

Recap of OO concepts

Objects, classes, methods and more.

Produced Ms. Mairead Meagher
by: Dr. Siobhán Drohan



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

Classes and Objects

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

The screenshot shows the Oracle Java Platform SE 7 API documentation for the `String` class. The browser address bar shows the URL `https://docs.oracle.com/javase/7/docs/api`. The navigation bar includes links for Overview, Package, Class (selected), Use, Tree, Deprecated, Index, and Help. Below the navigation bar, there are links for Prev Class, Next Class, Frames, No Frames, and All Classes. The main content area is divided into two sections: a left sidebar and a right main area.

Left Sidebar:

- java.lang
- Class String**
- java.lang.Object
- java.lang.String

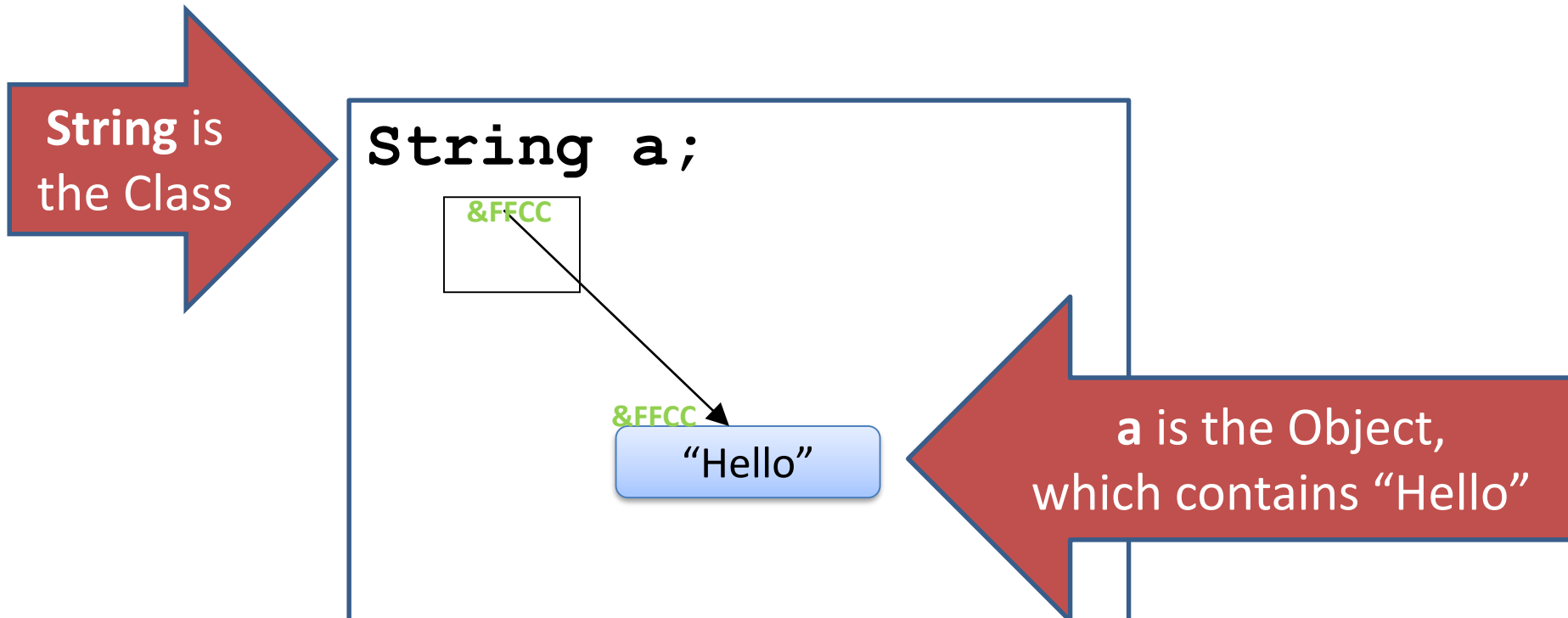
Right Main Area:

