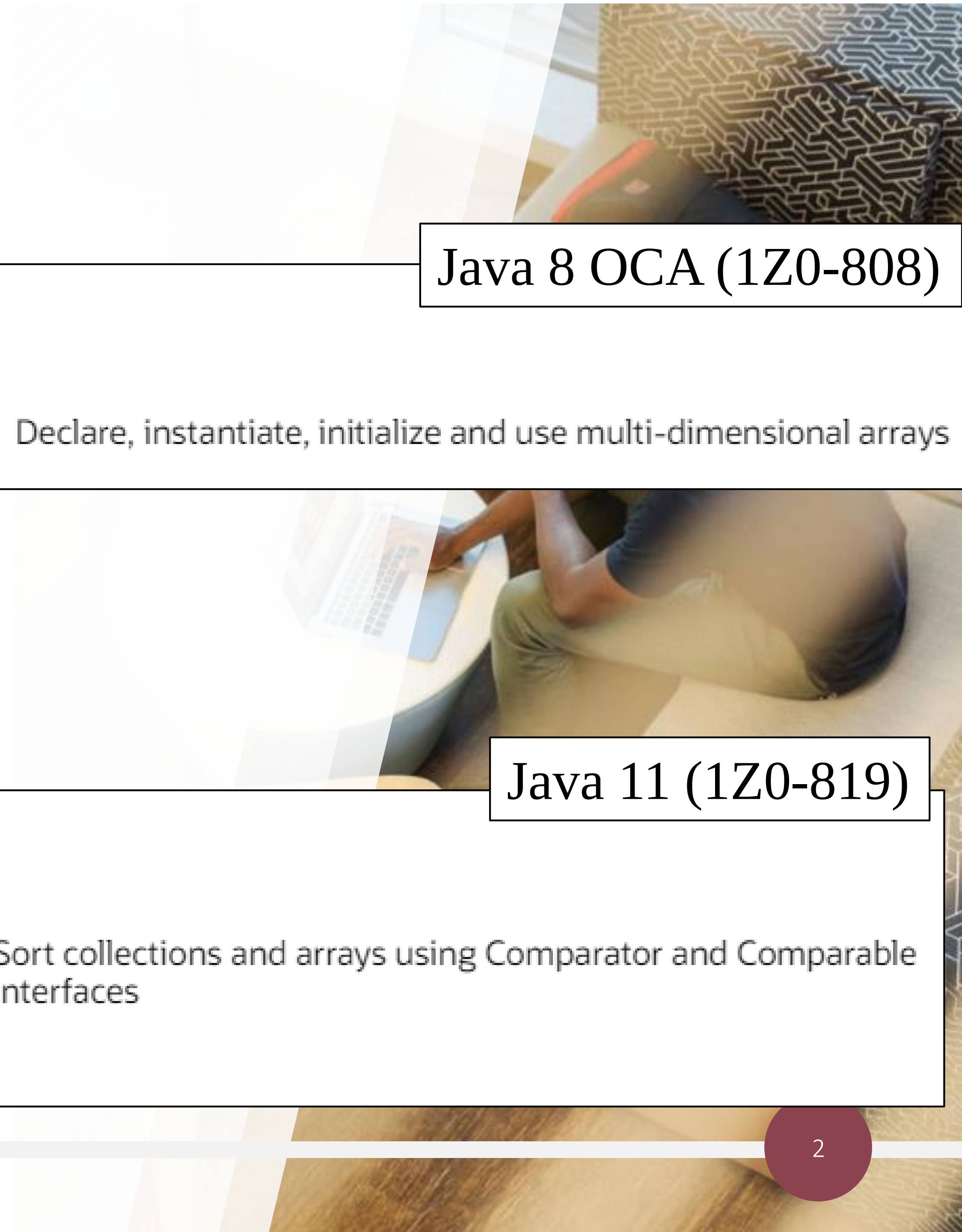
Arrays

Arrays



Java 8 OCA (1Z0-808)

Creating and Using Arrays

- Declare, instantiate, initialize and use a one-dimensional array
- Declare, instantiate, initialize and use multi-dimensional arrays

Java 11 (1Z0-819)

Working with Arrays and Collections

- ✓ Use generics, including wildcards
- ✓ Use a Java array and List, Set, Map and Deque collections, including convenience methods
- ✓ Sort collections and arrays using Comparator and Comparable interfaces

Arrays

- Arrays are objects that store multiple variables of the same type.

- Arrays can hold either primitives or object references but the array itself will always be an object on the heap (even if the array is declared to hold primitives).

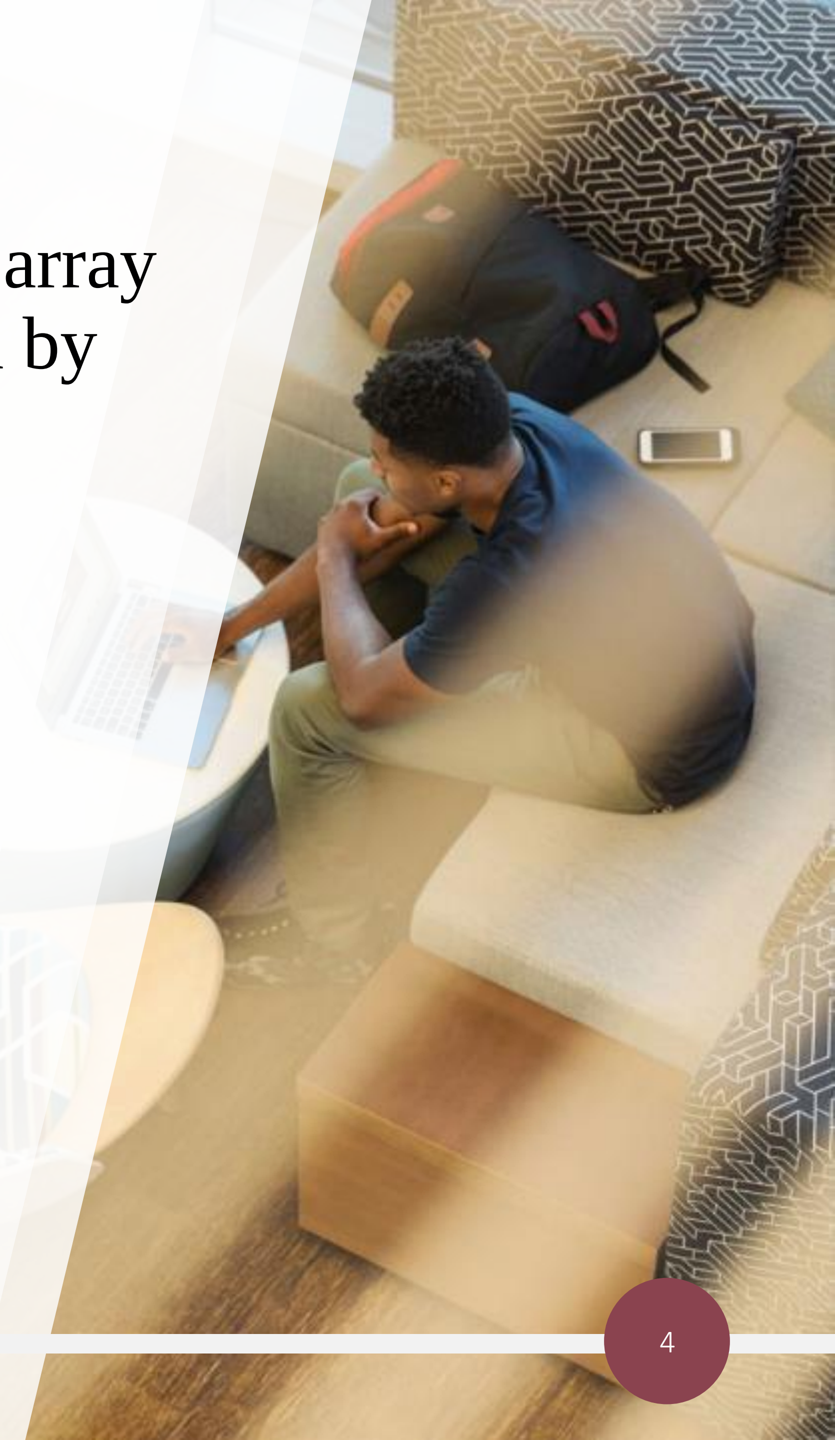
- There are three things to be aware of:
  - ➢ how to declare an array reference variable
  - ➢ how to construct an array object
  - ➢ how to initialise/populate the array with elements

Declaring an Array

- Arrays are declared by stating the type of element the array will hold, which can be an object or primitive, followed by square brackets to the left or right of the identifier.

- Declaring an array of primitives:
  - *int[] grades; // recommended*
  - *int grades[];*

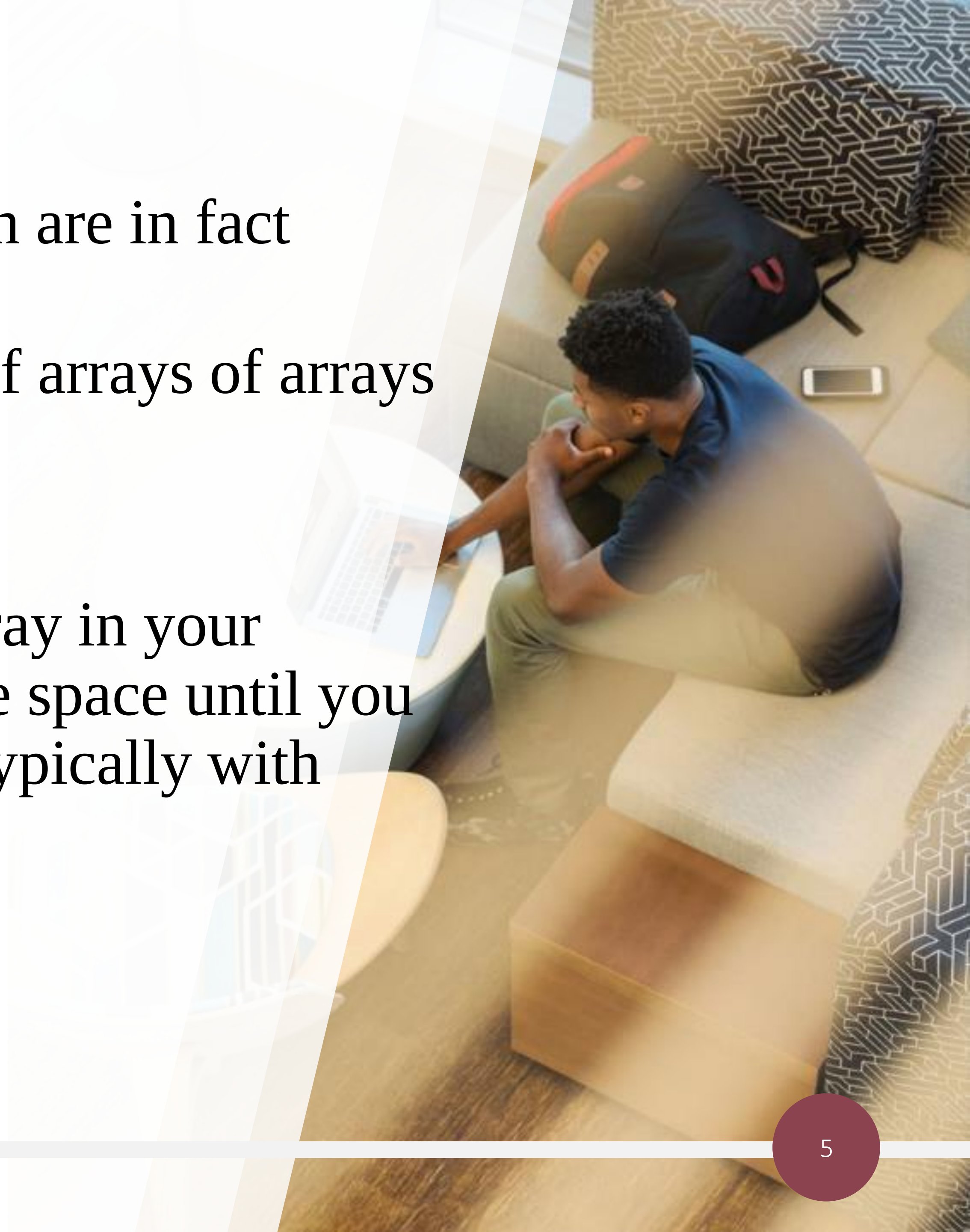
- Declaring an array of object references:
  - *String[] strings; // recommended*
  - *String strings[];*



Declaring an Array

- We can have multi-dimensional arrays which are in fact arrays of arrays.
  - *String [][][] threeDimArray*; // array of arrays of arrays
  - *String [] twoDimArray[];*

