Java Fundamentals Labs

Labs

- This document contains labs for the Java Fundamentals course.
- The first lab imports the github repository into your IDE.
- **For the remaining labs, remember to do your work in the StudentWork module.**

Lab 1

- Import the course project into your IDE. We will do this part collectively in class.
- Write a “hello world” program to make sure everything is in place.

Lab 2

- **Remember to do your work in the StudentWork module.**
- Control structures
  - Write a program that goes in a loop from -500 to +500 and prints out all numbers that are divisible by either 3 or 7. Do this with a **for** loop and a **while** loop

Lab 2.1 – Arrays

- **Remember to do your work in the StudentWork module.**
- Create an array of 10 **int**s.
- Initialize each element of the array to the square of it’s index
  - element[0] = 0
  - element[1] = 1
  - element[2] = 4
  - etc.
- Use the Java loop construct which is most appropriate for this use case.
- Print out the elements of the array, along with their indices.

Lab 3 – Methods And Arrays

- **Remember to do your work in the StudentWork module.**
- Methods and Arrays
  - Write a method called **createArray** that takes two arguments, called **size** and **limit.** It should create and return an **int** array of length **size**, initialized with random integers between zero and **limit.**
  - int rand = **ThreadLocalRandom.current().nextInt(10)**
    - creates a random integer between 0 and 10
  - Check out the documentation for more variations on how to create random numbers.
  - Remember to make your method static for now.
  - Write an appropriate test for your method.

Lab 4a – Classes

- **Remember to do your work in the StudentWork module.**
- Create a class called Student. The class should have at least the following properties
  - A first name and a last name
  - A date of birth of type **LocalDate**
    - more info on LocalDate on the next slide
  - A status which can be one of
    - Fulltime
    - Parttime
    - Hibernating
- You can add any other properties you want.
- Write Unit tests for your class

Lab 4a contd. - LocalDate

- **LocalDate** is one of several Java classes used to work with dates and times
  - Represents a date without a time zone
- Creation
  - LocalDate ld = LocalDate.now() //The date today
  - LocalDate sixOct1922 = LocalDate.of(1922, 10, 6)
- Comparison
  - long yearsOld = sixOct1922.until(LocalDate.now(), ChronoUnit.YEARS)

Lab 4b – Classes/Encapsulation

- Make sure you Student class properly encapsulates its variables.
  - private variables
  - getters and setters
- Unit tests

Lab 4c – Classes/Constructors

- Constructors
  - Provide a constructor that takes a first name, last name and date of birth as arguments, and another that takes arguments for all fields. Create any other constructors you think would be useful.
  - Make sure your objects are always in a valid state, no matter which constructor is called.
  - Write Unit Tests for your Constructors.

Lab 4d – Classes/Methods

- Objects etc.
  - Create a method that returns the *formal* name of a student
    - lastname, firstname
  - Create a method called isActive that checks whether a student is active or not.
    - returns true if the Status is Fulltime or Parttime
    - returns false if the Status is Hibernating
  - Create a method called **getCurrentInfo** that returns the formal name and whether the student is active or not.
  - Create an application class that creates a few Students, and prints out the their current status.
  - Unit tests.

Lab 5 – id

- Add an **id** property of type int to the Student class.
- Create a way to automatically assign a unique id to every new Student object you create . (Hint – use **static)**
- Create an **array** of 4 Students and make sure they have unique ids.
- Create a method to be able to retrieve the next id that will be used.
- Change the **getCurrentInfo** method to also return the **id.**

Lab 6 - Inheritance

- Create a subclass of Student called **VisitingStudent**.
- Resolve constructor requirements properly.
- Add a field to VisitingStudent called **homeUniversity** that will hold the student's home university.
- Modify the **currentInfo** *behaviour* so that for Visiting Students it also indicates the home University.
- In an application class, create an array of 2 Student and 2 VisitingStudent object. Print out the current info of each object.
- Add a test to check that the new **currentInfo** is working correctly.

Lab 7 - Interfaces

- **Remember to do your work in the StudentWork module.**
- Slightly different instructions for this one
- The premise of the lab is that you have to create an application that interacts with a collection of Sensors to collect data about the environment, e.g. a Temperature Sensor, a Humidity Sensor etc.
- Your application needs to periodically ask each of the sensors in it’s collection to take a reading.
- This will require the use of an Interface.
- Look in **src/test/java/org/ttl/javafundas/labs/interfaces.**
- The class **RunInterfaceLab** has a Junit test and some TODO instructions on what you need to do.
- Uncommenting the code will produce a bunch of errors which you will have to fix.

Lab 8 - Generics

- **Remember to do your work in the StudentWork module.**
- Starter code is in the Unit test in **.../labs/generics**
- Use the Mixer class in the MixerLab project as your starting point.
- Refactor it so it becomes type safe
  - The frequency map should contain <String, Integer> as its map types.
  - The List should contain <String> as its element types.
- You should get rid of all the “raw type” warnings

Lab 8.1 – Generics optional

- The getFrequencyMap method in the Mixer class is very inefficient as written.
- Why is that?
- Rewrite it to be able to create the Map in one iteration through the args array.

Lab 9 - Enums

- Change your Student class to use an **Enum** for the status property.
- You will have to make changes to the signatures of all methods which deal with status.
  - Constructors
  - property methods

Lab 10 - Collections

- Change your code to use a List rather than an array. Pay attention to your use of generics.
- Write a method to create a map of Student by student id from your List of students.

Lab 11 - Exceptions

- You are going to add Exception handling code to your Student class constructor.
- If an attempt is made to create a Student with an age less than 20 years, the application should throw an **InvalidStudentException.**
  - The Exception should have a message explaining the error
- Which means that you are first going to have to create a new Exception class called InvalidStudentException.
- Your tests should test creation of valid and invalid students.
- Implement the Exception first as a checked exception, and then as an unchecked exception to see the different impacts they have on your code.

The End