RECAP: Arrays and Classes

Produced by:

Dr. Siobhán Drohan
Ms. Maireád Meagher
Ms. Siobhan Roche

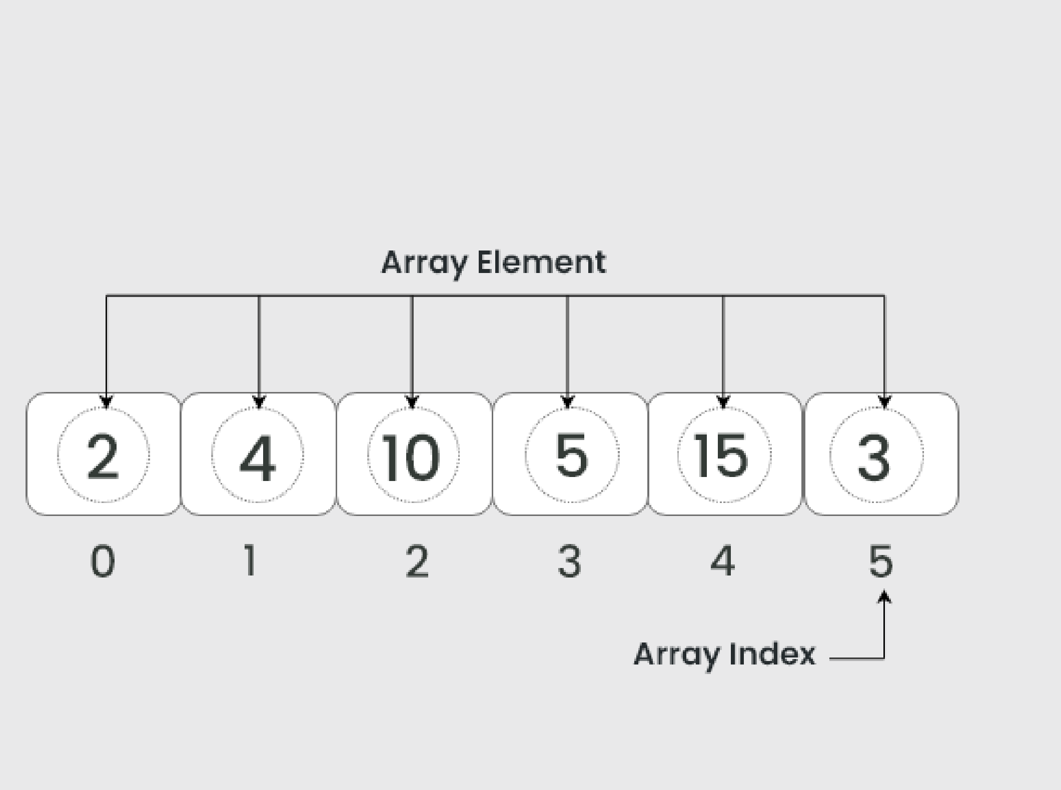
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

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

What is an Array?

An **array** is a collection of variables:

- All elements are the **same type**
- Stored in **contiguous memory locations**
- Accessed using an **index** (starting at 0)
- Has a **fixed size** once created



Let’s Look at arrays of different types

Arrays can store any type of data

Let’s look at some examples:

1. Array of primitives - **int**
2. Array of objects – **String**
3. Array of objects - **Product**

An array can store any type of data.

Primitive Types

```
int[] numbers = new int[10];

byte[] smallNumbers = new byte[4];

char[] characters = new char[26];
```

Object Types

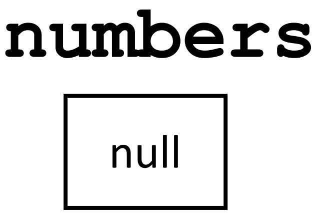
```
String[] words = new String[4];

Product[] products = new Product[10];
```