- It is **never** legal to include the size of the array in your declaration because the JVM does not allocate space until you actually instantiate (create) the array object (typically with *new).*
  - *int[5] grades*;   // compiler error

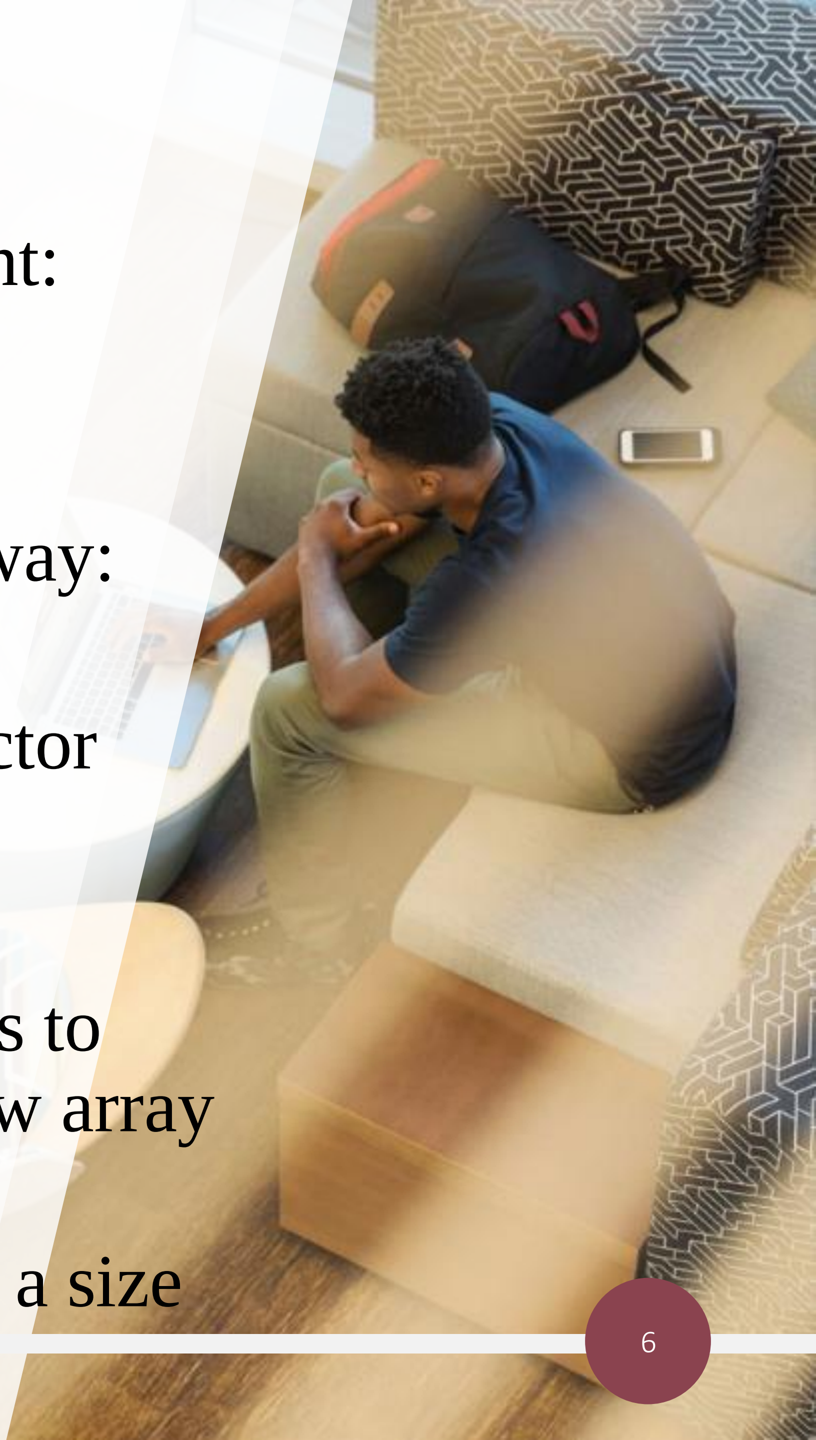


Constructing an Array

- Arrays can be declared and constructed in one statement:
    - *int[] grades = new int[4];*

- Arrays of object types can be constructed in the same way:
    - *Thread[] threads = new Thread[5];*
        - no *Thread* objects created (no *Thread* constructor invoked) ; just one *Thread* array object.

- Remember, the JVM (and therefore the compiler) needs to know how much space to allocate on the heap for the new array object when it is constructed:
    - *int[] grades = new int**[ ]**;* // compiler error, needs a size

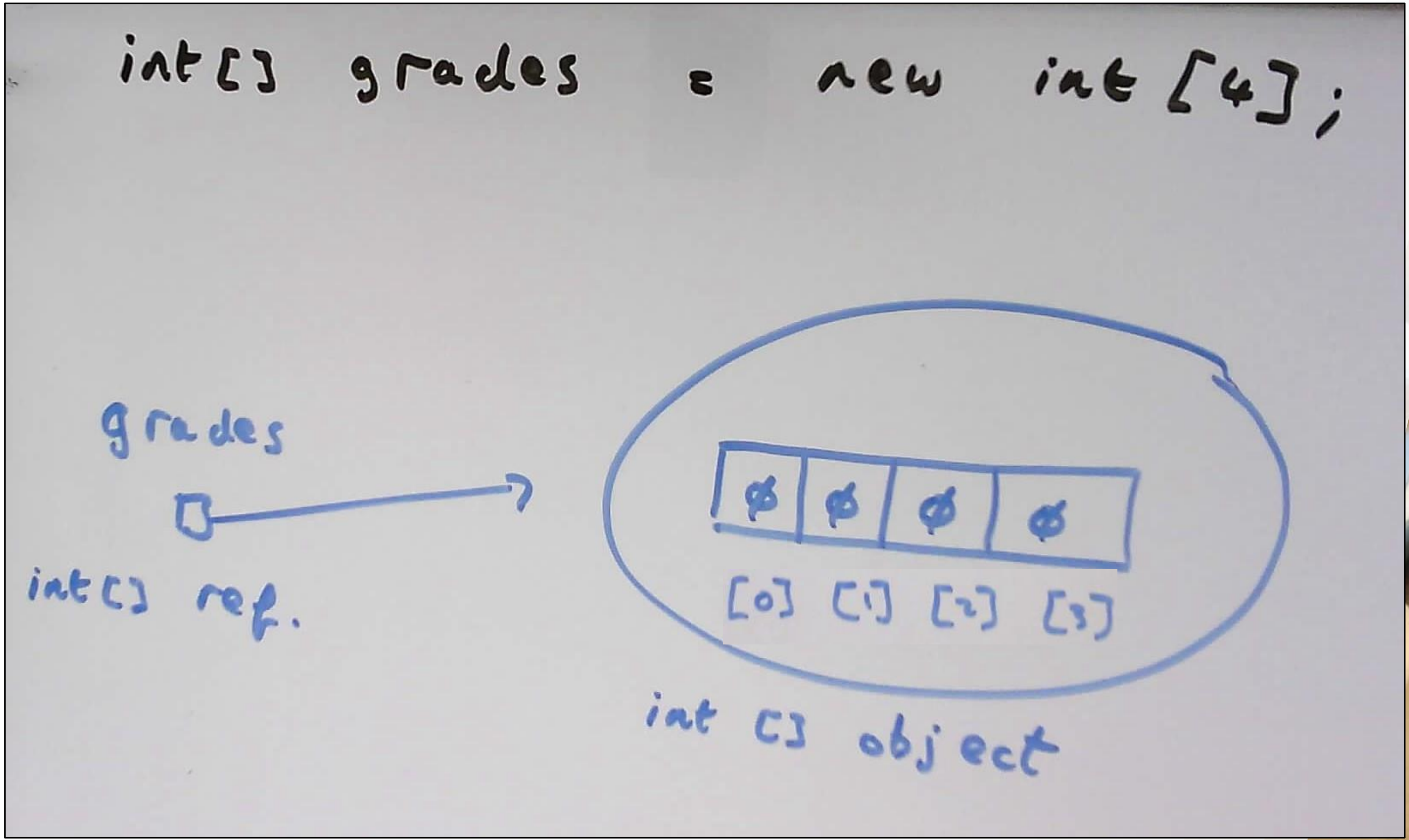


Constructing an Array

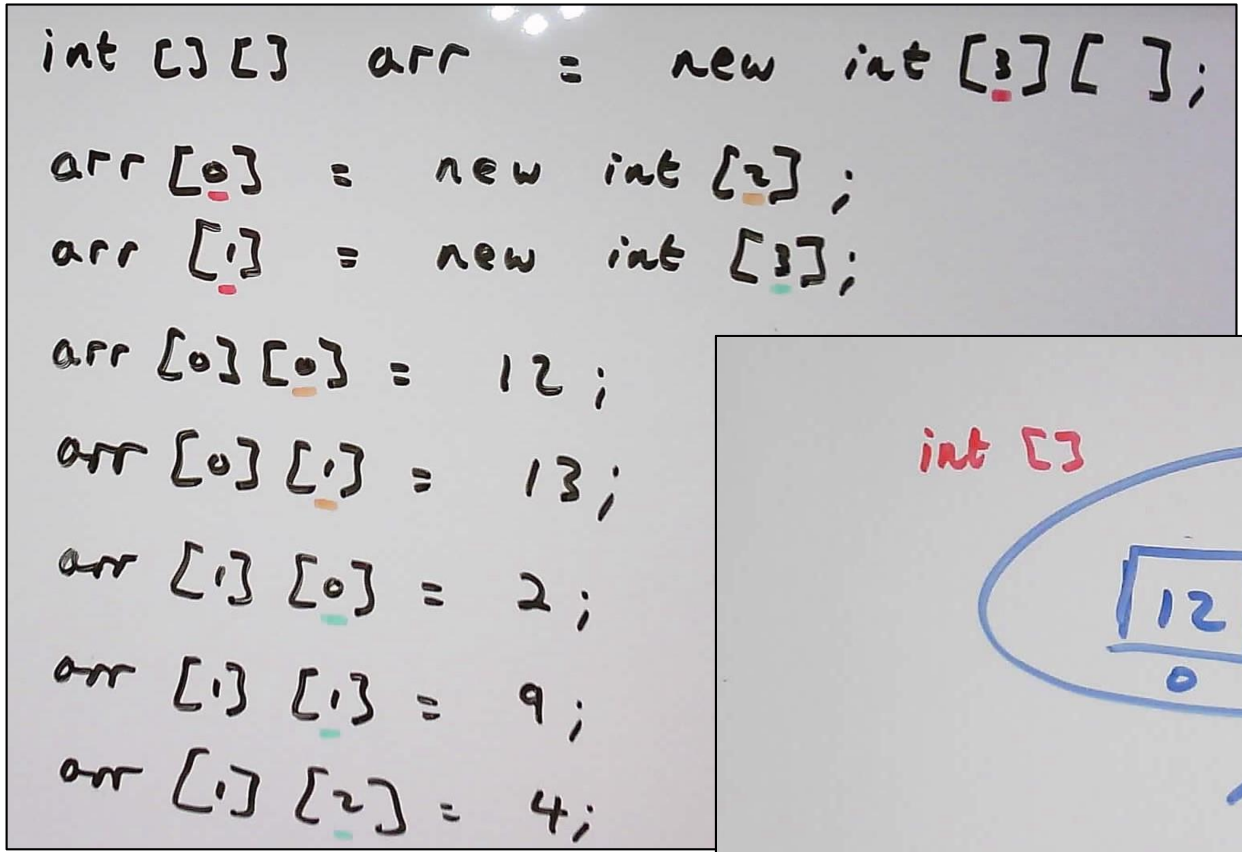
- In addition to being constructed with *new*, arrays can also be created using shorthand syntax (covered later) that both creates the array and initialises its elements to values supplied in the code (as opposed to default values).