Method Summary

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

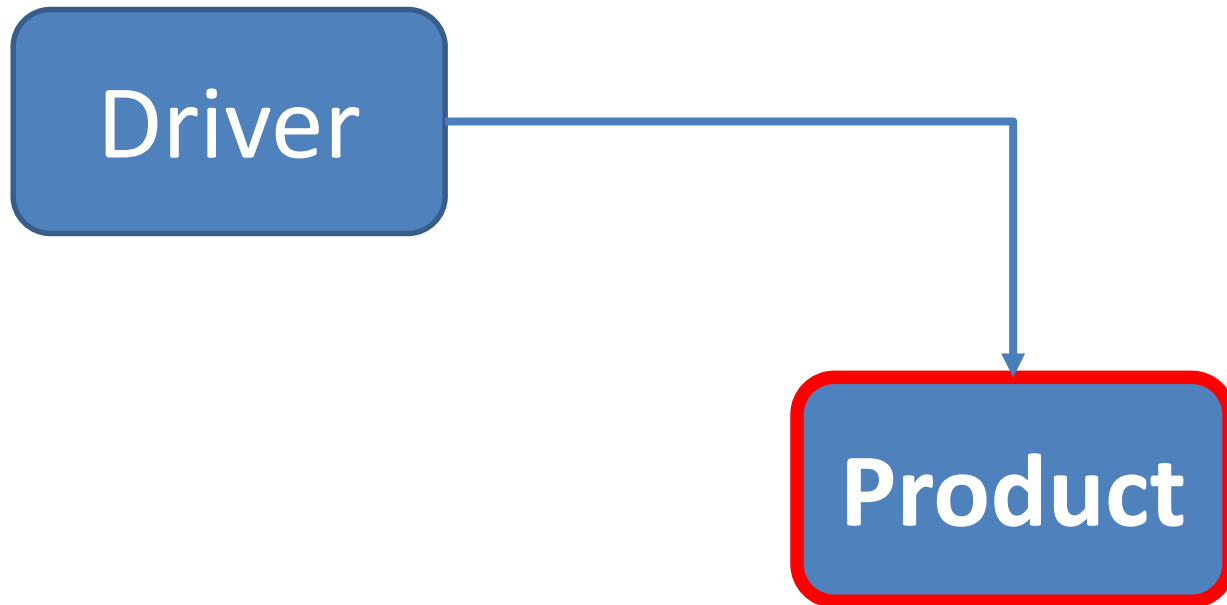
SHOP



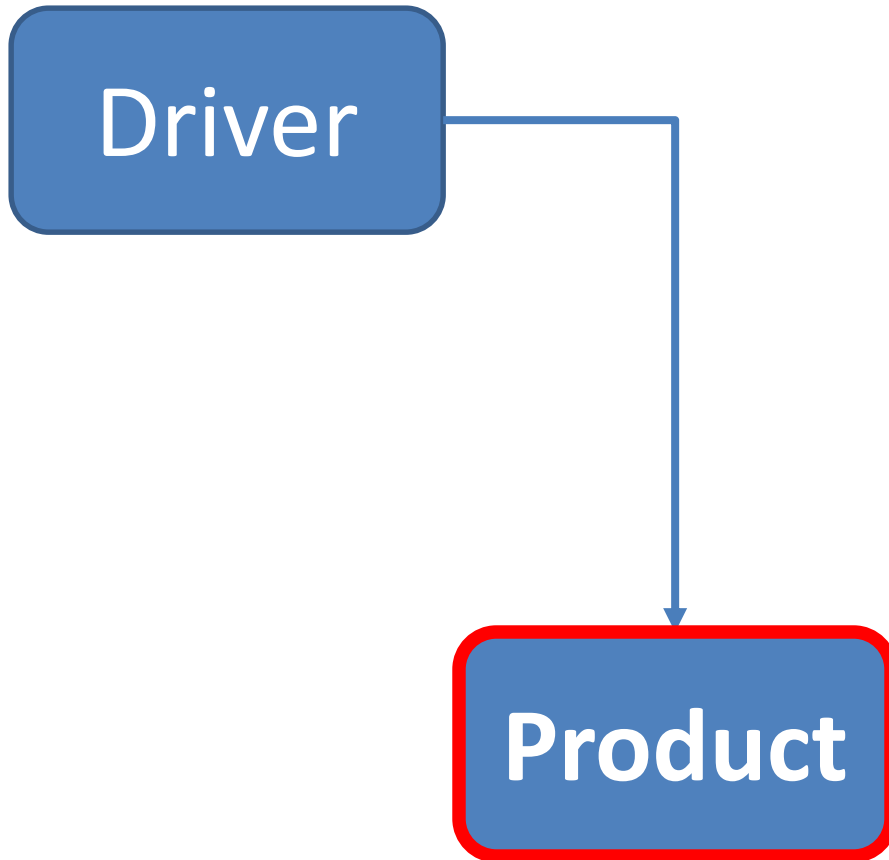
Shop V1.0 - Product



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



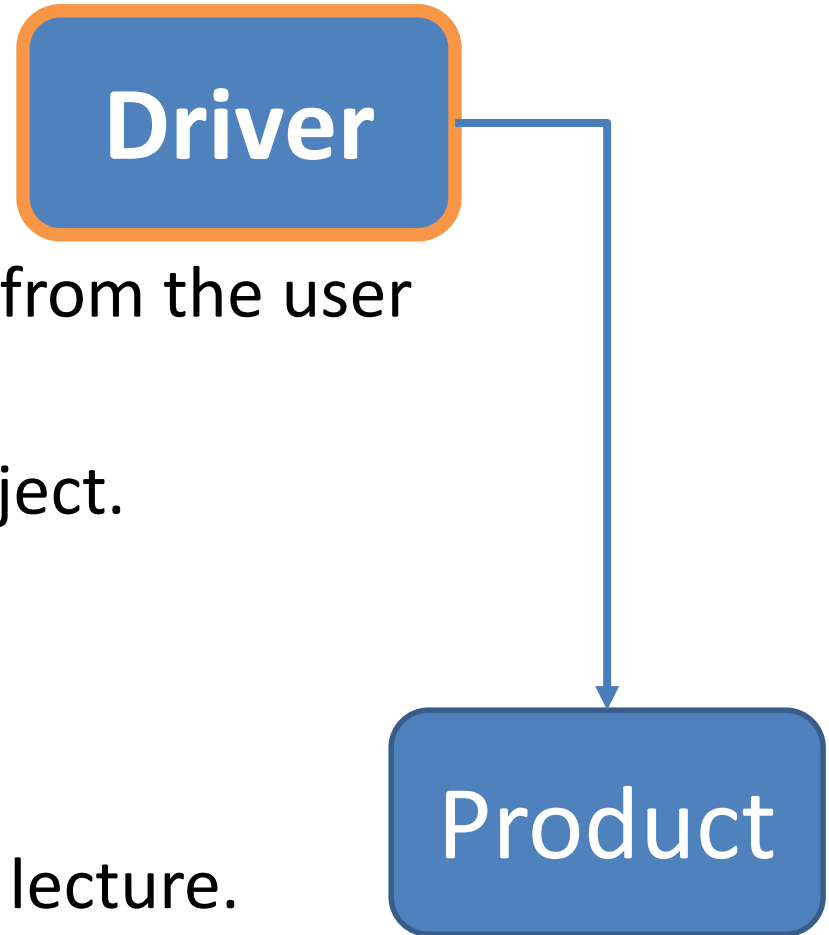
Shop V1.0 - Product



- 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

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

A Product Class...fields

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

Fields

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

Product

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

Fields (highlighted):

- f productName: String
- f productCode: int
- f unitCost: double
- f inCurrentProductLine: boolean