1) Array of **Primitives**

e.g. int

Structure of an **int** primitive array

```
int[]  numbers;
```



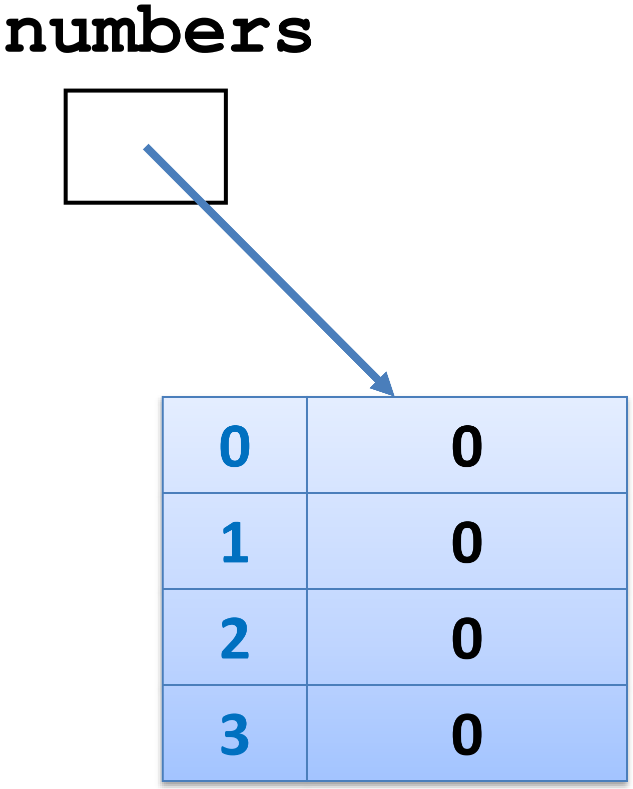
numbers

null

Structure of an **int** primitive array

```
int[]  numbers;
```

```
numbers = new int[4];
```



`numbers`

0	0
1	0
2	0
3	0

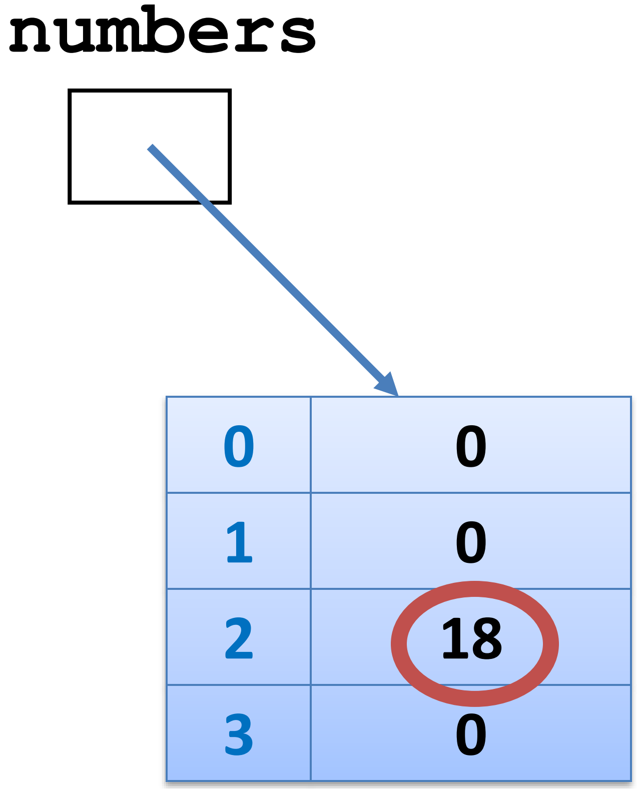
Structure of an **int** primitive array

```
int[]  numbers;
```

```
numbers = new int[4];
```

```
numbers[2] = 18;
```

We are directly accessing the element at index **2** and setting it to a value of **18**.



0	0
1	0
2	18
3	0

Structure of an **int** primitive array

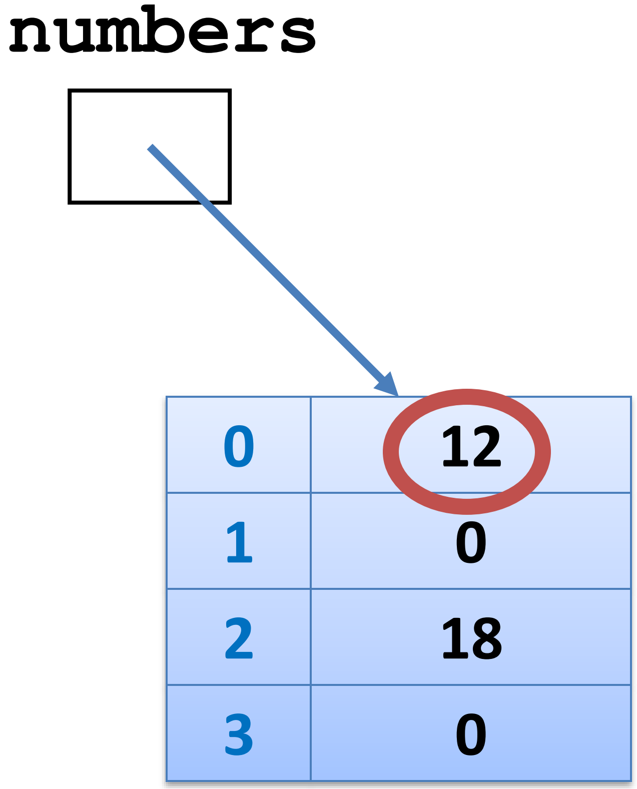
```
int[]  numbers;
```

```
numbers = new int[4];
```

```
numbers[2] = 18;
```

```
numbers[0] = 12;
```

