More on Objects

A byte size lesson in Java programming.

More on Objects

- Now that you have learned about objects, you can explore **more advanced features** of the Java programming language.

Arrays of Objects and 2D Arrays

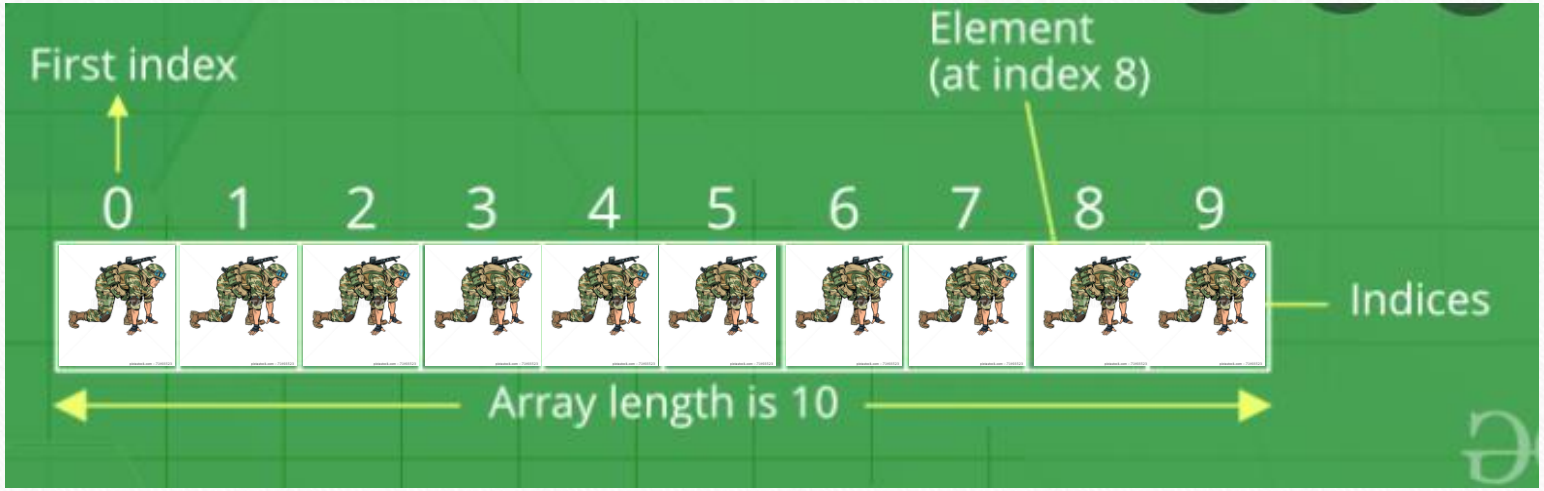
We learned about arrays last year, and all that we learned still applies. But there is more…

Array of Objects

- It is perfectly possible and often necessary to create arrays that store complex types like objects.
- You will find that declaring an array of objects is not much different to creating an array of simple types.

```
Soldier[] soldiers = new Soldier[10];
```

Array of Soldier



Assigning an array value

```
// create an array
Soldier[] soldiers = new Soldier[10];

// create an object instance and store it in array
soldiers[0] = new Soldier();

// works also
Soldier john = new Soldier();
soldiers[1] = john;
```