A Product Class...fields

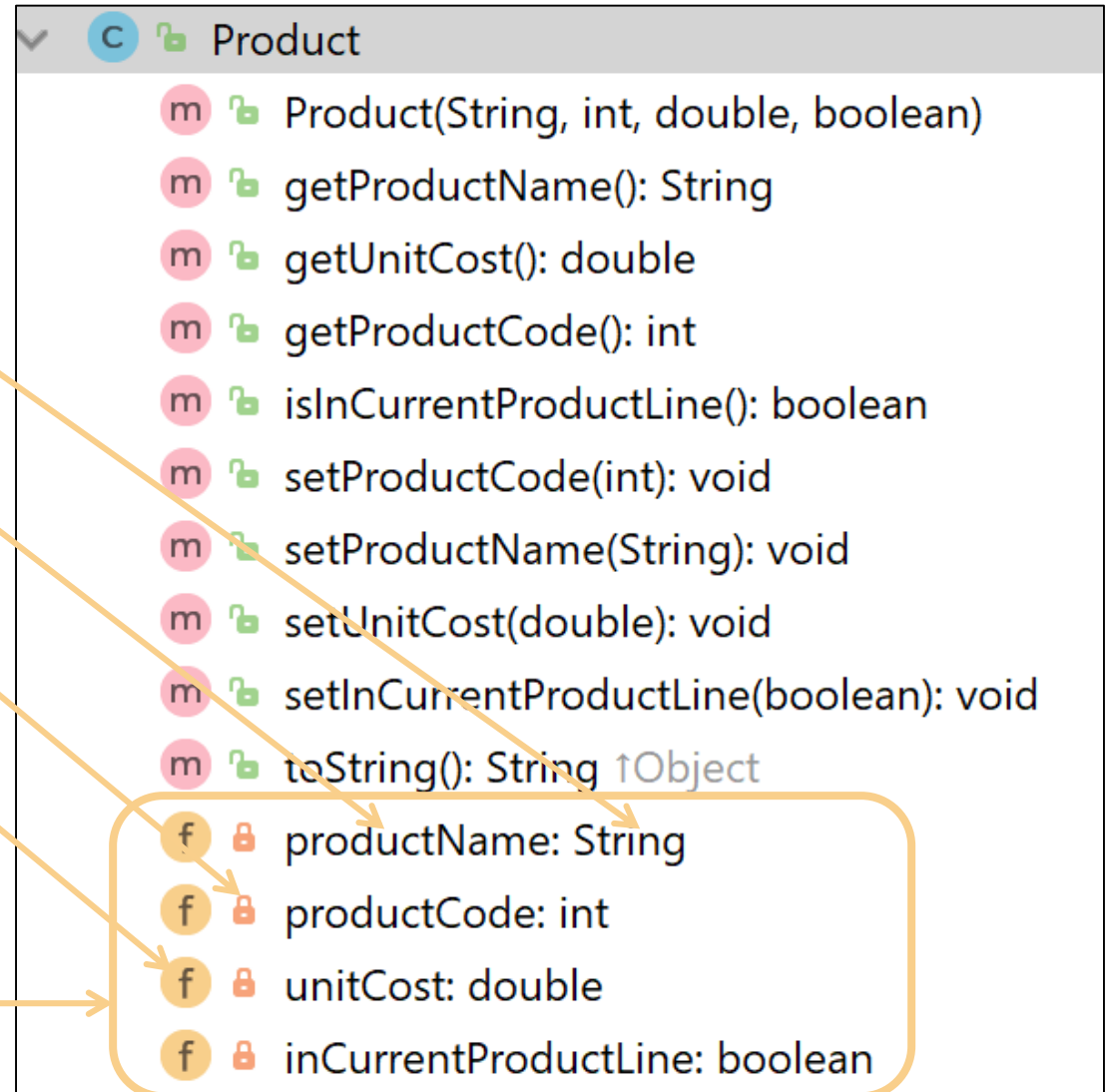
field **type**

field **name**

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

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

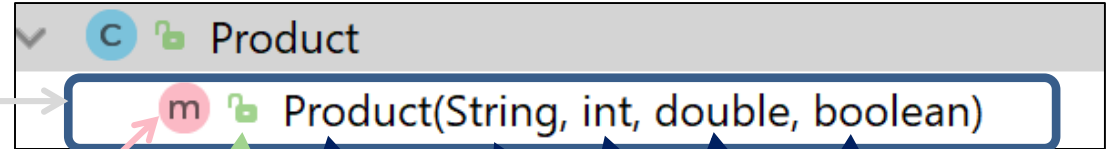
Fields
i.e. the **attributes / properties**
of the class



A Product Class... constructor

Constructor

i.e. for building objects.