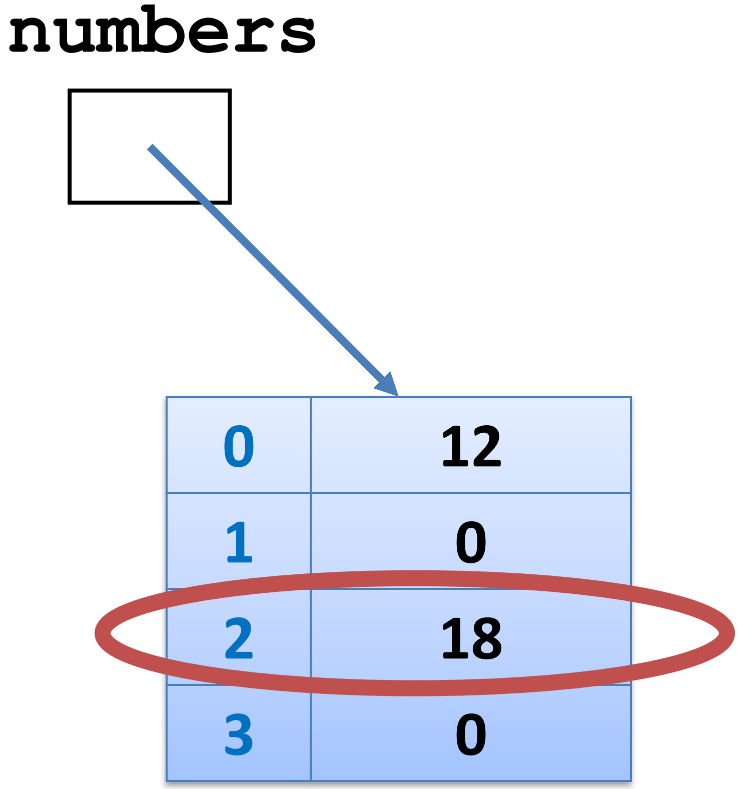
We are setting the element at index **0** to a value of **12**.



0	12
1	0
2	18
3	0

Structure of an **int** primitive array

```
int[]  numbers;

numbers = new int[4];

numbers[2] = 18;

numbers[0] = 12;

print(numbers[2]);
```



0	12
1	0
2	18
3	0

Here we are printing the contents of index location 2
i.e. 18 will be printed to the console.

2) Array of **Objects**
e.g. String

An array can store any type of data.

Primitive Types

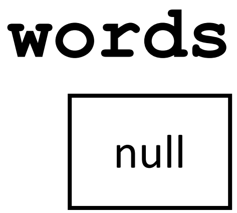
```
int[] numbers = new int[10];

byte[] smallNumbers = new byte[4];

char[] characters = new char[26];
```

Object Types

```
String[] words = new String[4];

Product[] products = new Product[10];
```

Structure of a **String** object array

```
String[]  words;
```



words

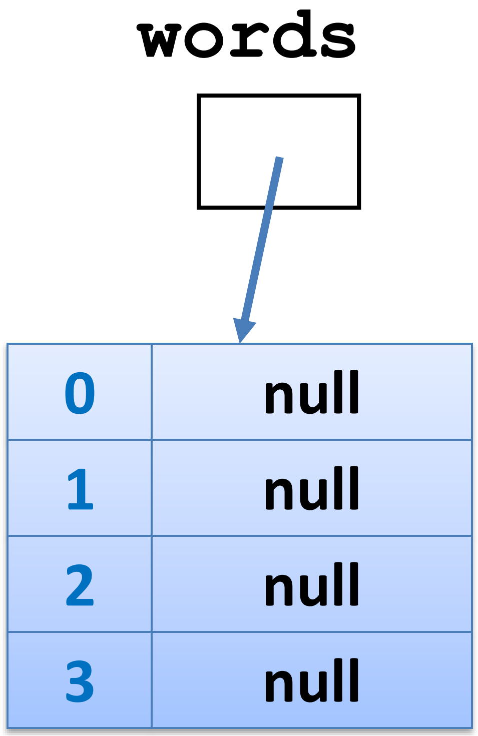
null

NOTE: words holds a reference to an array, the array hasn’t been created yet

Structure of a **String** object array

```
String[]  words;
```

```
words = new String[4];
```