- Therefore, because of these syntax shortcuts, objects can still be created without the keyword *new.*

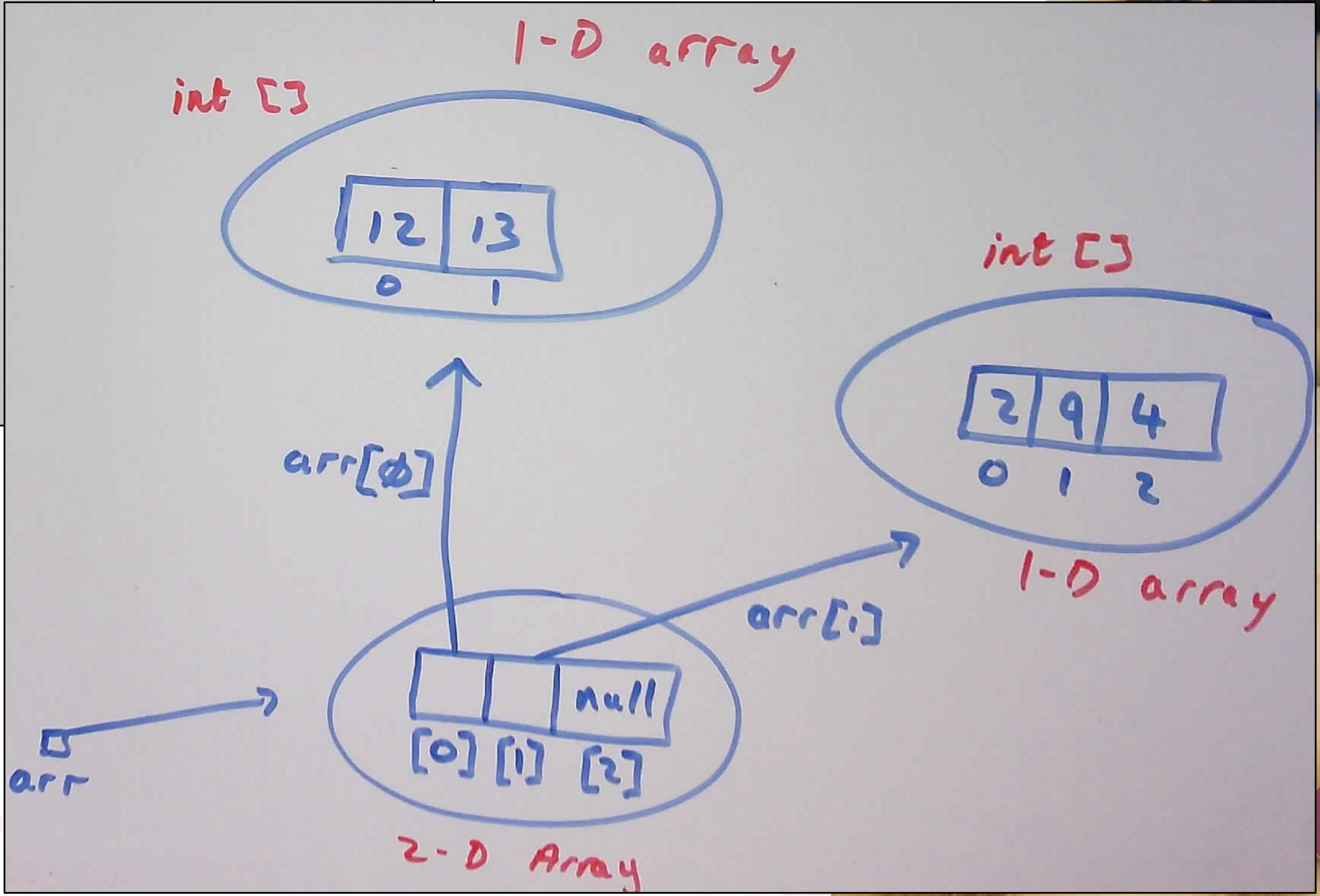
- Multidimensional arrays are arrays of arrays. The sizes on the right hand size must be populated <u>from left to right</u>:
  - *int [][]a = new int[3][];// ok*
  - *int [][]b = new int[][] // compiler error*
  - *int [][]c = new int[][3];// compiler error*



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



```
int [] [] arr = new int [3] [ ];
arr [0] = new int [2] ;
arr [1] = new int [3];
arr [0] [0] = 12 ;
arr [0] [1] = 13;
arr [1] [0] = 2 ;
arr [1] [1] = 9 ;
arr [1] [2] = 4;
```

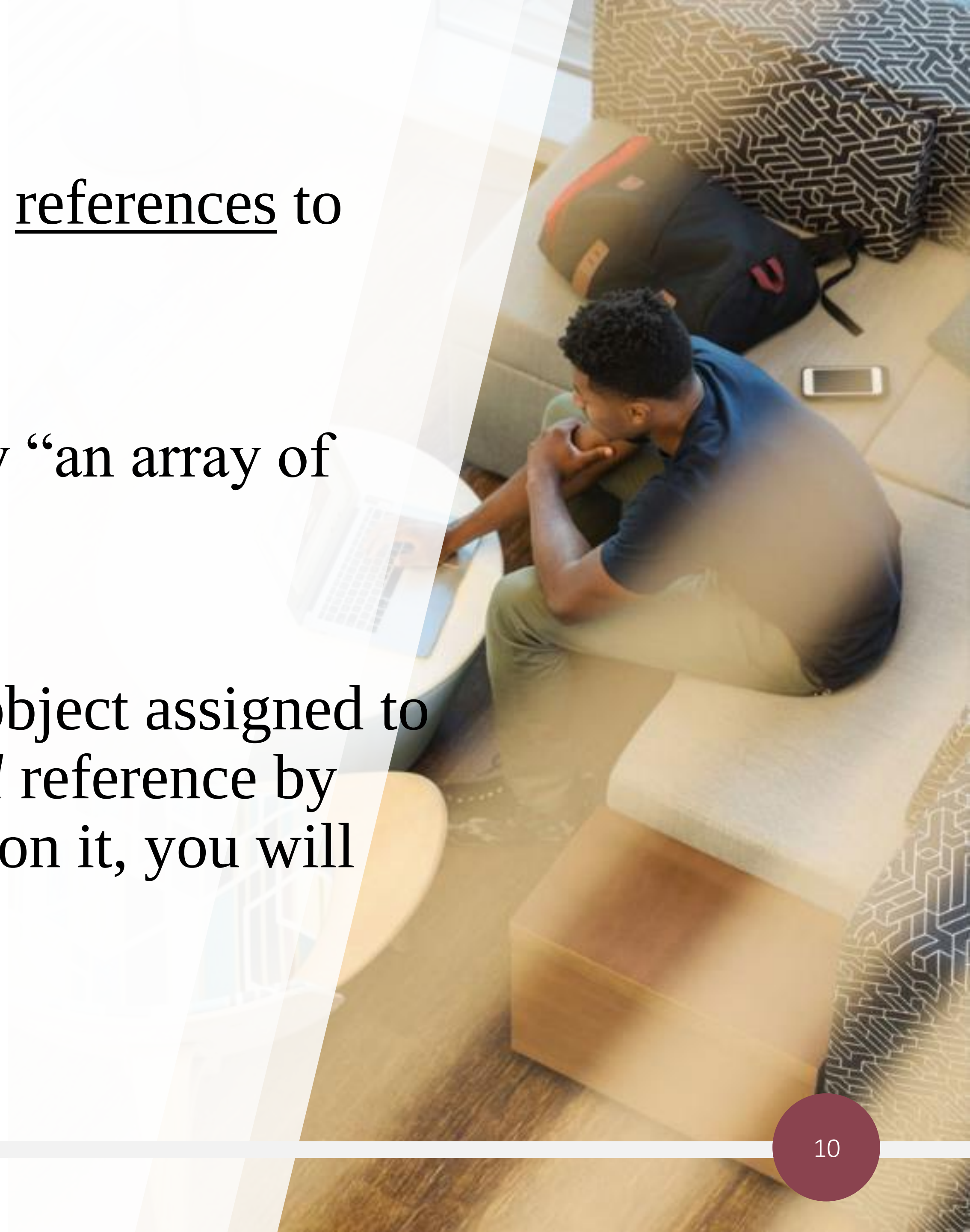


Initialising an Array

- An array of objects is essentially an array of references to those objects.

- Therefore, “an array of five strings” is really “an array of five references to *String* objects”.

- Remember, a reference that has not had an object assigned to it is a *null* reference. If you try to use that *null* reference by applying the dot operator to invoke a method on it, you will get a *NullPointerException.*



Initialising an Array

- An individual element in an array is accessed using an index number, with indices beginning at zero. Thus, an array of 5 elements goes from index 0 to 4 (length of the array - 1).

  -
    ```
    int [] x = new int[5];
    x[5] = 3; // ArrayIndexOutOfBoundsException

    int[] z = new int[2];
    int y = -3;
    z[y] = 4; // ArrayIndexOutOfBoundsException
    ```

- Array objects have a single *public* variable, *length*, that gives you the number of elements in the array. The last index value is always one less than the *length* i.e. *length* - 1.

Initialising an Array – syntax shortcuts

```
int x = 45;
int [] arr = {1, x, 99};
```
// size depends on the number of comma-separated elements between the { }.

- The above is the same as:

```
int []arr;
arr = new int[3];
int x=45;
arr[0]=1;
arr[1]=x;
arr[2]=99;
```

Initialising an Array – syntax shortcuts

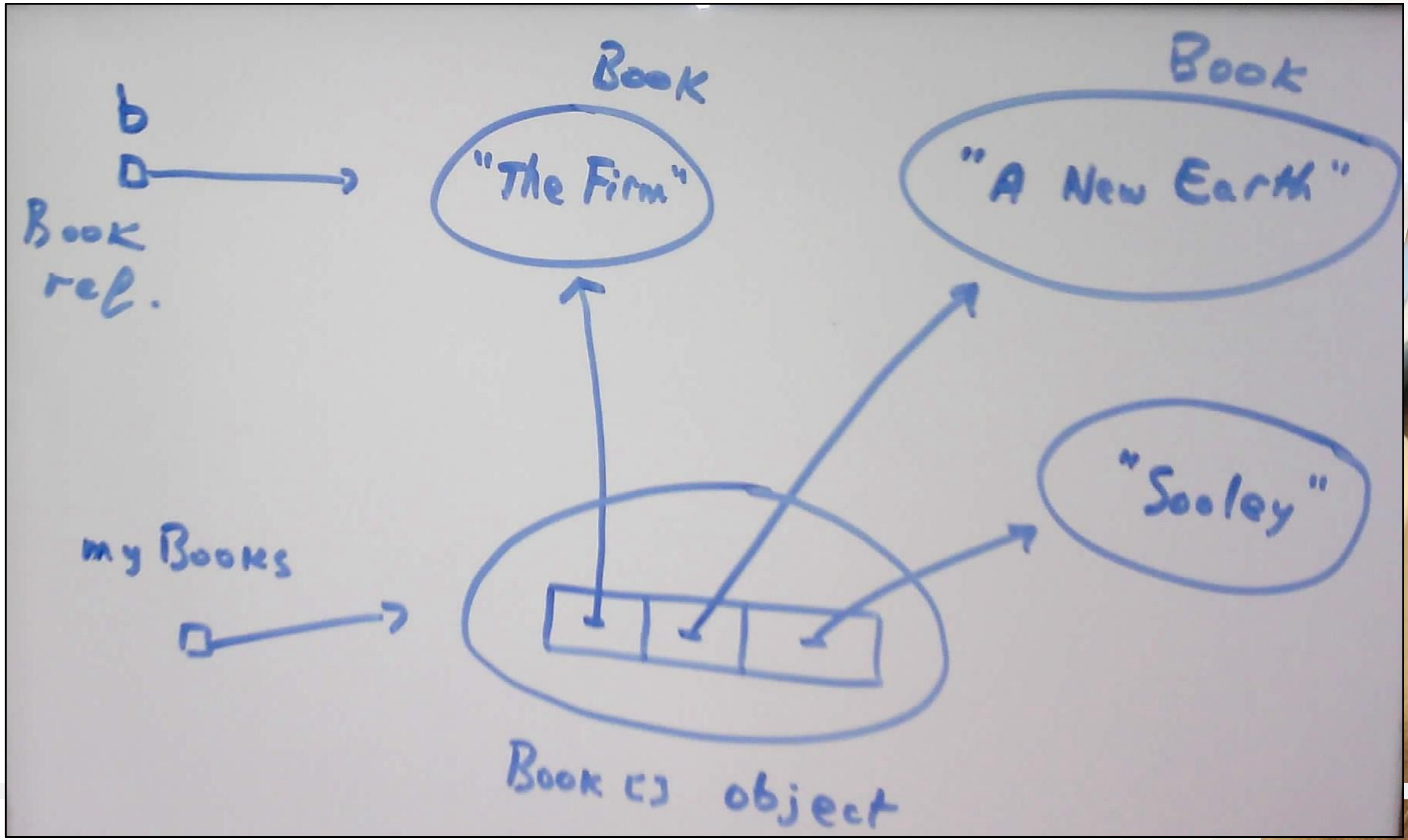
- We can do the same thing with object references:

```
Book b = new Book(“The Firm”);
Book[] myBooks = {b, new Book(“A New
Earth”), new Book(“Sooley”)};
```

- The above is the same as:

```
Book b = new Book(“The Firm”);
Book[] myBooks = new Book[3];
myBooks[0] = b;
myBooks[1] = new Book(“A New Earth”);
myBooks[2] = new Book(“Sooley”);
```

```
Book b = new Book("The Firm");
Book[] myBooks = {b, new Book("A New Earth"), new Book("Sooley")};
```



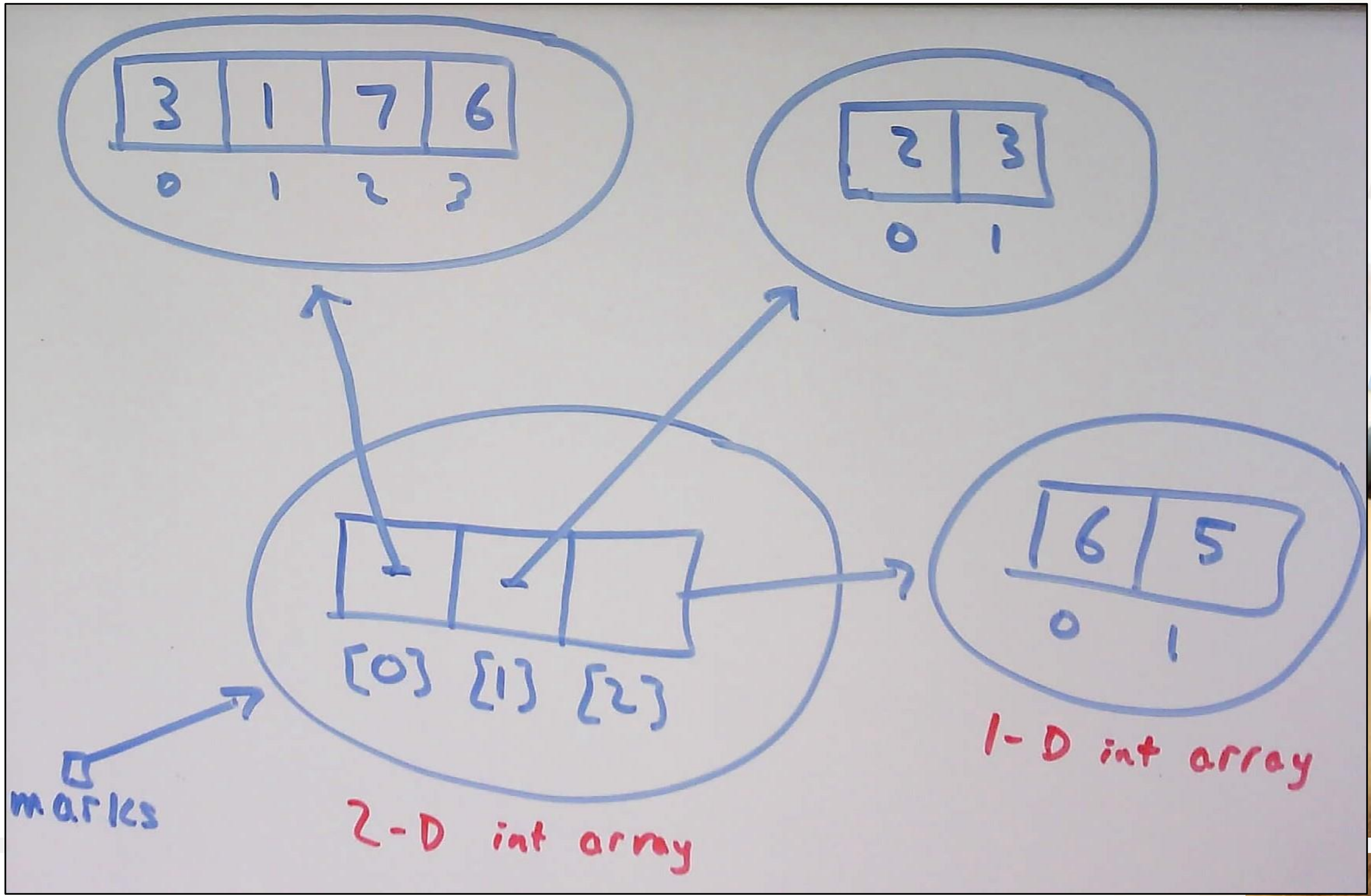
Initialising an Array – syntax shortcuts

- The shortcut syntax can also be used with multidimensional arrays.
- For example,

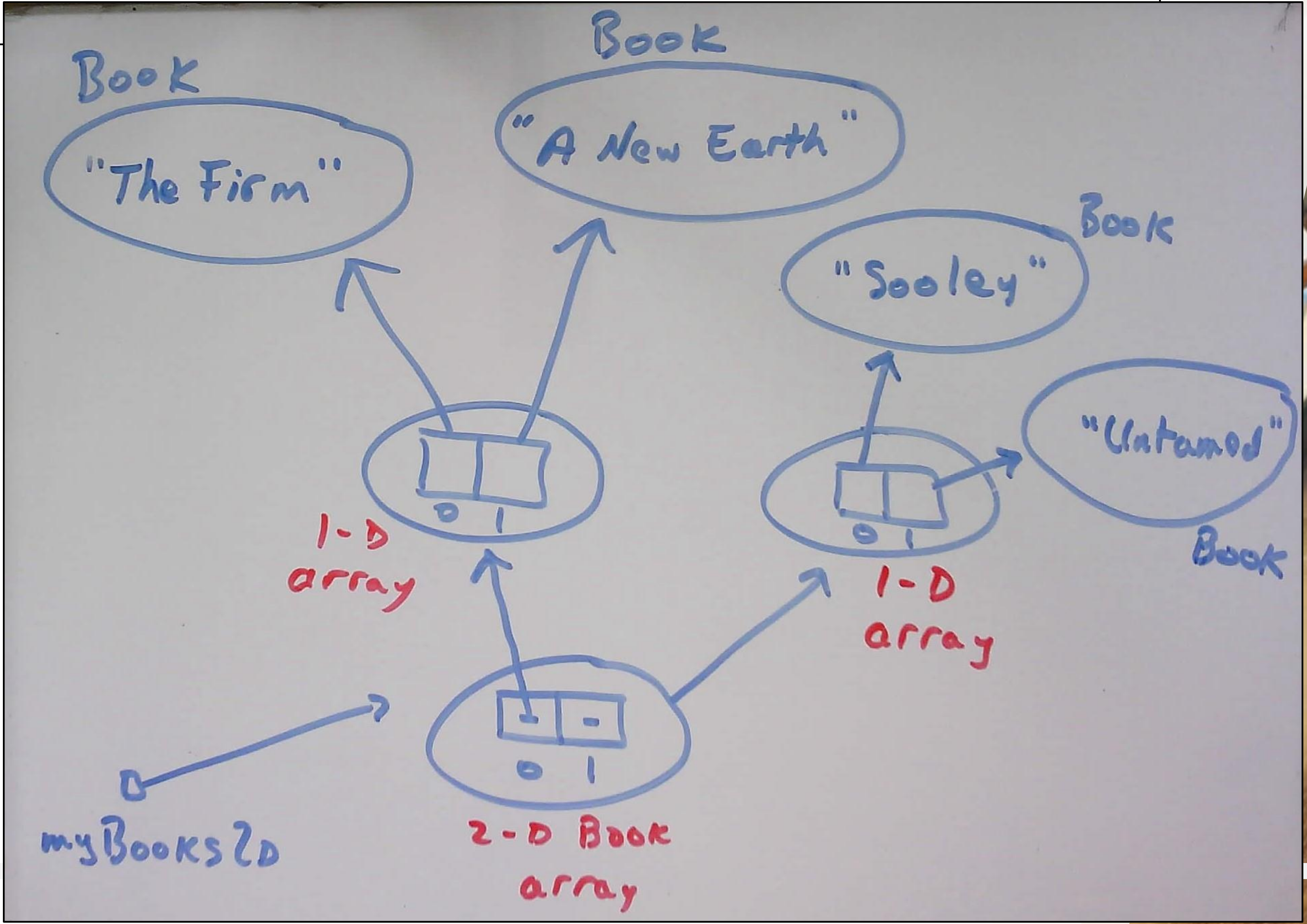
```
int [][]marks = { {3,1,7,6}, {2,3}, {6,5} };
```

- This creates an array of arrays i.e. an array, where each element refers to an array.
  - ➢ the *marks* array has a length of 3 – derived from the number of comma-separated items between the outer {}
  - ➢ each of the 3 elements in the *marks* array is a reference to an *int* array
    - the sizes of these 3 *int* arrays is based on the number of elements in the inner curly braces i.e. 4, 2 and 2 respectively.