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**

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

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

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

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

A Product Class... methods

Return type

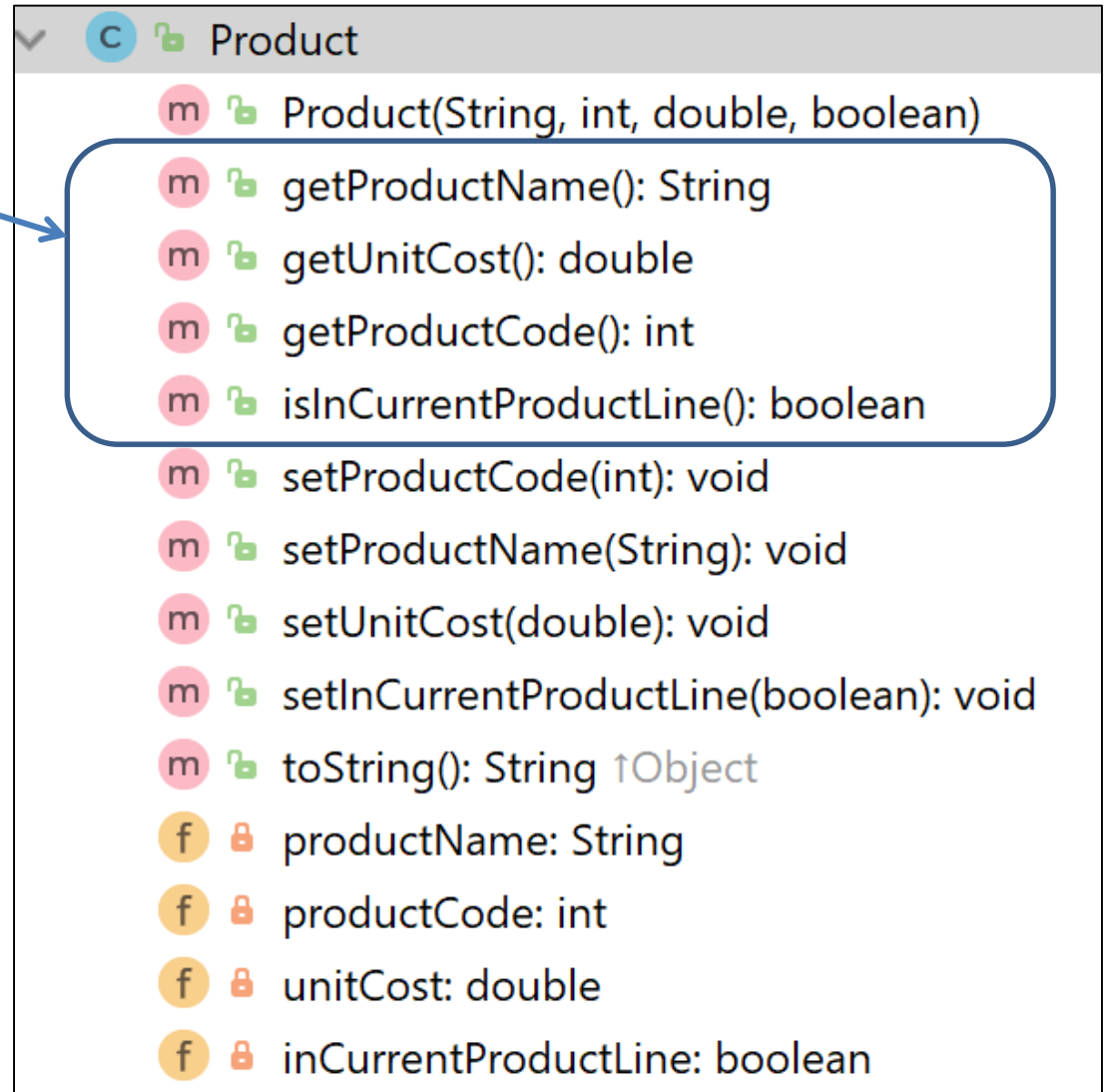
Method name

The screenshot shows the methods of a `Product` class in an IDE. The methods are listed with their return types and names. Annotations from the left point to specific parts of the `getProductName()` method signature.

- `m` `Product(String, int, double, boolean)`
- `m` `getProductName(): String` (Return type `String` is underlined)
- `m` `getUnitCost(): double`
- `m` `getProductCode(): int`
- `m` `isInCurrentProductLine(): boolean`
- `m` `setProductCode(int): void`
- `m` `setProductName(String): void`
- `m` `setUnitCost(double): void`
- `m` `setInCurrentProductLine(boolean): void`
- `m` `toString(): String` `↑Object`
- `f` `productName: String`
- `f` `productCode: int`
- `f` `unitCost: double`
- `f` `inCurrentProductLine: boolean`