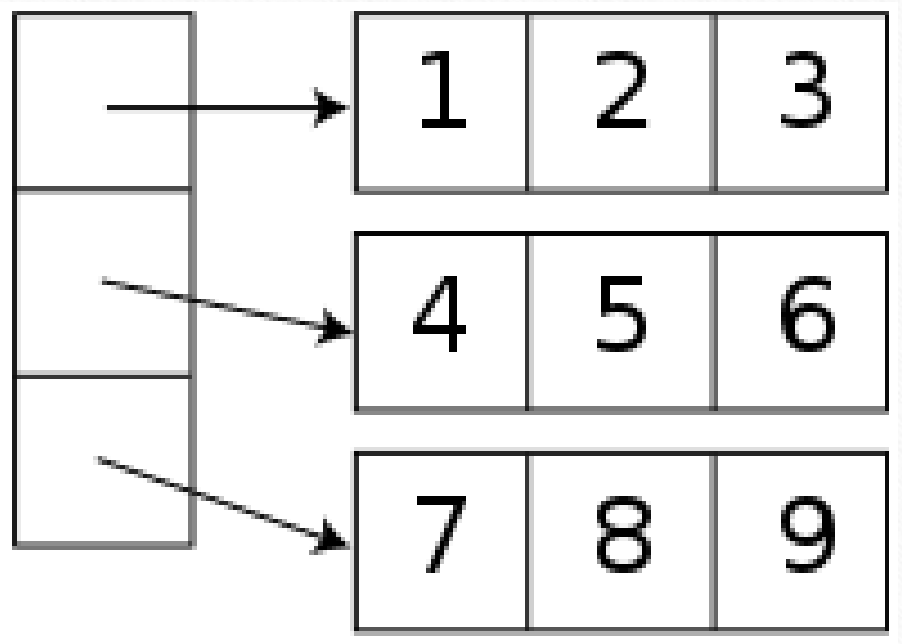
Let's test your understanding!

- Complete the following code…

```
                  accounts = new BankAccount[5];

System.out.println("Second account: " + accounts   .number);

System.out.println("          account: " + accounts[3].number);

// create and store the last account object in accounts
                                                              
```

Multidimensional Arrays

- A multidimensional array is an array of arrays. Each array element in a multidimensional array would be an array!
- We can create two dimensional arrays to organise data in a grid/table/matrix.



Initialising two dimensional arrays

- To create a 2D array we need to specify the size of each dimension.
- In this case, the number of rows and columns. The below can hold a maximum of 9 elements.

```
// 3 rows and 3 columns
int[][] a = new int[3][3];
```

	Column 1	Column 2	Column 3
Row 1	a[0][0]	a[0][1]	a[0][2]
Row 2	a[1][0]	a[1][1]	a[1][2]
Row 3	a[2][0]	a[2][1]	a[2][2]

Initialising two dimensional arrays

- This is an example of how we can initialise a 2D array of numbers.

```
// create an array
int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

Let's test your understanding!

- What is the output of the following?

```
int[][] a = {
    {1, 2, 3},
    {4, 5, 6, 7}
    {7}
};

for(int row = 0; row < 3; row++) {
    System.out.println("Row " + row + " has " + a[row].length + " cols");
}
```

Looping through 2D arrays

- As with 1D arrays, it is very common to loop through them and do something with each element:

```
int[][] matrix = { {1, 2, 3}, {4, 5, 6,}, {7, 8, 9} };

for(int row = 0; row < 3; row++) {
    for(int col = 0; col < 3; col++) {
        int number = matrix[row][col];
        System.out.print(number + “, ”);
    }
}
// 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Random Class

This is a Java utilities class in **java.util.Random**.

What is the Random Class?

- Java comes with certain classes with methods that programmers can use to add more functionality to their programs.
- The **Random** class provides several methods to help us generate random numbers.
- Using random numbers can add an element of realism to our programs and make them much more usable.

Creating an instance of Random

- This is an example of how we can create an instance of the Random class.

```
// create an instance of the Random class
Random generator = new Random();
```

- This requires an **import** at the top of your program file.

```
import java.util.Random;
```

Generate Random Whole Numbers

- This is an example of how we can get a random number between 0 to 10 (exclusive).

```
// we are storing the number generated into random number
int randomNumber = generator.nextInt(10);
```

Let's test your understanding

- What **method** from Random could we call to store a value in isHeads?

```
// simulate a coin flip using the Random instance
boolean isHeads = generator.                    ;
```

Useful methods in the Random Class

- nextInt()
- nextDouble()
- nextBoolean()
- nextFloat()

Example usage

- Here we are using the Random class to get a random array index so that we can randomly select a PastPaper object.

```
// use the random object
Random random = new Random();
int randomIndex = random.nextInt(index);
PastPaper loadPastPaper = myLibrary[randomIndex];
```

