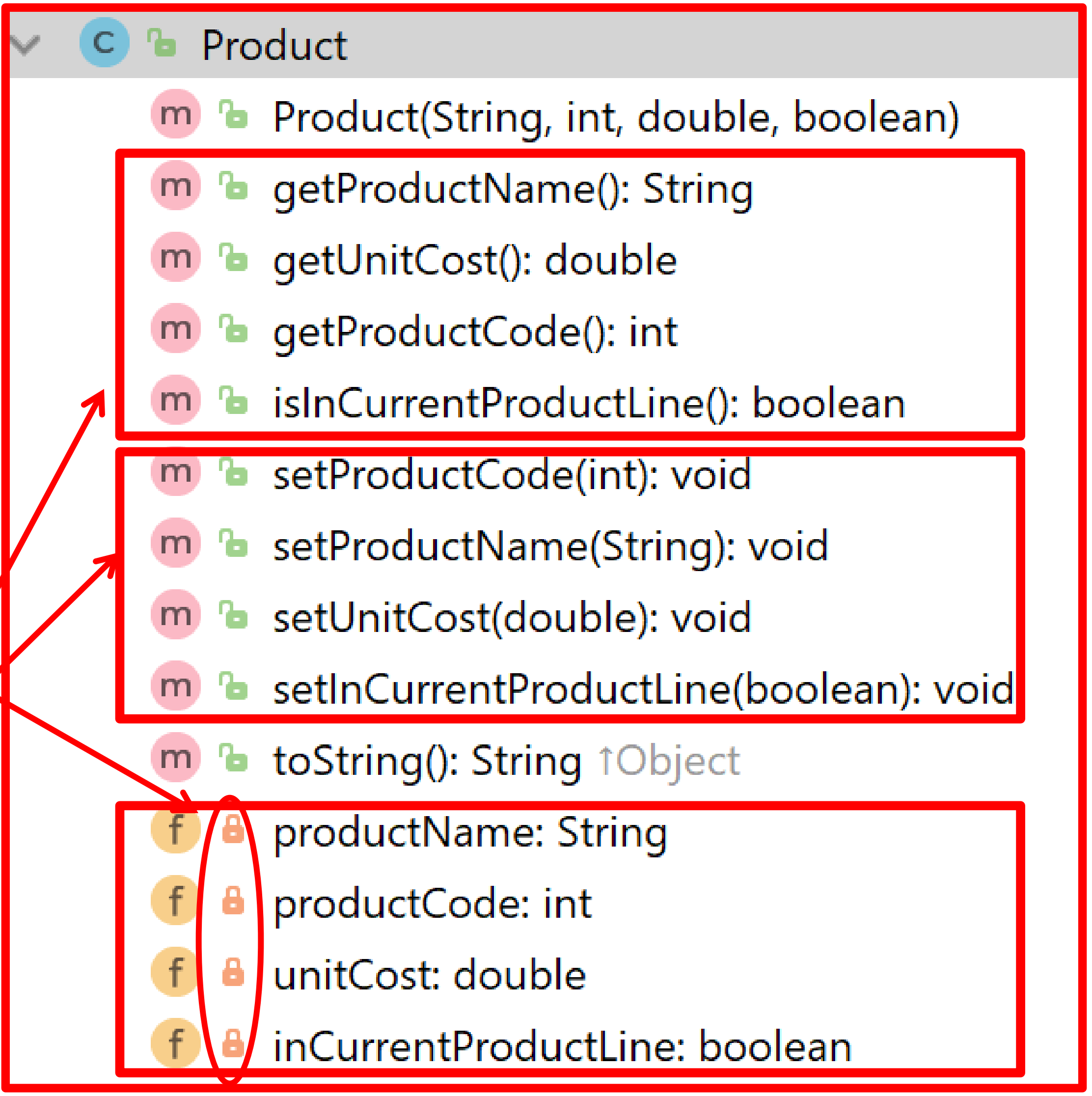
Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean

A Product Class... **An Encapsulated Class**

1. Product class **wraps** the data (fields) and code acting on the data (methods) together as **single unit.**

2. Fields are **hidden** from other classes.

3. **Access** the fields only through the methods of Product (e.g. **get**ter and **set**ter methods).

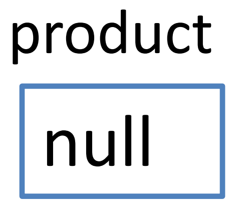

Product
Product(String, int, double, boolean)
getProductName(): String
getUnitCost(): double
getProductCode(): int
isInCurrentProductLine(): boolean
setProductCode(int): void
setProductName(String): void
setUnitCost(double): void
setInCurrentProductLine(boolean): void
toString(): String ↑Object
productName: String
productCode: int
unitCost: double
inCurrentProductLine: boolean

Using the Product Class

1

```
private Product product;
```

Declaring an object **product**, of type **Product**.