0	null
1	null
2	null
3	null

NOTE:
The array holds references to objects.
No String objects exist yet
Each element is set to null

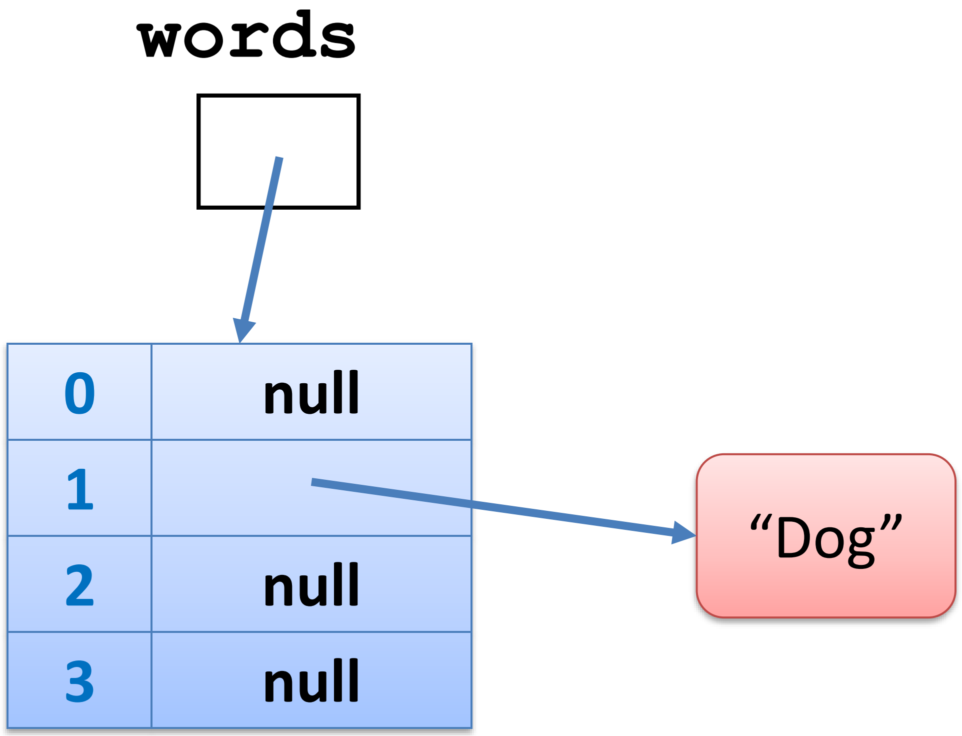
Creating the array does NOT create objects

Structure of a **String** object array

```
String[]  words;
```

```
words = new String[4];
```

```
words[1] = “Dog”;
```



0	null
1	
2	null
3	null

NOTE:
Objects are created and stored at specific indexes.
Other positions remain null

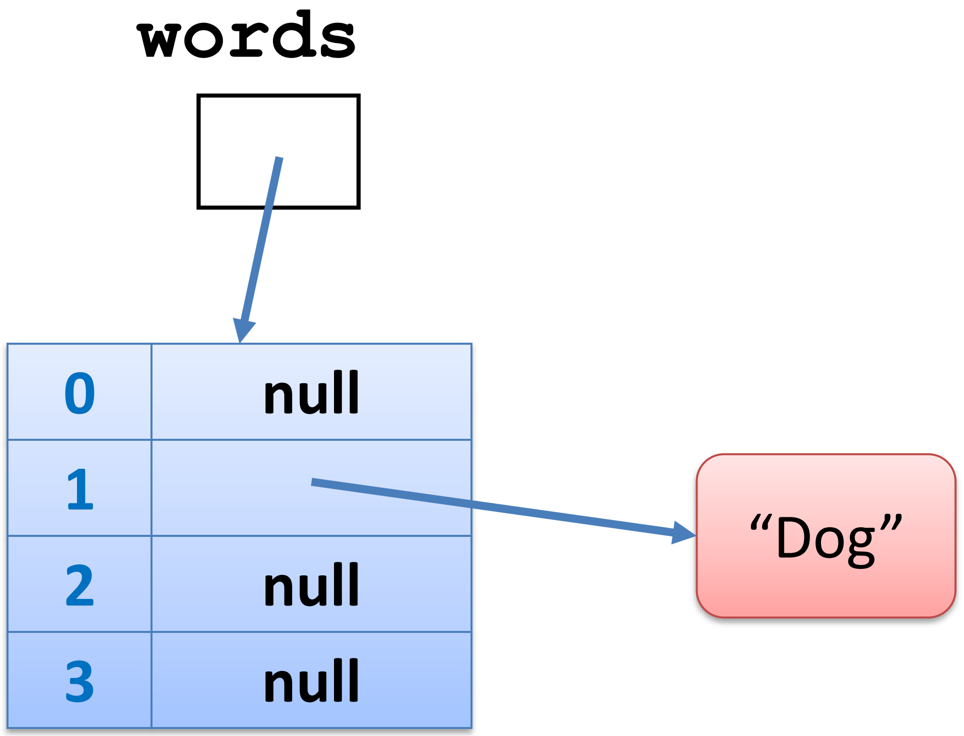
Structure of a **String** object array

```
String[]  words;
```

```
words = new String[4];
```

```
words[1] = “Dog”;
```