```java
int [][]marks = { {3,1,7,6}, {2,3}, {6,5} };
```



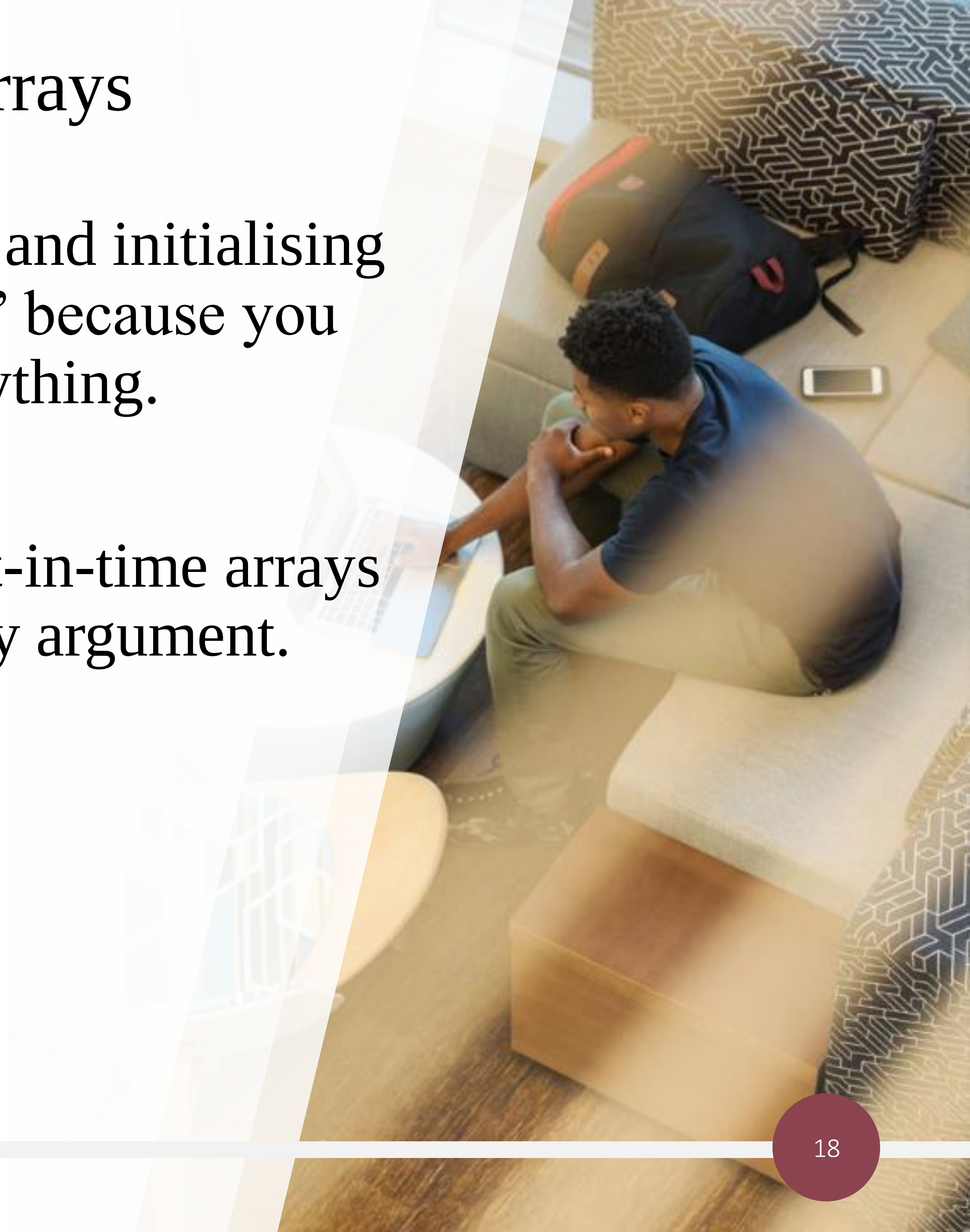
```java
Book[][] myBooks2D = {
    { new Book("The Firm"), new Book("A New Earth")},
    { new Book("Sooley"), new Book("Untamed")}
};
```



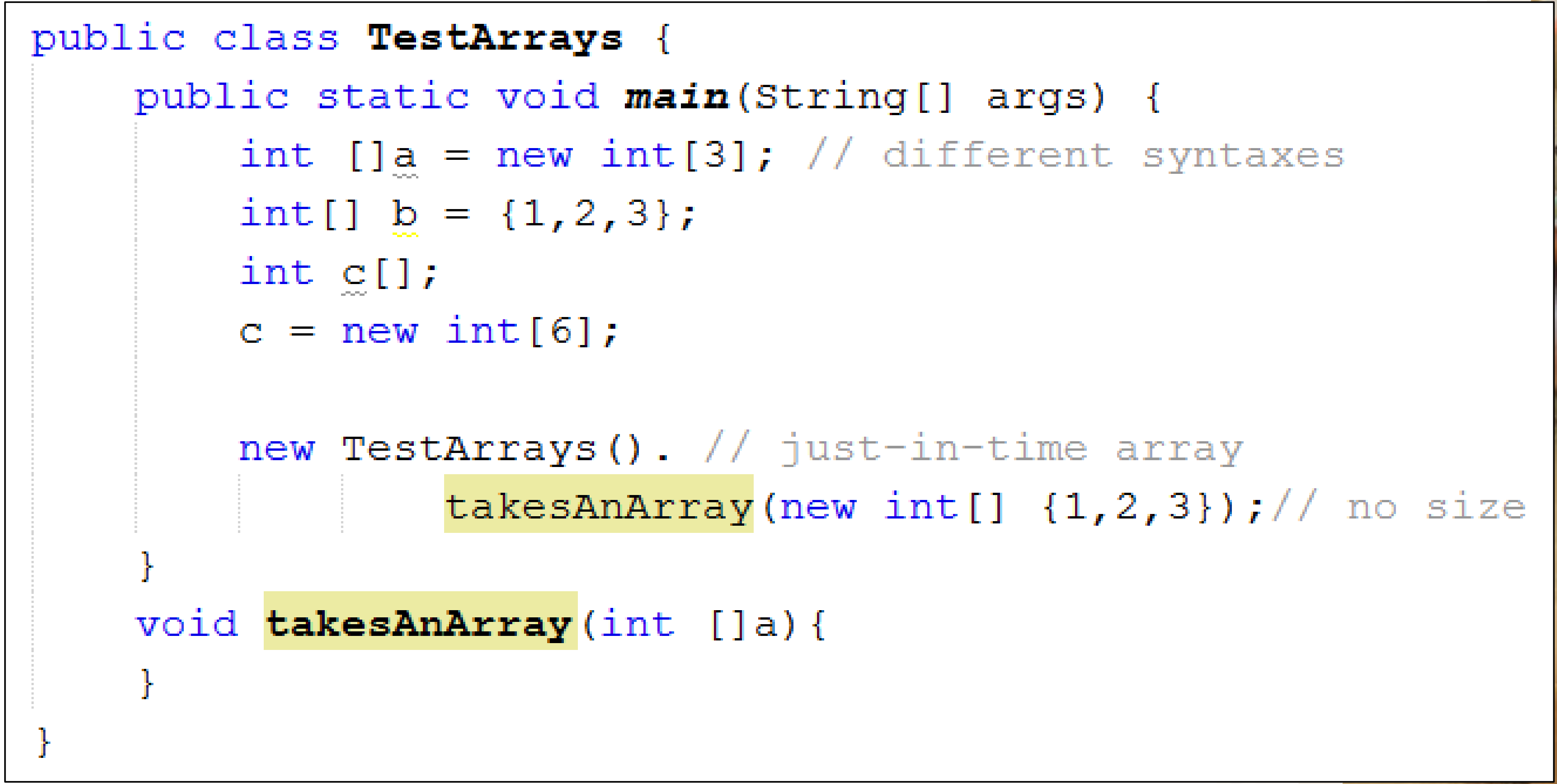
Anonymous (Just-in-time) Arrays

- Another syntax shortcut is used for creating and initialising an anonymous array. It is called “anonymous” because you don’t even need to assign the *new* array to anything.

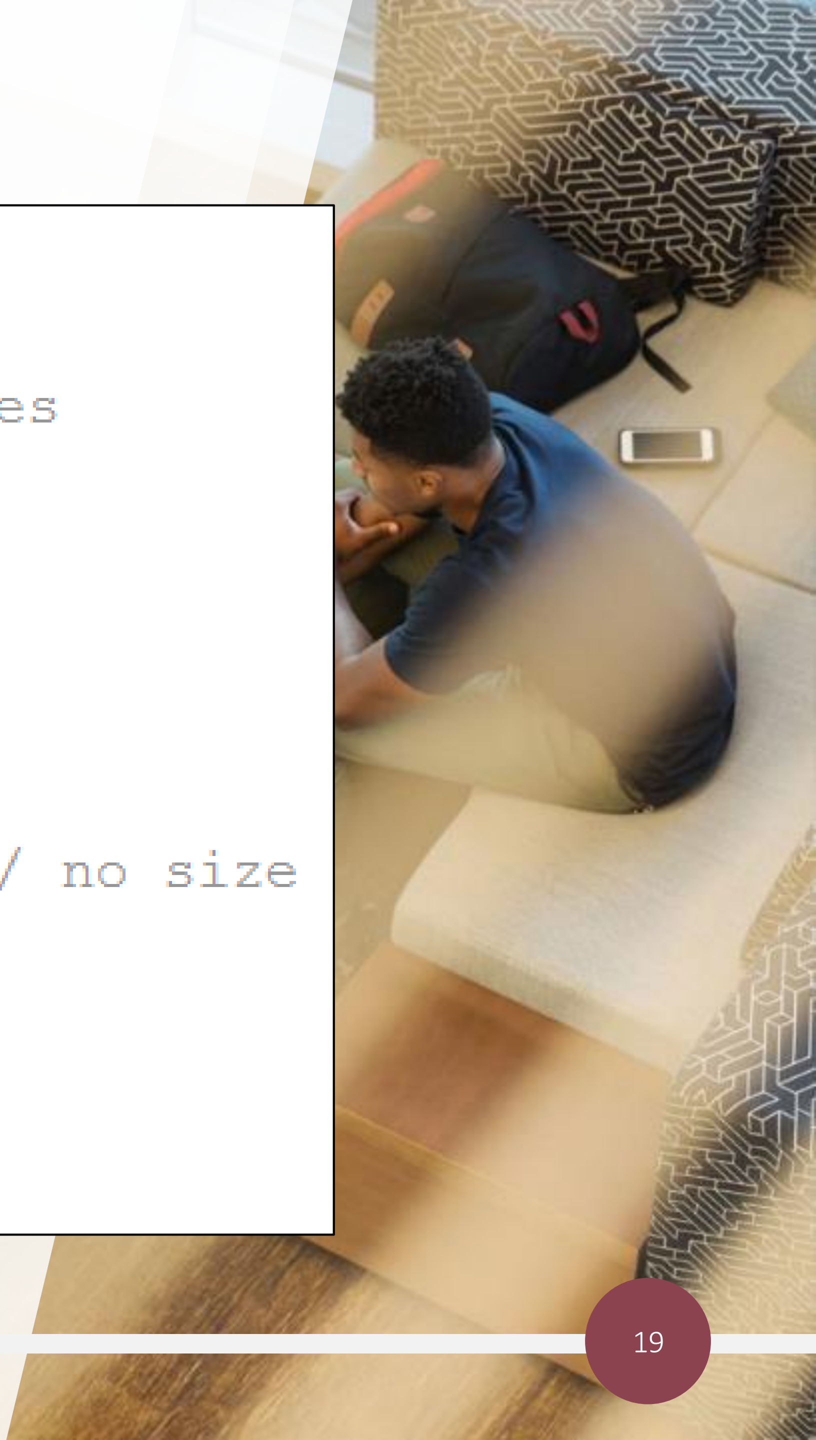
- They are useful for example, in creating just-in-time arrays when invoking a method that requires an array argument.



Anonymous (Just-in-time) Arrays

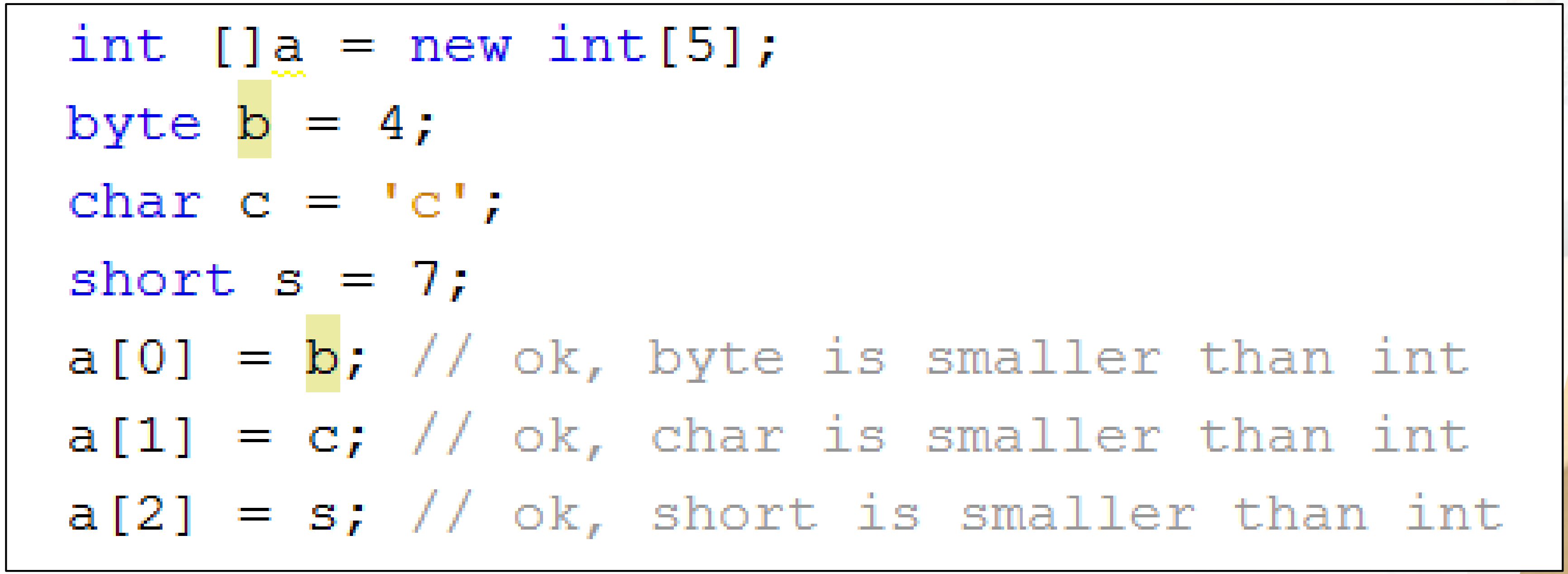