Using the Product Class

1

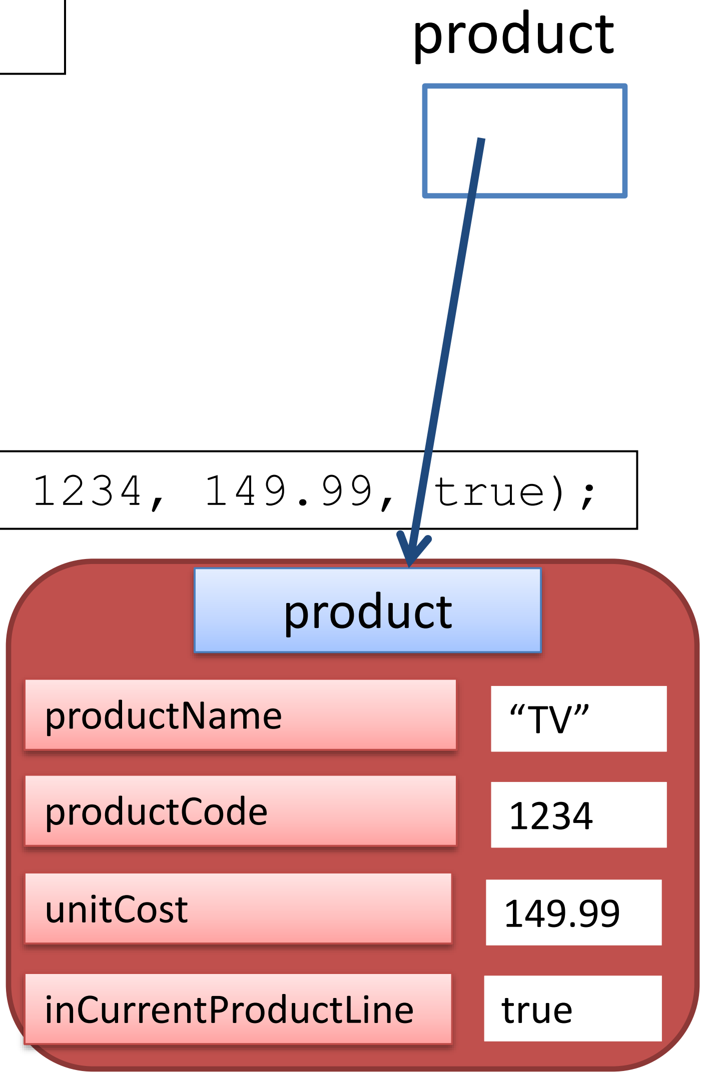
```
private Product product;
```

Declaring an object **product**, of type **Product**.

2

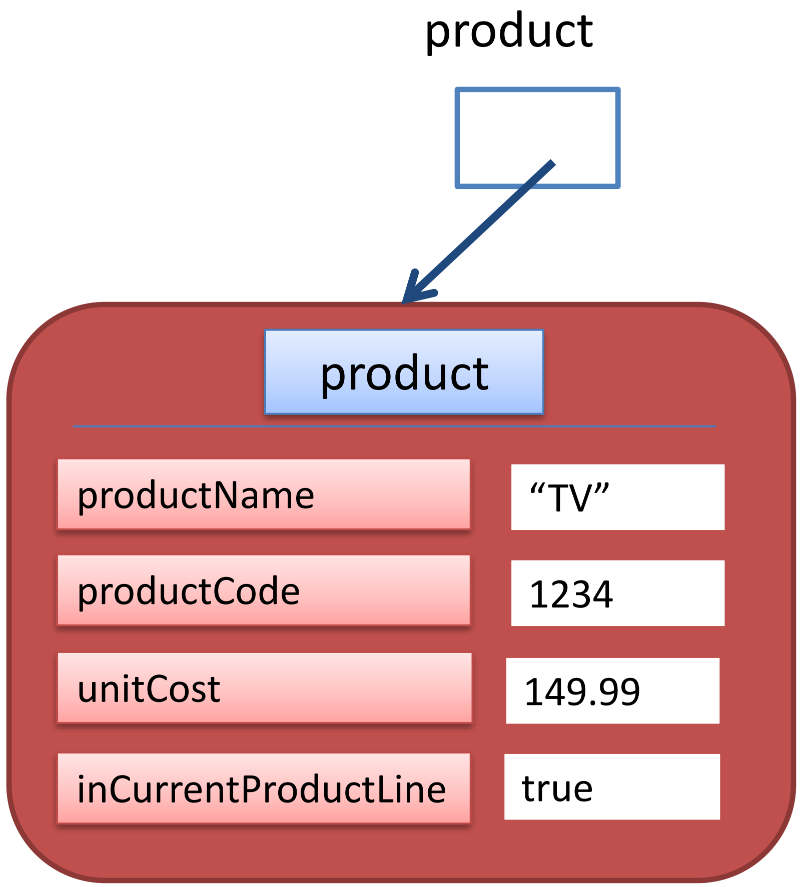
```
product = new Product("TV", 1234, 149.99, true);
```

Calls the **Product** *constructor* to build the **product** object in memory.