A Product Class... **getters**

getters

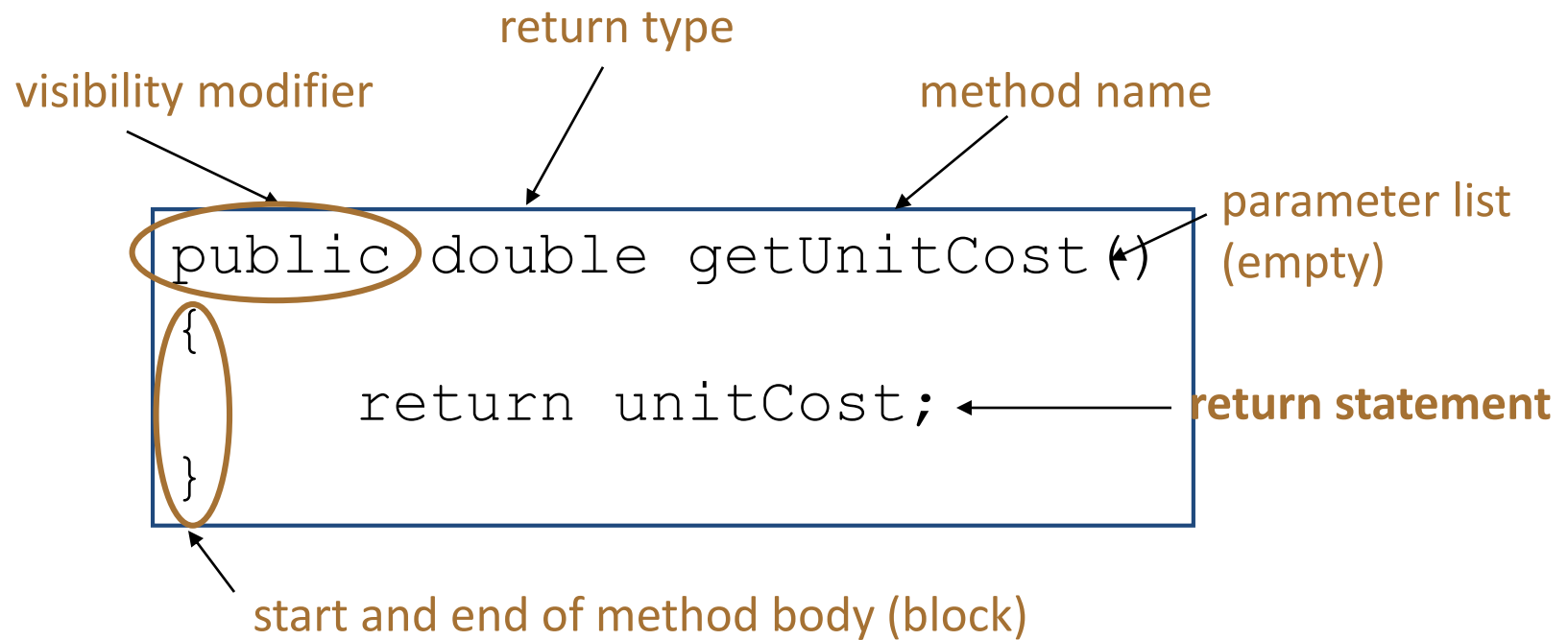


```
Product
  m Product(String, int, double, boolean)
  m getProductName(): String
  m getUnitCost(): double
  m getProductCode(): int
  m isInCurrentProductLine(): boolean
  m setProductCode(int): void
  m setProductName(String): void
  m setUnitCost(double): void
  m setInCurrentProductLine(boolean):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

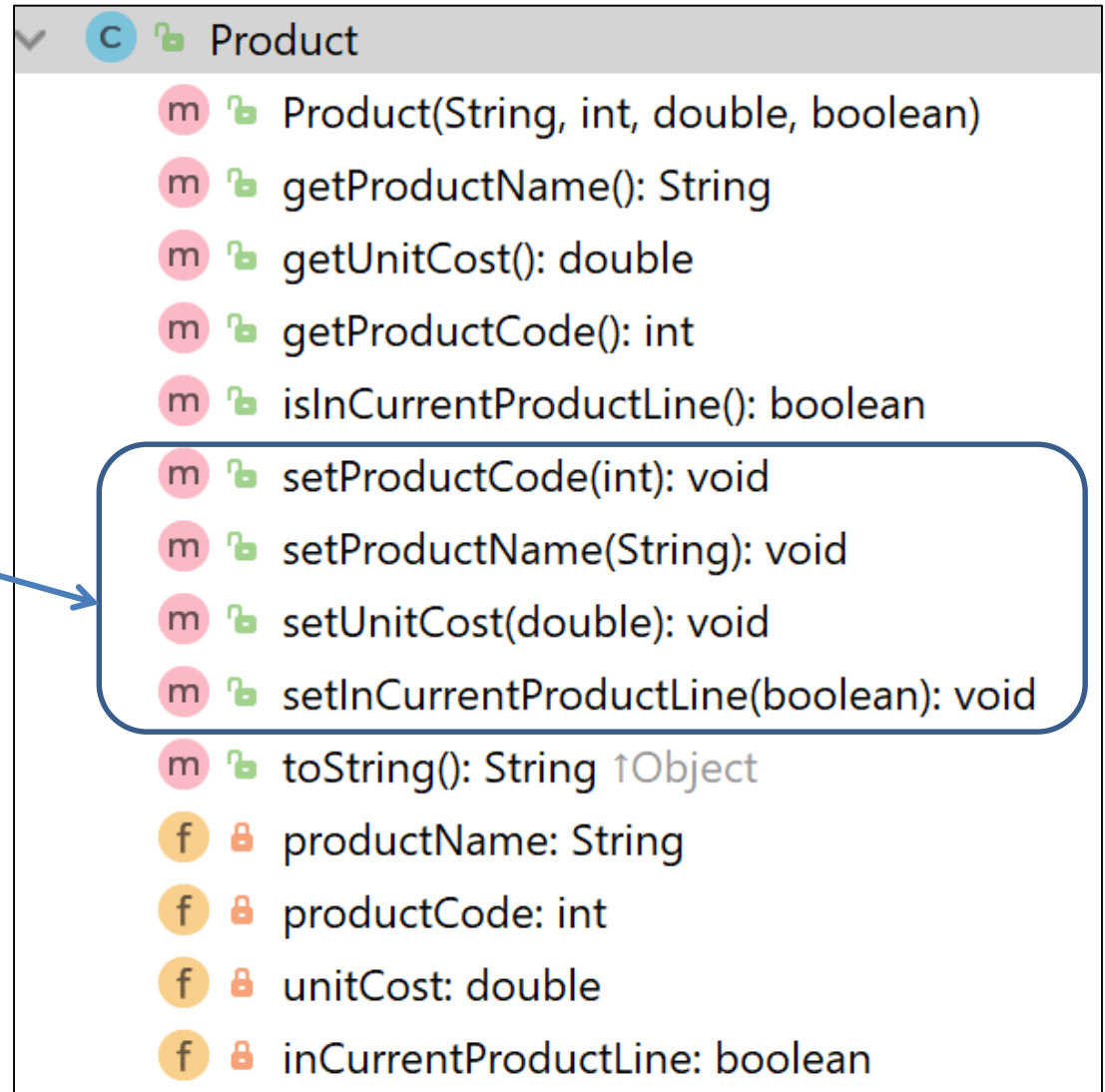


A Product Class...getters