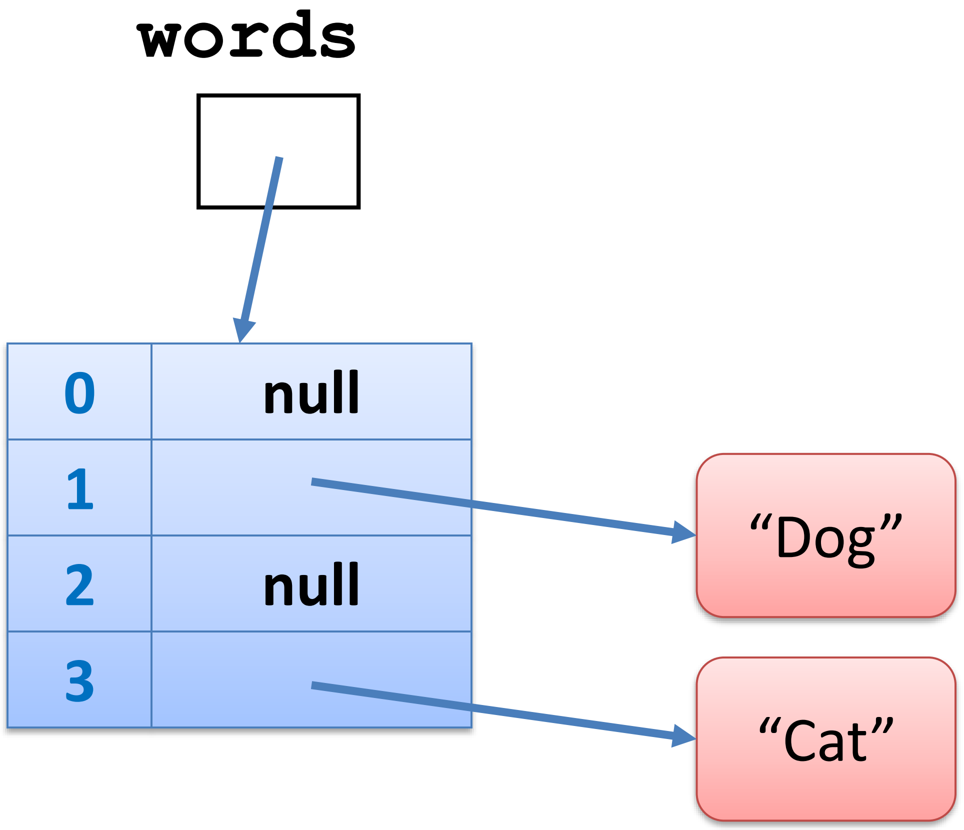
We are directly accessing the element at index **1** and setting it to a value of **“Dog”**.