Multiple Product Objects

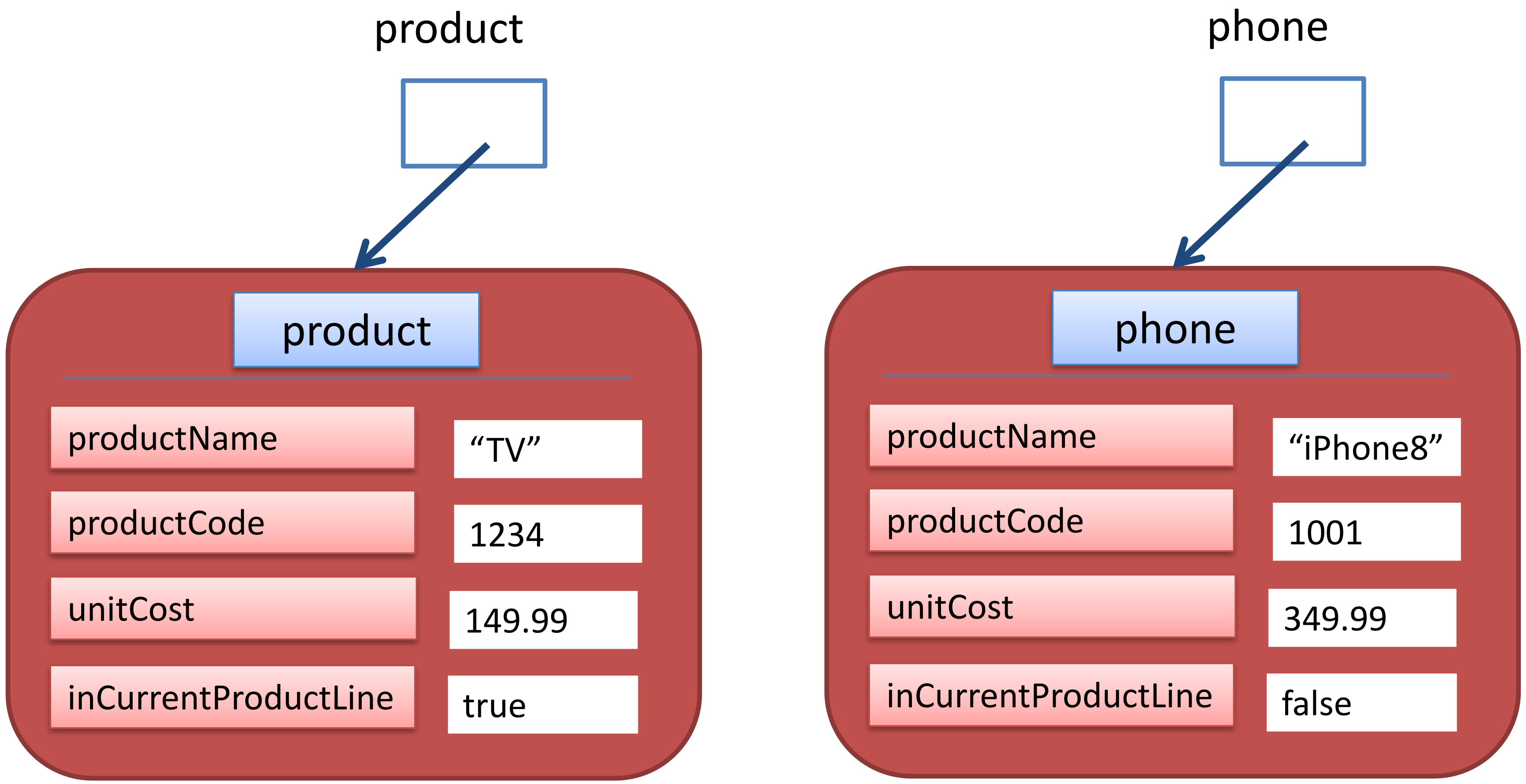
```
private Product product = new Product(“TV”, 1234, 149.99, true);
```



Multiple Product Objects

```
private Product product = new Product("TV", 1234, 149.99, true);
```

```
private Product phone = new Product("iPhone8", 1001, 349.99, false);
```



Questions?