```java
public class TestArrays {
    public static void main(String[] args) {
        int []a = new int[3]; // different syntaxes
        int[] b = {1,2,3};
        int c[];
        c = new int[6];

        new TestArrays(). // just-in-time array
                takesAnArray(new int[] {1,2,3});// no size
    }
    void takesAnArray(int []a){
    }
}
```



Arrays of primitives

- Primitive arrays can accept any value that can be promoted implicitly to the declared type e.g. an *int* array can hold any value that can fit into a 32-bit variable.



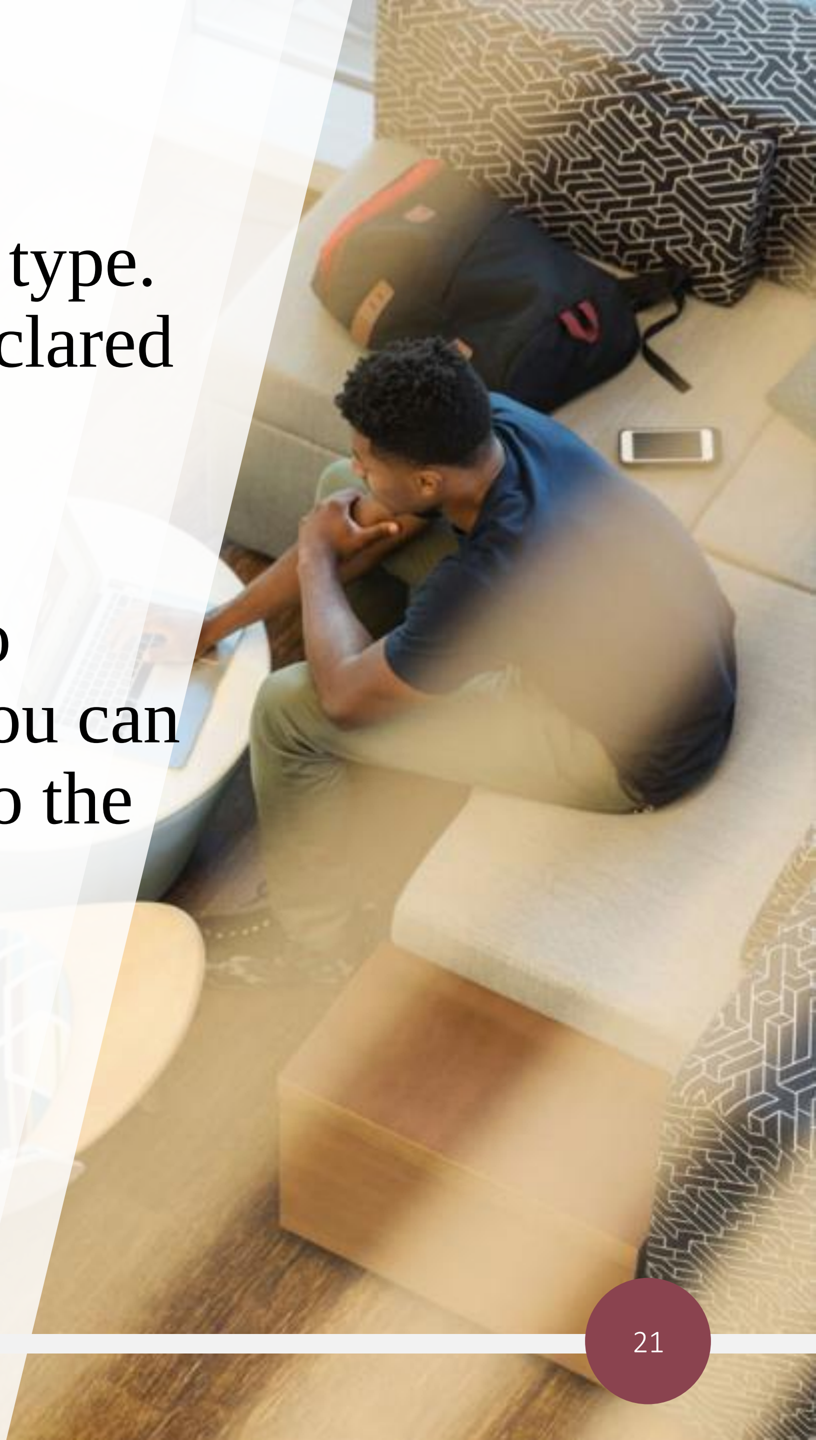
```
int []a = new int[5];
byte b = 4;
char c = 'c';
short s = 7;
a[0] = b; // ok, byte is smaller than int
a[1] = c; // ok, char is smaller than int
a[2] = s; // ok, short is smaller than int
```

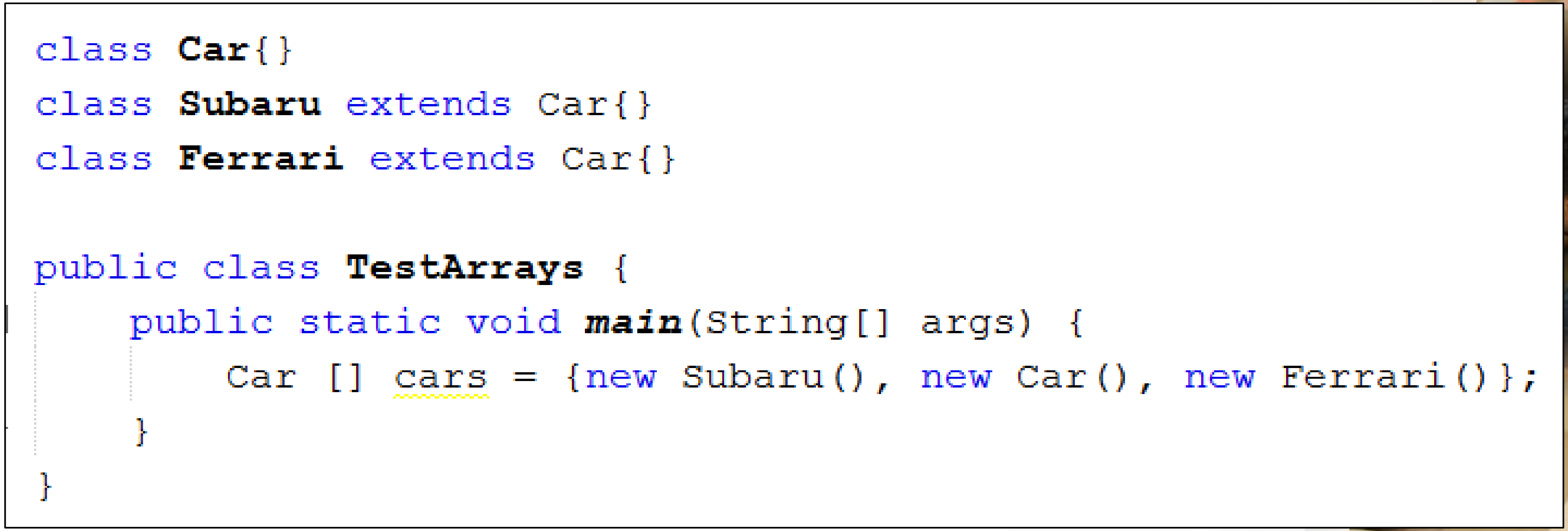
Arrays of object references

- The array elements are reference variables of the array type. Therefore, any object that passes the IS-A test for the declared array type can be assigned to an element of that array.

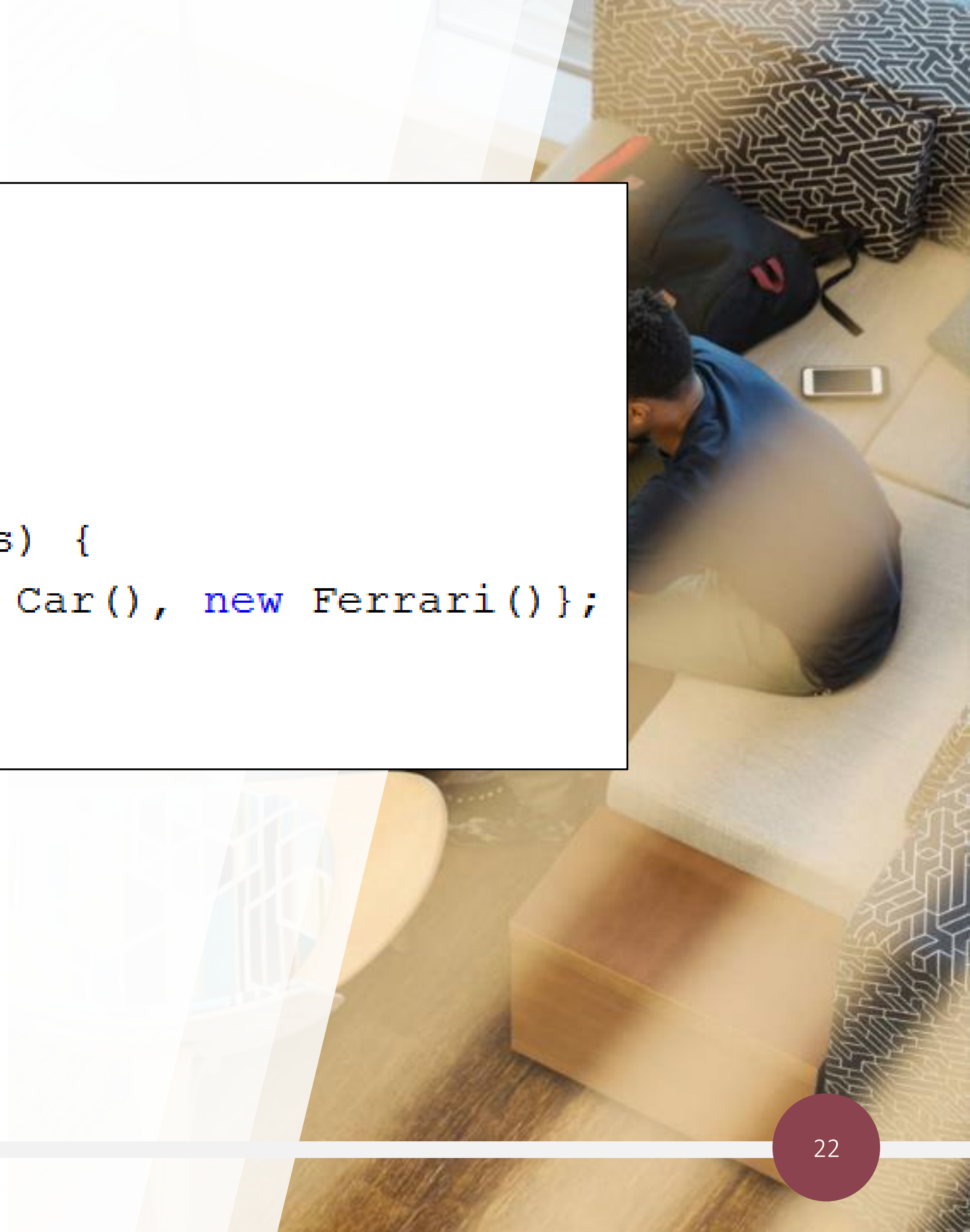
- If the declared type is a class, you can put references to objects of the declared type into the array. In addition, you can also put references to subclasses of the declared type into the array.

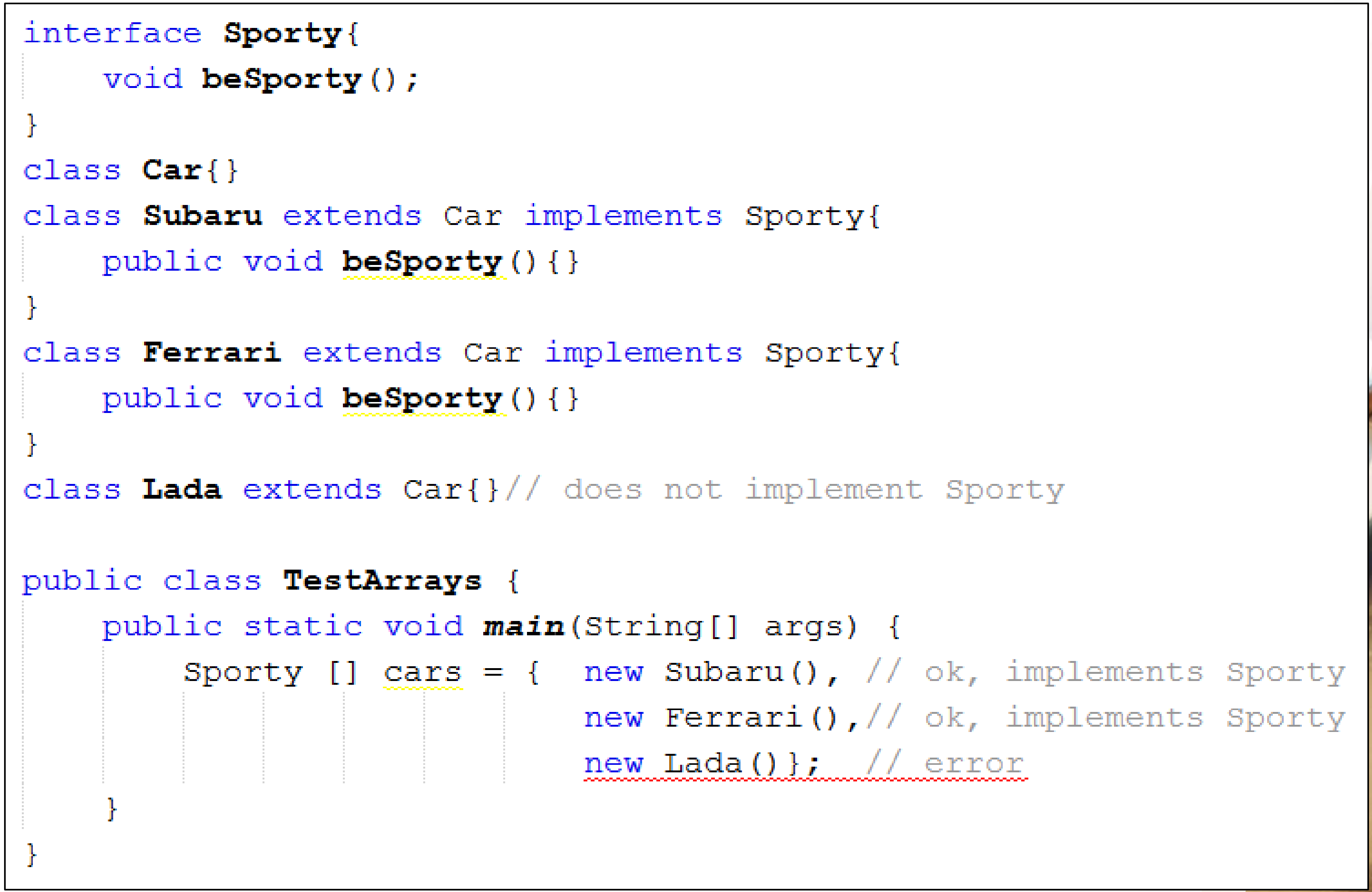
- If the array is declared as an interface type, the array elements can refer to any instance of any class that implements the declared interface.





