# Expeditors Backend Academy Labs

## Introduction

This document contains the labs for the Expeditors Backend Academy.

The instructions are split up into **Classwork** and **Homework**. The Classwork we will do in class together. You should do the Homework in the **Homework** module. Create a new package for each week's homework if it makes sense. I will go through an example before we start.

## Week 2

### Classwork

1. Arrays and Collections
2. Methods
3. Start to write classes

### Homework

#### Objectives

1. Arrays, Collections, Methods.
2. Write a program to pick a student at random from a list of students
    a. Create an array with the names of all the students in the class.
    b. Create a method that returns a student chosen randomly from your student array.
        1. Use *ThreadLocalRandom.current().nextInt(limit)* to create a random number. e.g.

            int rand = ThreadLocalRandom.current().nextInt(10) will create a random number between 0 and 10 exclusive.
            Check out the API documentation for other ways to create random numbers.
    c. Create a client application that calls your method a few times and prints out the results.
3. Learn how to create methods
4. Create an Adopter class as the first step to creating a Pet adoption application.

**Tasks**

1. Arrays 1.
   a. Create an array of 10 ints
   b. Initialize each element of the array to the square of it's index. e.g

      element[0] = 0
      element[1] = 1
      element[2] = 4
      etc.

   c. Write a JUnit test to make sure your code works. e.g. assert that element[6] == 36
2. Arrays and Methods. .
   a. Write a method called *createArray*.
      1. It should take two arguments called *size* and *limit*.
      2. It should create and return an *int* array of length '*size'* and initialize it with random integers between 0 and '*limit'.*

         See below for ways to generate a random number.
   b. Write a main method and an appropriate test for your code.
3. Instead of the Array of students, use a List of students.
4. Adopter class. This is going to be the start of a long running exercise to build an application for a pet adoption service. You should put all code for this exercise into it's own root package e.g. *expeditors.backend.adoption.* Create sub-packages under the root as necessary.
   The Adoption Service should keep track of people who have adopted a pet. It should handle information about the adopter and the pet they have adopted. For now the service deals with only three types of pets: Cats, Dogs and Turtles.
5. In this iteration you will create an Adopter domain class. The services etc. will follow later.
   a. You should keep at least the following information: with at least the following properties
      1. **id** of the adopter
      2. **name** of the adopter
      3. **phone number of the adopter**
      4. **date of adoption** of type LocalDate – see below on how to create LocalDate objects
      5. The type of pet adopted

6. The name of the pet if any
7. The breed of the pet if known (siamese, poodle, etc)

   You can add other properties as you see fit

b. Make sure you encapsulate your properties appropriately
   1. private variables
   2. getters and setters
c. Create an application class with a main method where you create 2 instances of your class, populate them with appropriate values, and then print them out. Note how you have to interact with the class. Is it cumbersome to use?

d. Some ways to work with LocalDate. Check API docs for more examples.

```
//today:
LocalDate now = LocalDate.now();
//dob of 04/17/1956:
LocalDate dob = LocalDate.of(1924, 10, 17);
//Years between dob and today:
long age = dob.until(LocalDate.now(), ChronoUnit.YEARS);

assertEquals(99, age);
```