0	null
1	
2	null
3	null

Structure of a **String** object array

```
String[]  words;

words = new String[4];

words[1] = “Dog”;

words[3] = “Cat”;
```



Structure of a **String** object array

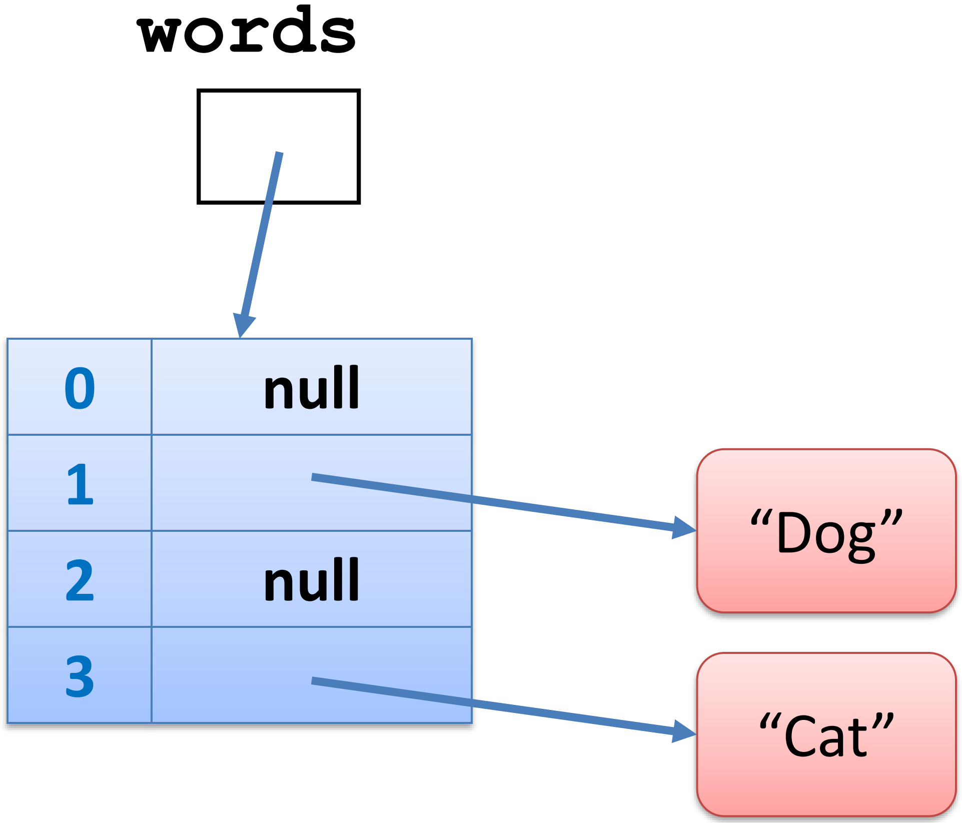
String[] words;

words = new String[4];

words[1] = “Dog”;

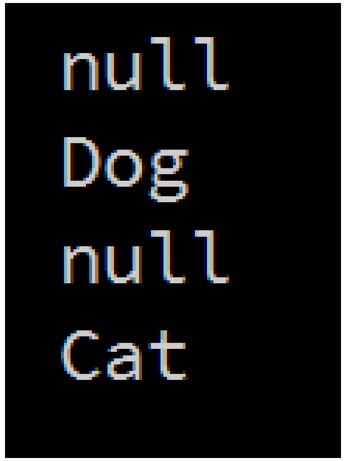
words[3] = “Cat”;

The element at index **3** is set to **“Cat”**.



Structure of a **String** object array

```
String[] words;

words = new String[4];

words[1] = "Dog";
words[3] = "Cat";

for (int i=0; i < words.length; i++)
{
   System.out.println(words[i]);
}
```



```
null
Dog
null
Cat
```

Why does null matter?

- If we try to use an object that is null:

```
words[0].length();
```

This causes a:

- – **NullPointerException**

This is a common runtime error when using arrays of objects. *(we will learn about Exception Handling later in the semester)*

3) Array of **Objects**

e.g. Product

An array can store any type of data.