```
class Car{}
class Subaru extends Car{}
class Ferrari extends Car{}

public class TestArrays {
    public static void main(String[] args) {
        Car [] cars = {new Subaru(), new Car(), new Ferrari()};
    }
}
```



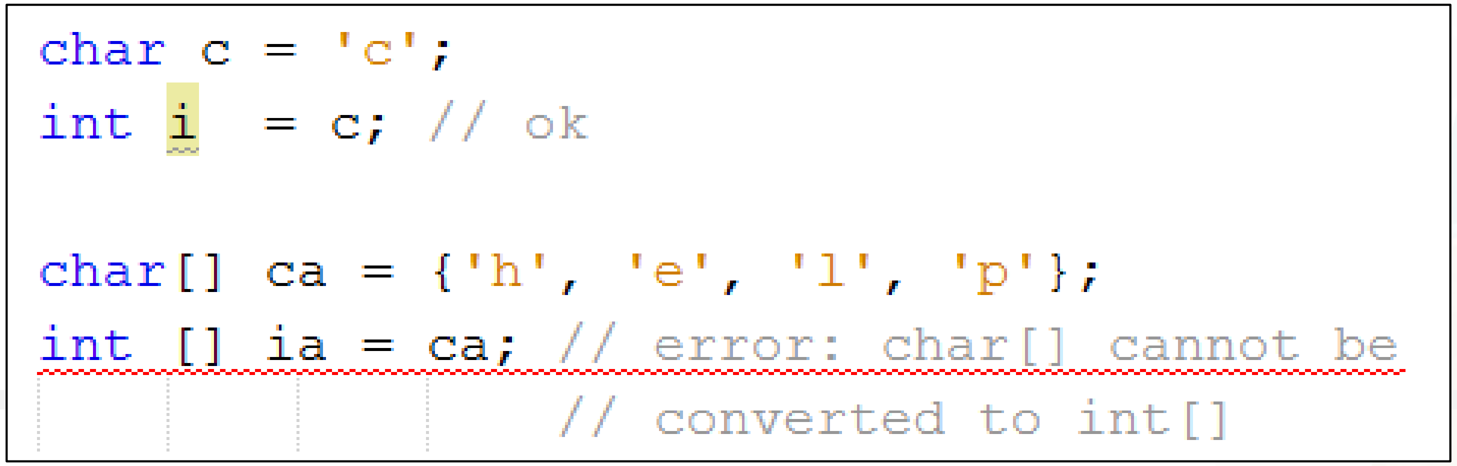


```java
interface Sporty{
    void beSporty();
}
class Car{}
class Subaru extends Car implements Sporty{
    public void beSporty(){}
}
class Ferrari extends Car implements Sporty{
    public void beSporty(){}
}
class Lada extends Car{}// does not implement Sporty

public class TestArrays {
    public static void main(String[] args) {
        Sporty [] cars = {  new Subaru(), // ok, implements Sporty
                            new Ferrari(),// ok, implements Sporty
                            new Lada()};  // error
    }
}
```

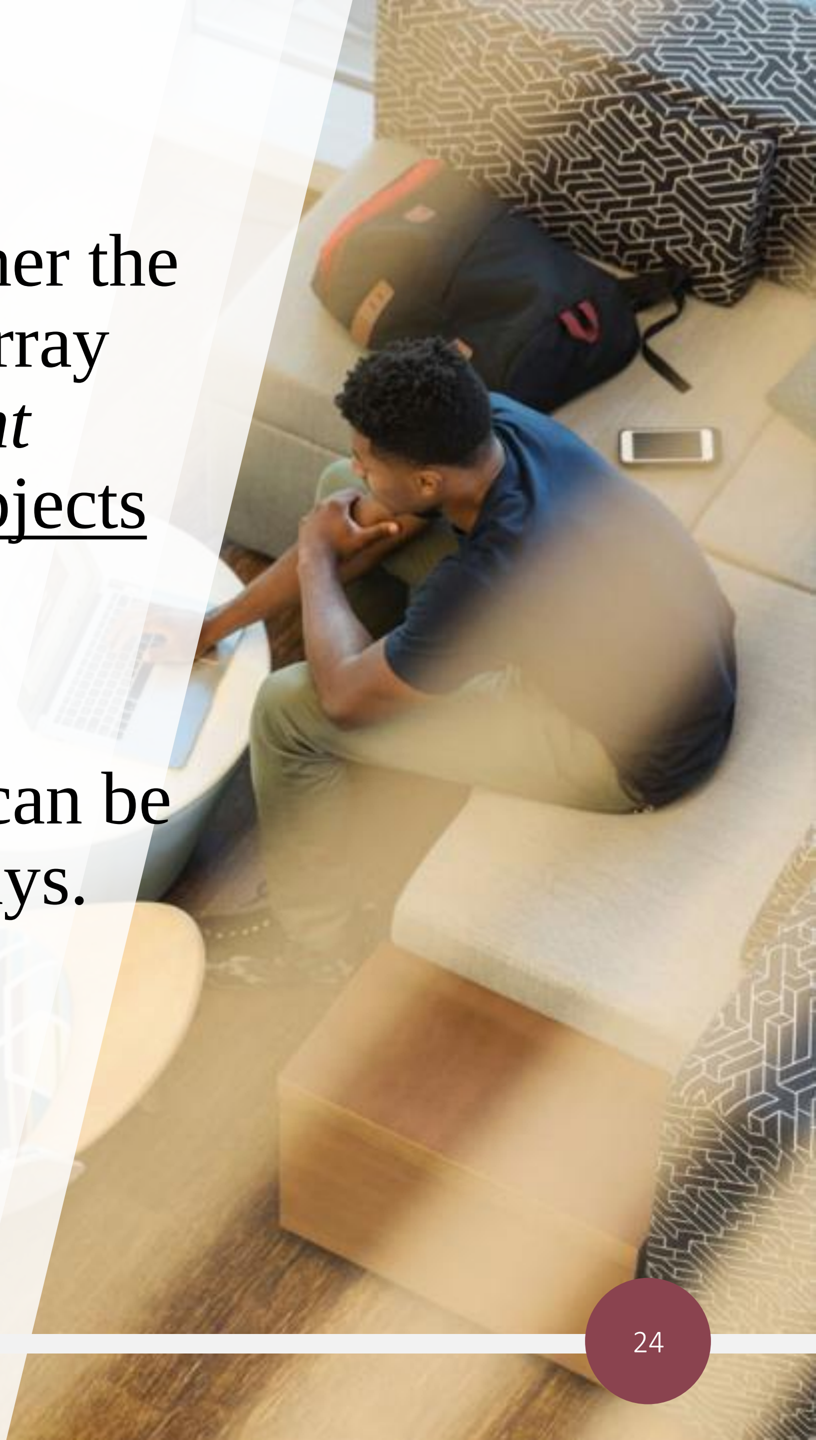
Array reference assignments

- We are not concerned here with array elements but rather the <u>reference to the array object </u>itself. For example, an *int* array reference cannot be assigned to anything that is not an *int* array, including an *int* value. Remember <u>all arrays are objects</u> so an *int* array reference cannot refer to an *int* primitive.

- Note that even though *byte*, *short* and *char* primitives can be promoted to an *int*, you CANNOT do the same with arrays.



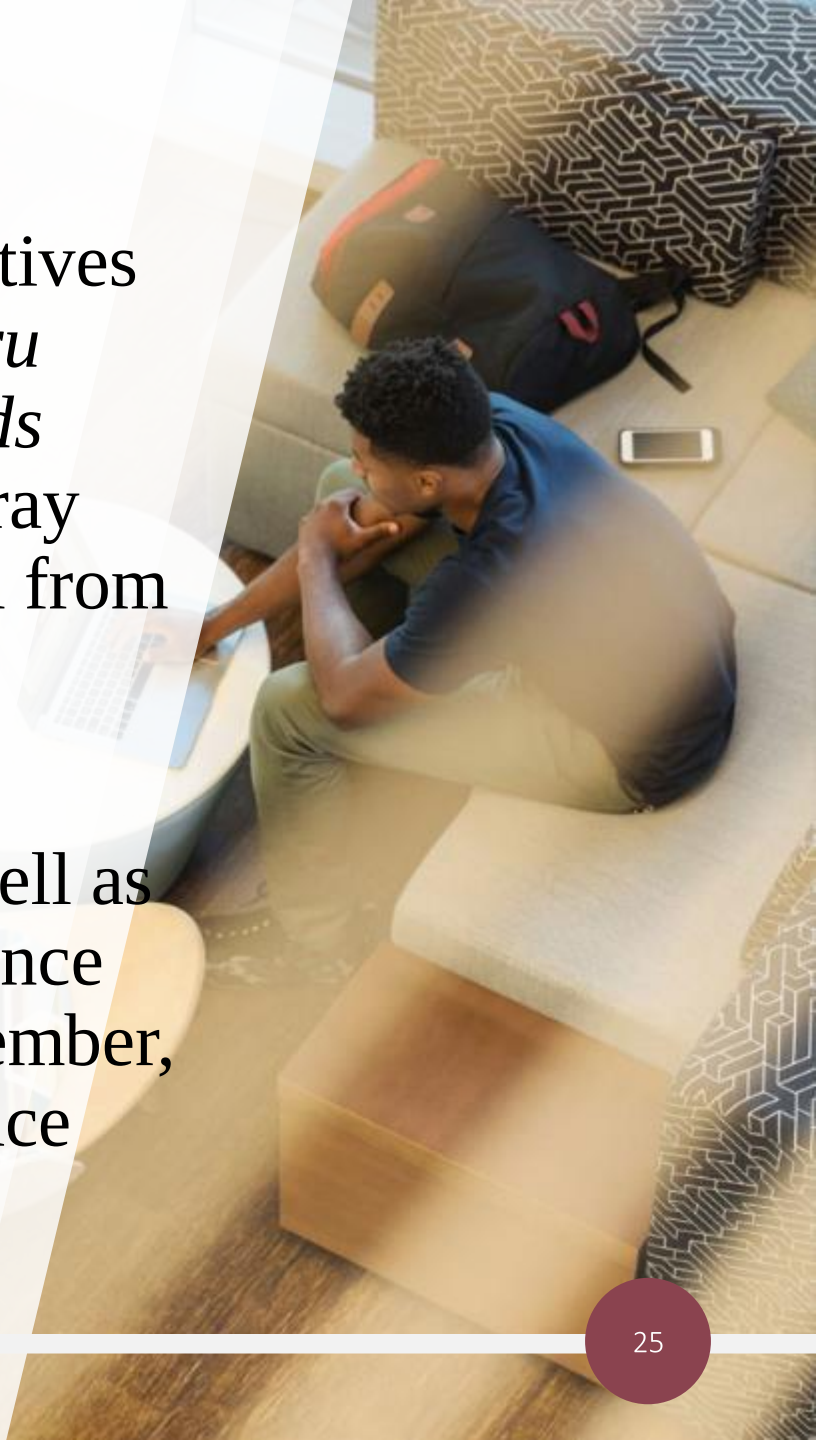
```
char c = 'c';
int i  = c; // ok