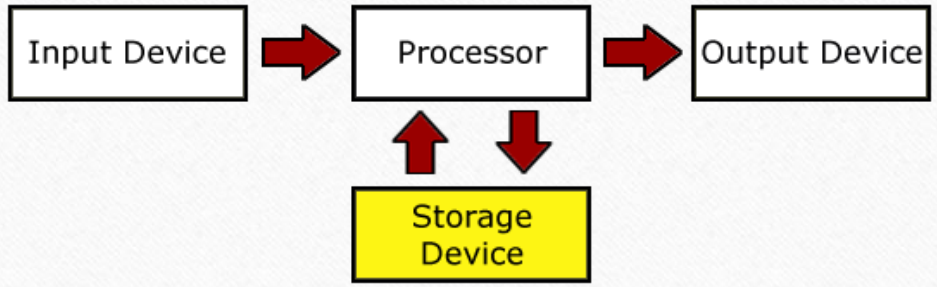
Java File Handling

There is a suite of Java classes under their **“io”** package.

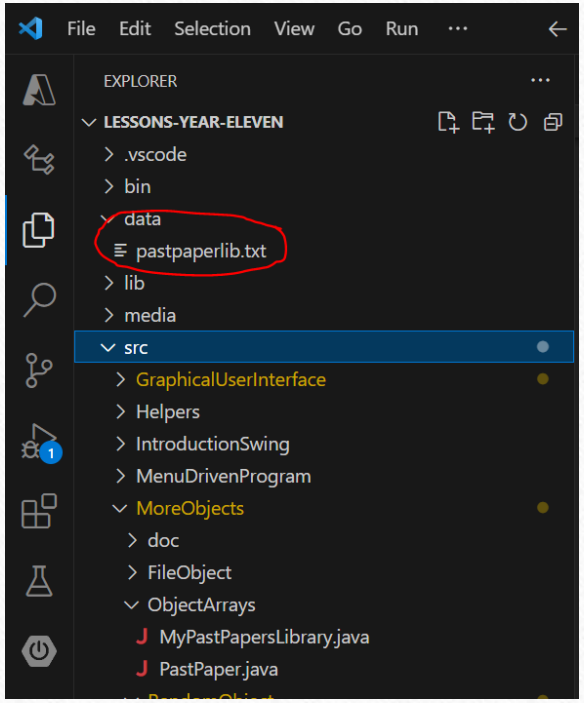
We will look at the **java.io.File** class.

Why is it important?

- File Handling is important in applications.
- As discussed in other topics, saving the program state in secondary storage is desirable.
- Java allows us to save data in a text file so the user can pick up from where he left off when the program is used multiple times.



Create a File



- In your project folder, create a new folder called **"data"** to have a place for files created by the program.
- Ensure the folder is at the same level as the **"src"** folder.

Create a File

- Create an instance of the File class and specify the relative path inside the (). Here is an example:

```
File dataFile = new File("data/filename.txt");
```

- This requires an **import** at the top of your program file.

```
import java.io.File;
```

Write to a File

- We need to create an instance of the **FileWriter** class (from java.io) and pass use our **File** instance.

```
FileWriter writer = new FileWriter(dataFile);
```

- We can then use methods like **write()** to put data on the file.
- However, this line of code can cause a **run-time error**. Java calls these, **exceptions**. Use a **try…catch** statement when working with these objects to prevent your program from crashing.

Example Usage

```
try {
   File dataFile = new File("data/pastpaperlib.txt");
   FileWriter writer = new FileWriter(dataFile);
   writer.write("Write your data.");
   writer.close();
} catch (IOException e) {
    System.out.println("There was an error");
}
```

Read from a File

- We need to create an instance of the **Scanner** class (from java.util) and pass use our **File** instance.

```
Scanner reader = new Scanner(dataFile);
```

- We can then use methods like **hasNextLine()** and **nextLine()** to read all our data on the file.
- This line of code can also cause a **run-time error** so use a **try…catch** statement when working with these objects to prevent your program from crashing.

Example Usage

```
try {
   File dataFile = new File("data/pastpaperlib.txt");
   Scanner reader = new Scanner(dataFile);
   while (reader.hasNextLine()) {
       String nextLine = reader.nextLine()
       // do something with nextLine
    }
   reader.close();
} catch (IOException e) {
     System.out.println("There was an error");
}
```

Conclusion

- We have shown four ways in which you can make use of objects: storing objects in an array, creating 2D arrays, the Random class, and file handling.
- Any **one** of these techniques can be applied to your coursework and earn you a lot of marks.