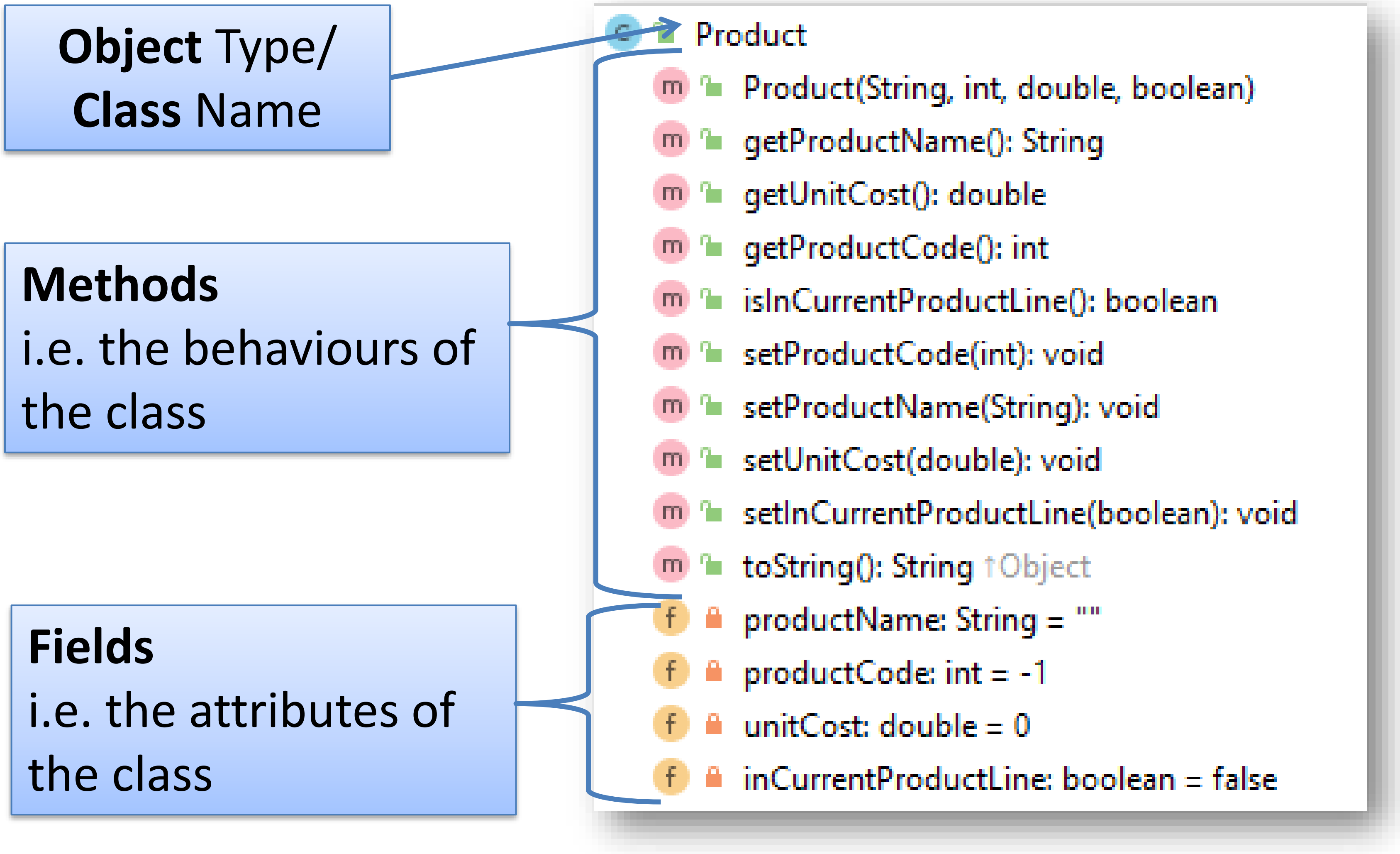
Primitive Types

```
int[] numbers = new int[10];

byte[] smallNumbers = new byte[4];

char[] characters = new char[26];
```

Object Types

```
String[] words = new String[4];

Product[] products = new Product[10];
```

Product Class



Object Type/
Class Name

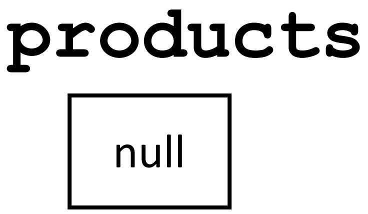
Methods
i.e. the behaviours of the class

Fields
i.e. the attributes of the class

```
c 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 = -1
  f unitCost: double = 0
  f inCurrentProductLine: boolean = false
```

Structure of a **Product** primitive array

```
Product[]  products;
```



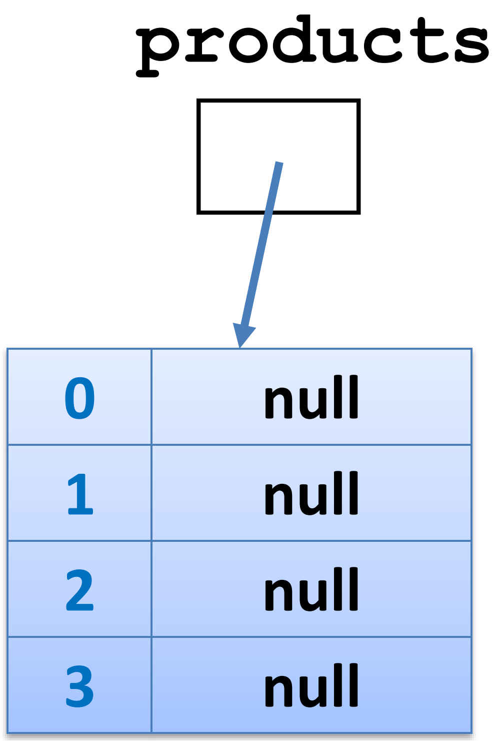
products

null

Structure of a **Product** primitive array

```
Product[] products;
```

```
products = new Product[4];
```