char[] ca = {'h', 'e', 'l', 'p'};
int [] ia = ca; // error: char[] cannot be
                // converted to int[]
```



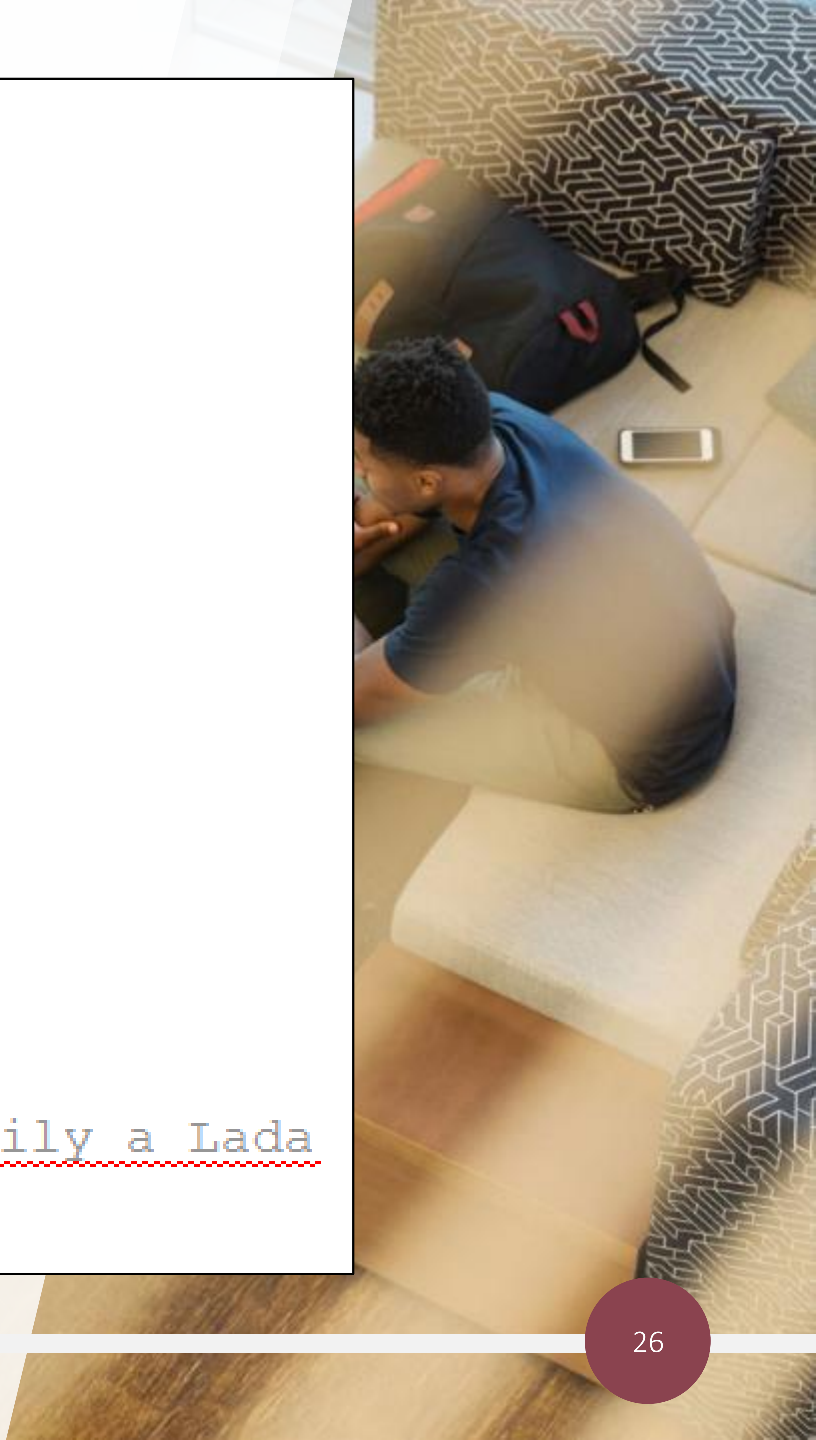
Array reference assignments

- Arrays that hold object references, as opposed to primitives are not as restrictive. Just as we were able to put a *Subaru* object reference into a *Car* array (because *Subaru extends Car*), you can assign an array of type *Honda* to a *Car* array reference variable. Apply the IS-A test to check the legal from the illegal.

- The rules for array assignment apply to interfaces as well as classes. An array declared as an interface type can reference an array of any type that implements the interface. Remember, any object from a class implementing a particular interface will pass the IS-A test (*instanceof*) test for that interface.

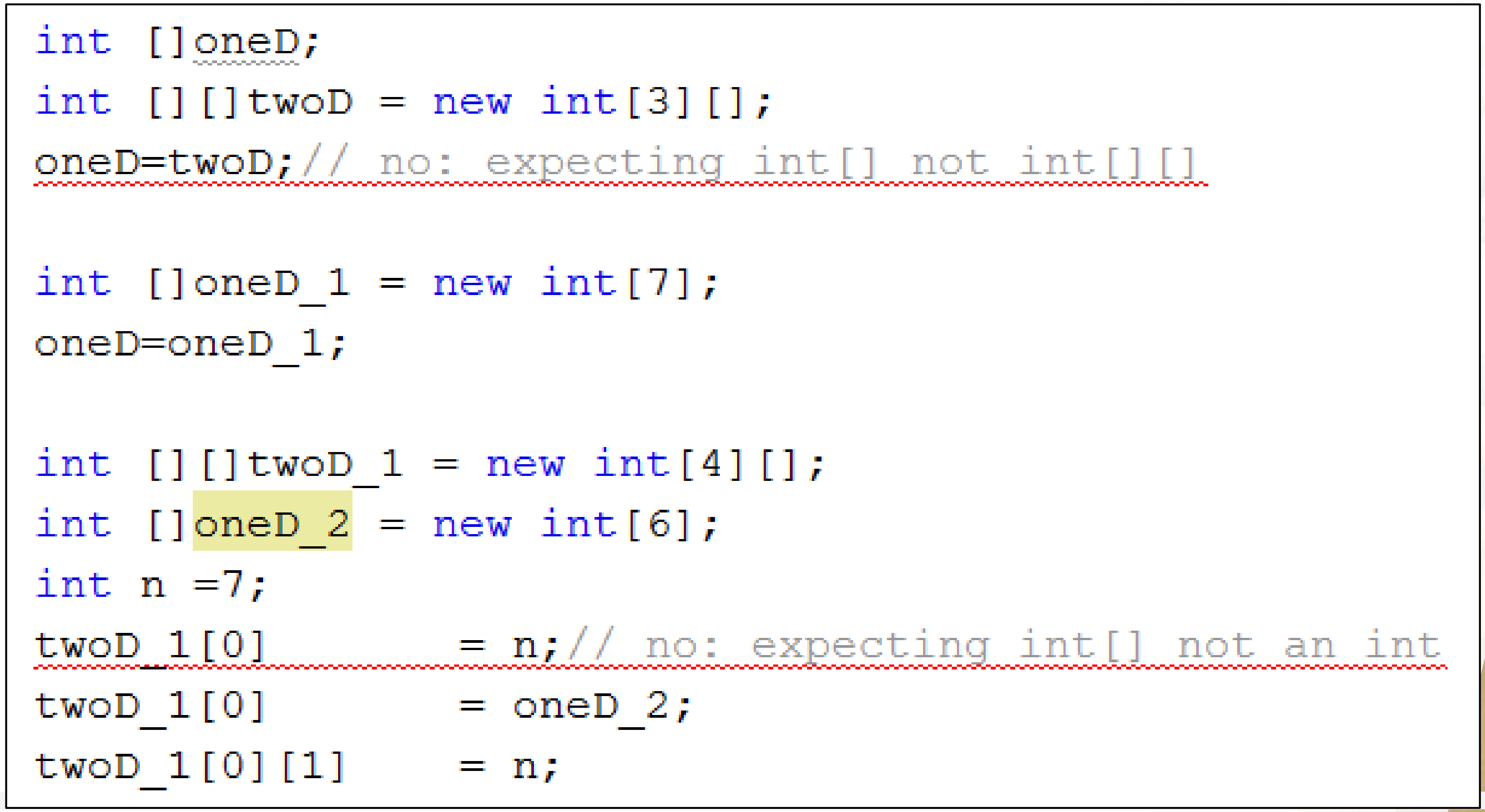


```java
// 1. Array of object references
Car[] cars = new Car[3];
Subaru[] subaruCars = new Subaru[3];
cars = subaruCars;// ok, Subaru IS-A Car
Chair[] chairs = new Chair[3];
cars = chairs;  // error: Chair fails IS-A test

// 2. Array of interface references
Sporty[] sportyThings = new Sporty[3];
sportyThings = subaruCars; // ok, Subaru IS-A Sporty

// 3. Assignment will not work the other way
Car[] moreCars = {new Ferrari(), new Car()};
Lada[] ladaCars = {new Lada()};
ladaCars = moreCars; // error: a Car is not necessarily a Lada
ladaCars = (Lada[])moreCars; // ClassCastException
```

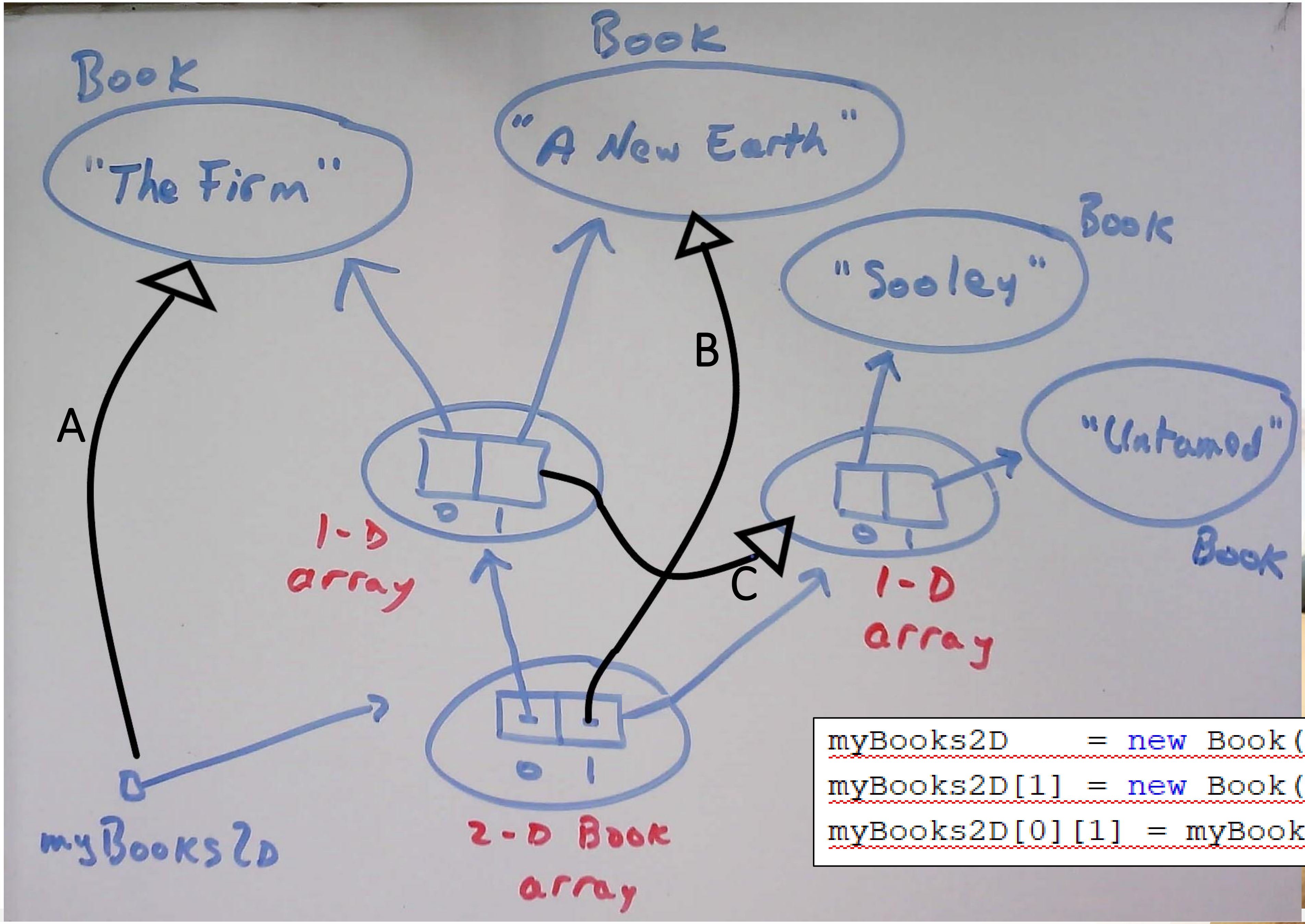


Array reference assignments

- When assigning an array to a previously declared array reference, the dimensions must match.



```
int []oneD;
int [][]twoD = new int[3][];
oneD=twoD;// no: expecting int[] not int[][]

int []oneD_1 = new int[7];
oneD=oneD_1;

int [][]twoD_1 = new int[4][];
int []oneD_2 = new int[6];
int n =7;
twoD_1[0]        = n;// no: expecting int[] not an int
twoD_1[0]        = oneD_2;
twoD_1[0][1]     = n;
```



```
myBooks2D       = new Book("The Firm"); // A
myBooks2D[1]  = new Book("A New Earth");// B
myBooks2D[0][1]  = myBooks2D[1];// C
```