```
public String getProductName() {  
    return productName;  
}  
  
public double getUnitCost() {  
    return unitCost;  
}  
  
public int getProductCode() {  
    return productCode;  
}  
  
public boolean isInCurrentProductLine() {  
    return inCurrentProductLine;  
}
```


A Product Class...**setters**


setters





Product

- m  Product(String, int, double, boolean)
- m  getProductName(): String
- m  getUnitCost(): double
- m  getProductCode(): int
- m  isInCurrentProductLine(): boolean
- m  setProductCode(int): void
- m  setProductName(String): void
- m  setUnitCost(double): void
- m  setInCurrentProductLine(boolean): void
- m  toString(): String  Object

f  productName: String

f  productCode: int

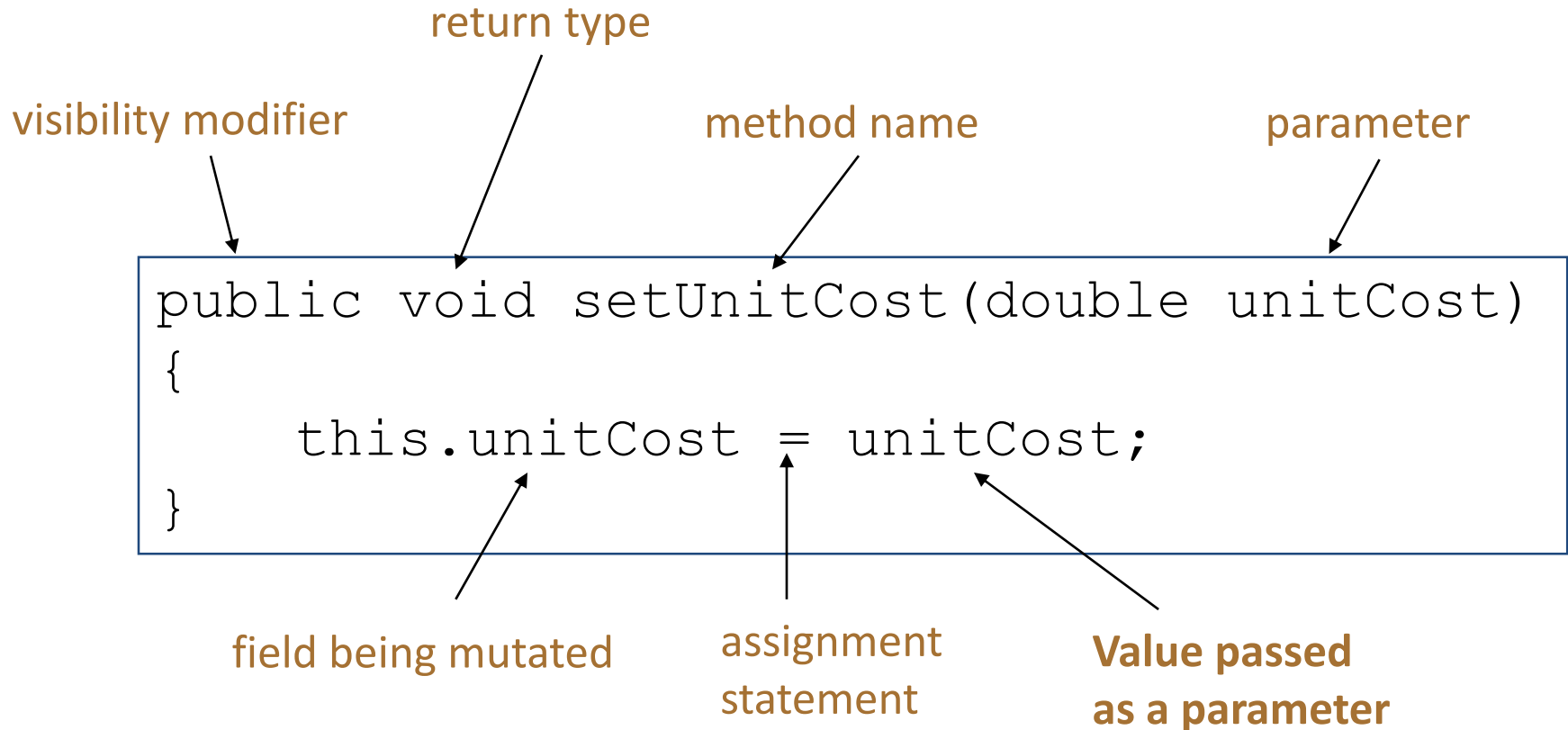
f  unitCost: double

f  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



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 setCurrentProductLine(boolean inCurrentProductLine) {
    this.inCurrentProductLine = inCurrentProductLine;
}
```