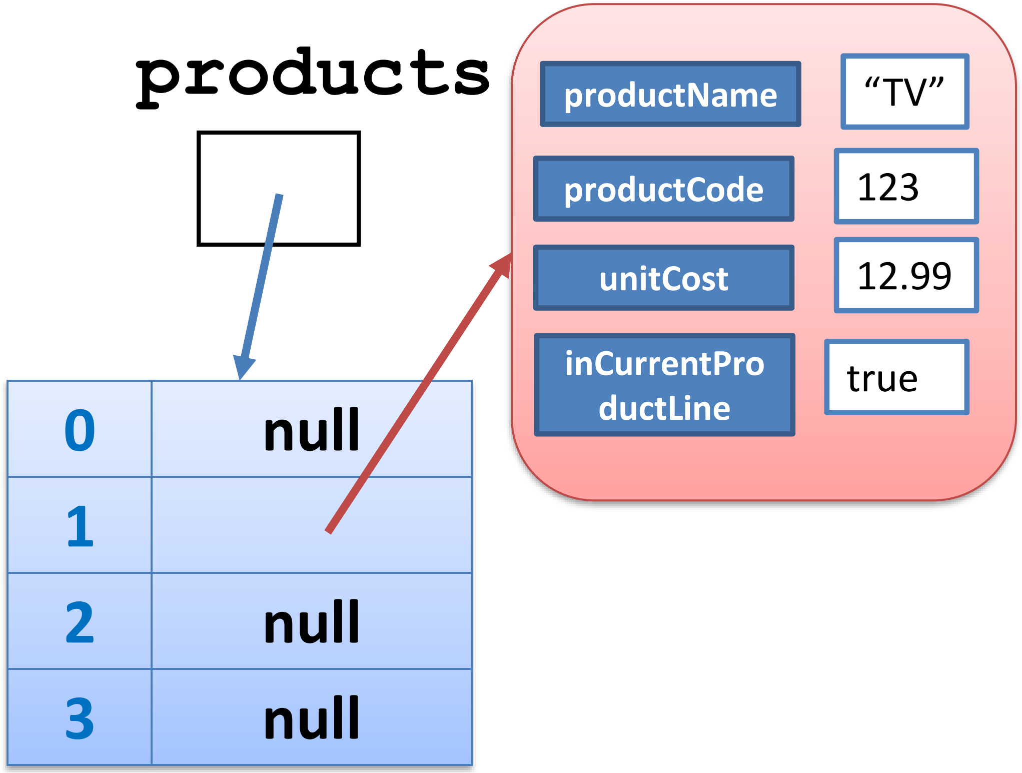
0	null
1	null
2	null
3	null

Note:
Each position can store a Product object
Initially, all positions are null

Structure of a **Product** primitive array

```
Product[] products;
```

```
products = new Product[4];
```



products

0	null
1	
2	null
3	null

productName	"TV"
productCode	123
unitCost	12.99
inCurrentProductLine	true

```
products[1] =  new Product("TV", 123, 12.99, true);
```

Example using a **Product** object array

```
public String listProducts() {

    String listOfProducts = "";

    for (int i = 0; i < total; i++) {
       listOfProducts += i + ": " + products[i].toString() + "\n";
    }

    return listOfProducts;
  }
}
```

Returns a String containing all the products stored in the primitive array.

Note:
We need a separate total variable to keep track of how many products added to array

The array does not track how many objects are stored

Limitations of Arrays

Arrays have several drawbacks:

- Fixed size (cannot grow or shrink)
- We must track how many elements are used
- Unused positions contain null
- Easy to cause runtime errors

Why This Matters

When programs become larger:

- Managing arrays becomes error-prone
- Code becomes harder to read and maintain
- This leads us to a better solution.

Introducing ArrayList (Next Week)

ArrayList:

- Grows automatically
- Tracks its own size
- Stores objects only
- Reduces null problems

Questions?