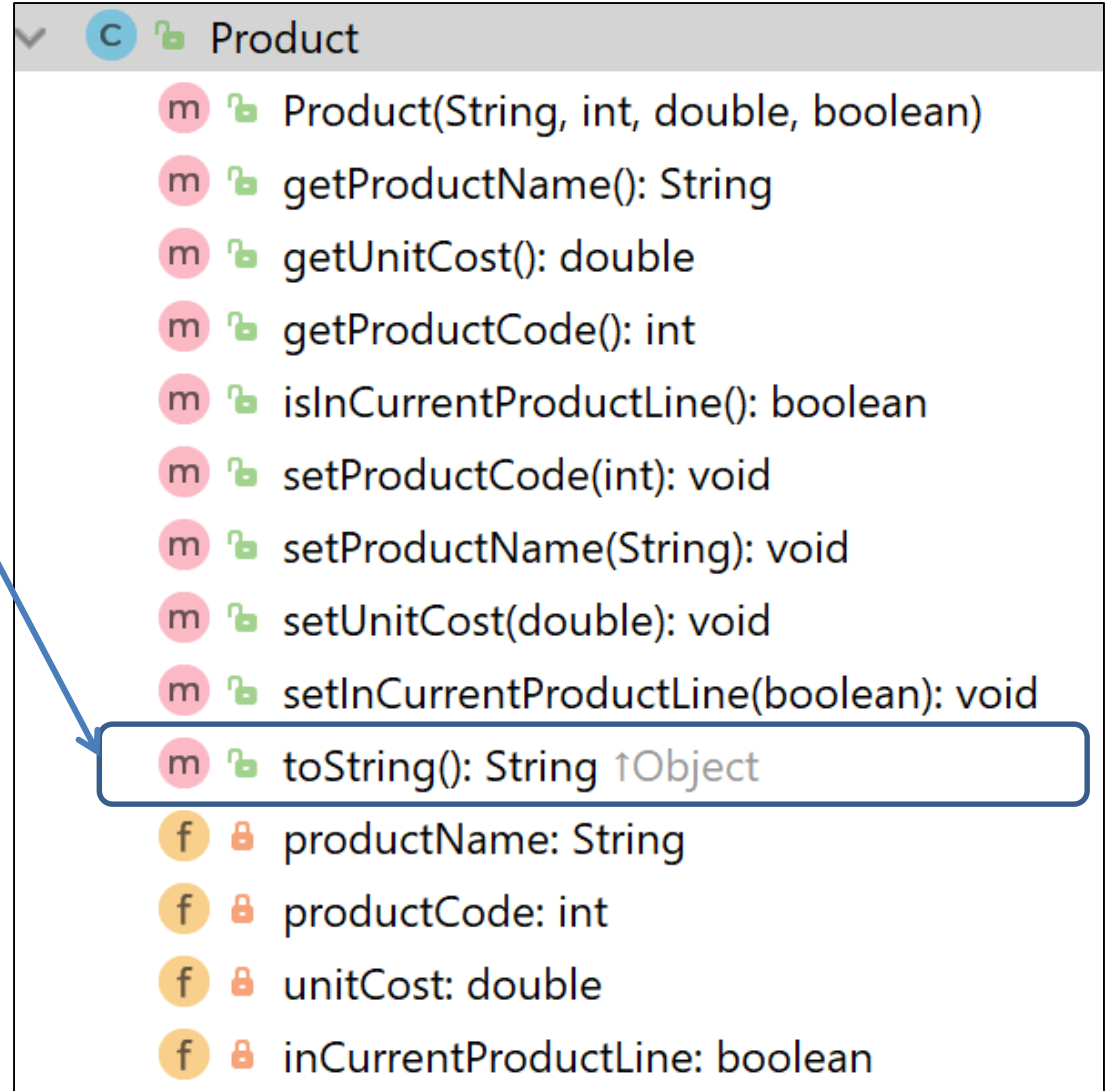
Getters/Setters

- For **each instance field** in a class, you are normally asked to write:
 - A **getter**
 - Return statement
 - A **setter**
 - Assignment statement
















A Product Class...toString

toString():

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



Product

- m  Product(String, int, double, boolean)
- m  getProduct_name(): String
- m  getUnitCost(): double
- m  getProductCode(): int
- m  islnCurrentProductLine(): boolean
- m  setProductCode(int): void
- m  setProduct_name(String): void
- m  setUnitCost(double): void
- m  setlnCurrentProductLine(boolean): void
- m  toString(): String 
- f  product_name: String
- f  productCode: int
- f  unitCost: double
- f  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);
```

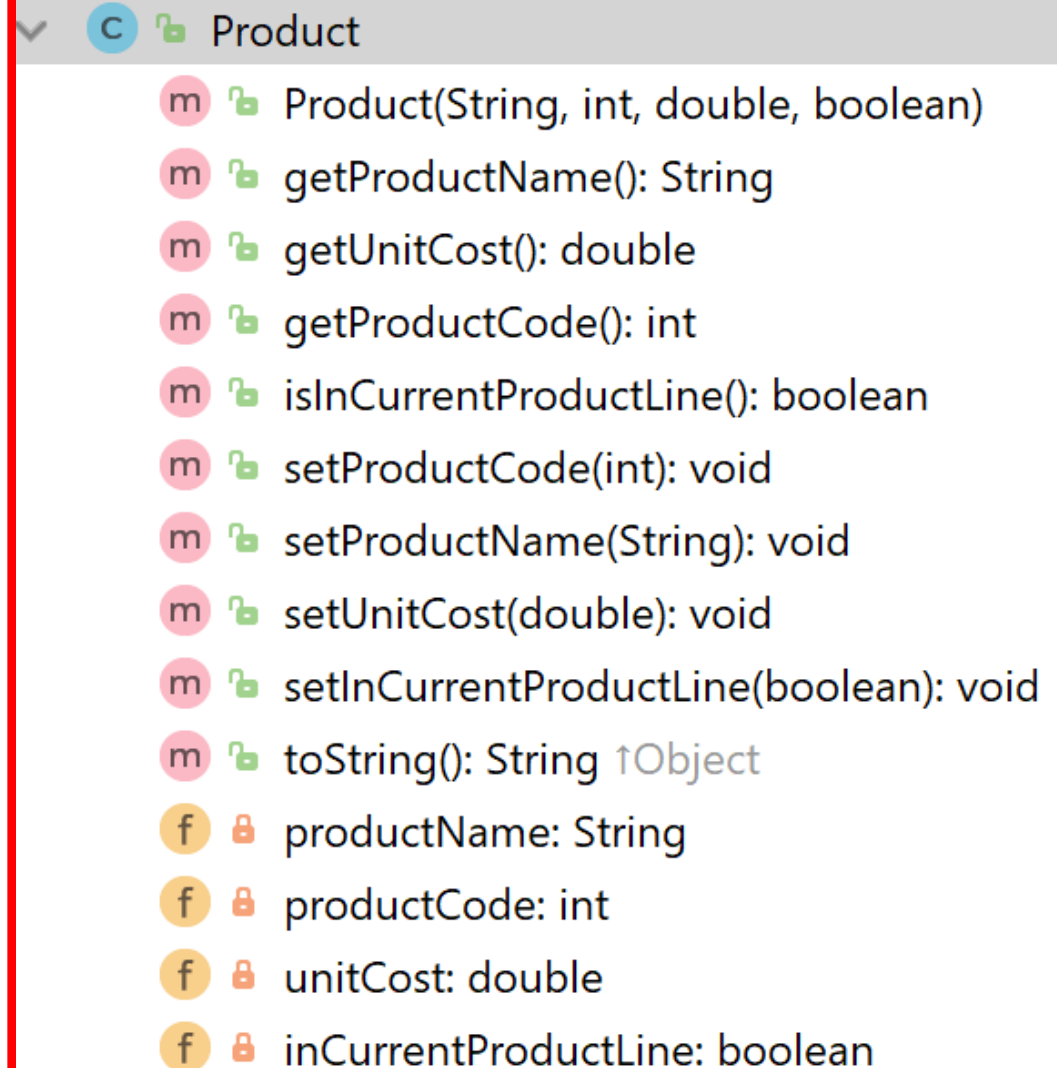
```
System.out.println(product.toString());
```

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.	<pre>public class <i>ClassName</i> { <i>Fields</i> <i>Constructors</i> <i>Methods</i> }</pre>
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> setter and getter methods to modify and view the fields values.

A Product Class... An Encapsulated Class

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



The screenshot shows the 'Product' class in an IDE. It lists 11 methods (marked with 'm' in a pink circle) and 4 fields (marked with 'f' in an orange circle). A red arrow points from the text 'single unit' in the first block to the class definition in this block.

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

The screenshot shows the `Product` class in an IDE. It lists several methods (marked with 'm' and a lock icon) and four private fields (marked with 'f' and a lock icon). A red box highlights the private fields, and a red oval is drawn around them. A red arrow points from the text 'single unit' in the first box to the entire class listing. Another red arrow points from the text 'hidden' in the second box to the private fields.

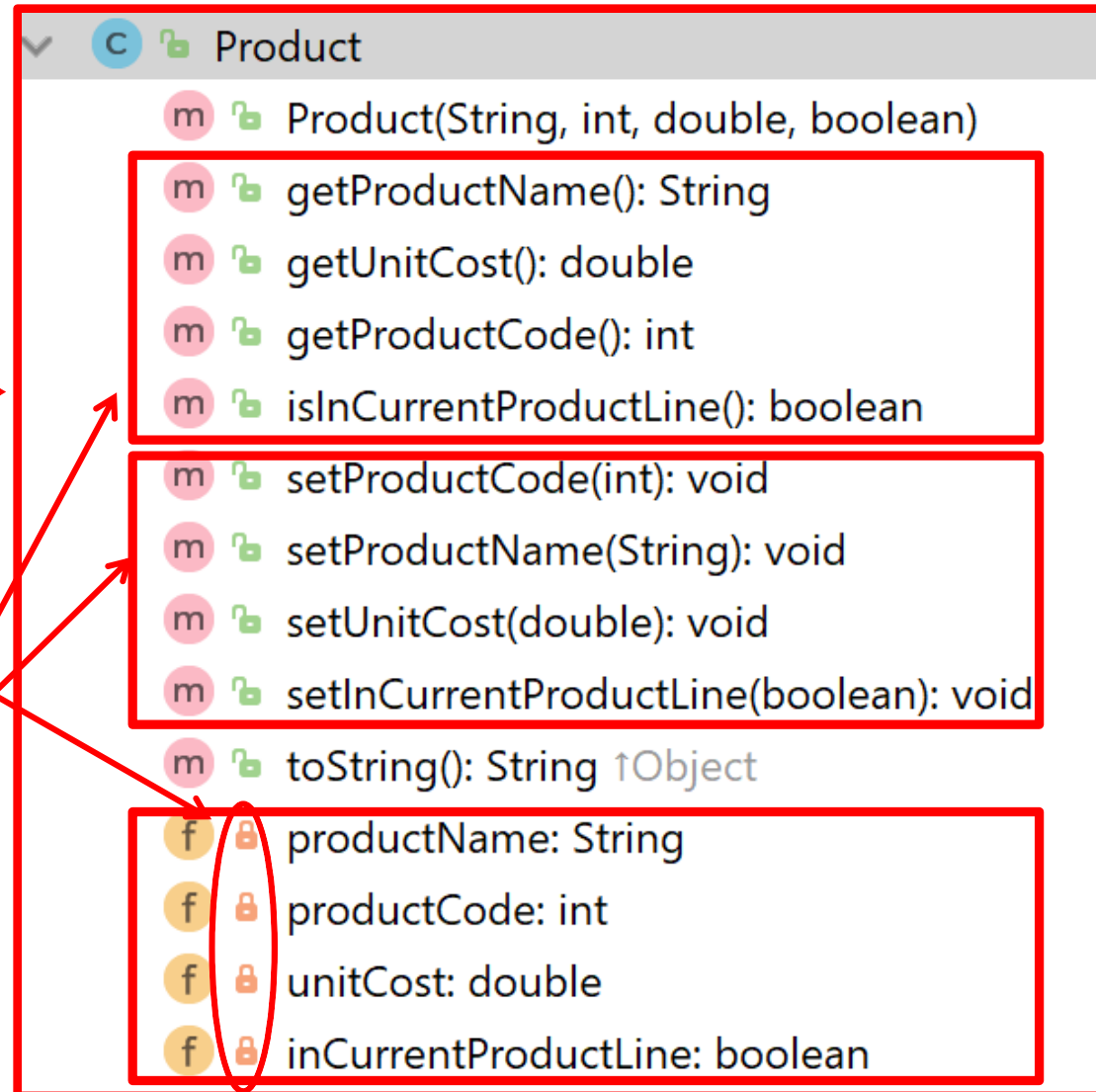
```
Product
m Product(String, int, double, boolean)
m getProductName(): String
m getUnitCost(): double
m getProductCode(): int
m isInCurrentProductLine(): boolean
m setProductCode(int): void
m setProductName(String): void
m setUnitCost(double): void
m setInCurrentProductLine(boolean): void
m toString(): String ↑Object
f productName: String
f productCode: int
f unitCost: double
f 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. **getter** and **setter** methods).



Using the Product Class

1

```
private Product product;
```

Declaring an object
product, of type
Product.

product

null

Using the Product Class

1 `private Product product;`

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

product



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

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

product

productName

"TV"

productCode

1234

unitCost

149.99

inCurrentProductLine

true

Multiple Product Objects

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

product



product

productName

"TV"

productCode

1234

unitCost

149.99

inCurrentProductLine

true

Multiple Product Objects

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

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

product



product

productName

"TV"

productCode

1234

unitCost

149.99

inCurrentProductLine

true

phone



phone

productName

"iPhone8"

productCode

1001

unitCost

349.99

inCurrentProductLine

false

Questions?

