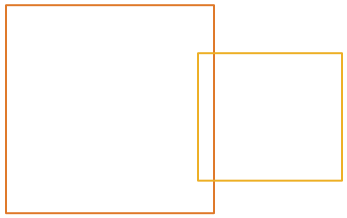
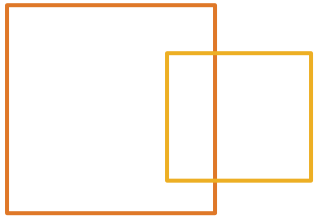
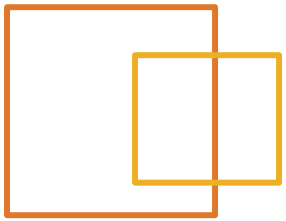
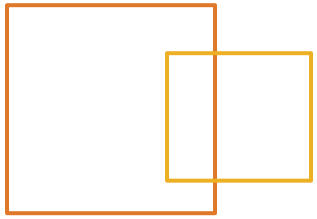
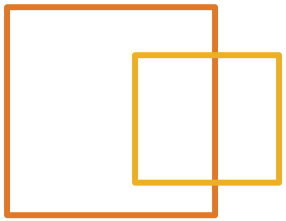





Fast Track to Java Labs

Delivered by DevelopIntelligence

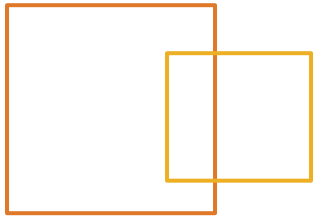
Lab 1

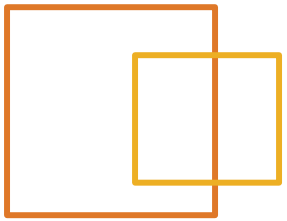




- Create a workspace for the course
- Write a “hello world” program to make sure everything is in place

Lab 2



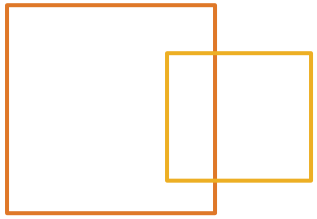


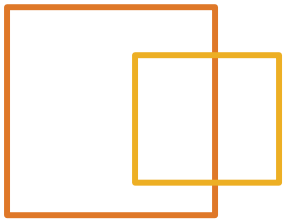
- Write a program that counts in a loop so that the output starts at 5,and then counts by 3 to 20. Do this using a for loop, and with a while loop.
- Using methods:
  - Write a program that declares and creates two arrays of integers, of different sizes. Initialize both arrays with random integers between 0 and 5000.
  - Create a method called **addToAll** that takes an integer array and an int as arguments. The method should add the **int** argument to each element of the array argument.
  - Remember to make your method **static**.
  - Call the method with each of your arrays, and print out their contents after the call to make sure the method works.

Lab 2 contd.

- Using Arrays:
  - Write a program that declares an array of Strings. Declare a numeric value "language", and two constants, ENGLISH, and SPANISH with the values 0, and 1, one respectively.
  - Use the value of "language" to control a switch/case statement that initializes the elements of the array with the names of the numbers, from "zero" to "ten", or in Spanish (“uno”, “dos”, “tres”, “quatro”, “cinco”, “seis”, “siete”, “ocho”, “nueve”, “diez”)
  - Iterate over the loop printing output that looks like: *The name of 5 is five*

Lab 3

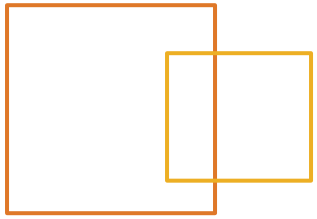


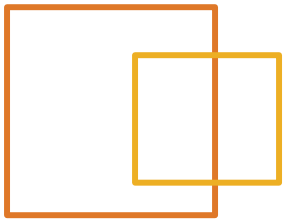


Simple Objects

- Create a class that describes a Person. Pay attention to Java's coding conventions. The class should have fields for first name, last name, gender and birthdate (day and month).
- Provide a constructor that initializes first name, last name, and gender, and a second that initializes all the fields.
- Create a method that gets the person's full name (first and last name concatenated), and another method that returns the prefix "Mr" or "Ms" depending on gender.

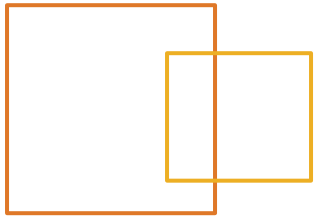
Lab 3

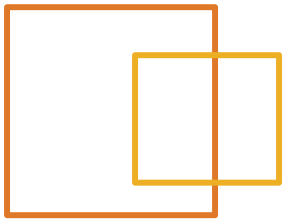




- Simple Objects (contd)
  - Create a third method that gets the "formal address" of the person by concatenating the prefix and the full name.
  - Create a method that takes arguments for day and month, and indicates if the day given is the person's birthday.
  - Create an “application” class with a main method that exercises the Person class by creating an array of four Person objects with different names and genders, and print out the formal addresses for each.
  - Solution: SimpleObjectLab

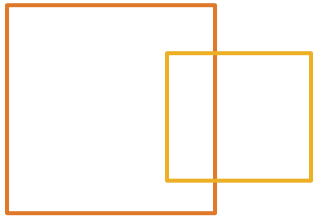
Lab 4

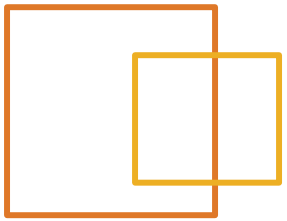




- Add an ID to the Person class
  - Make sure that your Person class properly encapsulates it's state variables.
  - Add an **id** property to the person class.
  - Create an automatic way to assign a unique id to every new Person object created. (Hint – use a **static** variable to hold the next ID)
  - Create two or more instances of your new Person class, and make sure that they all have unique **id**s.

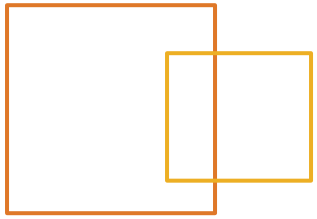
Lab 5a

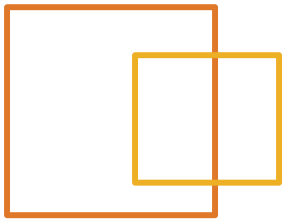




- Inheritance
  - Use the Person class created in section 2 (or create it if you did not) and create a subclass called UniversityPerson. Resolve the constructor requirements, and add a new field that indicates if the Person has a teaching role or not. Modify the behavior of the method that returns the prefix so that it returns "Professor" if the person has a teaching role, or "Dr" if not.
  - In your “application” class create an array of six Person references. Initialize the array with three Person objects, and three UniversityPerson objects. Print out the formal addresses for each.
  - Solution: SimpleInheritanceLab

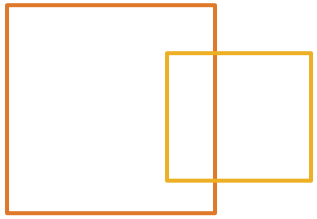
Lab 5b

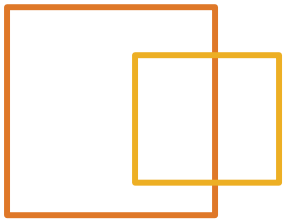




- Interfaces
  - Define an interface Photographer, that declares a method takePhoto(). Create two classes that implement this interface. One will be called Artist, and this class will be a subclass of your existing Person class. The other class will be SpySatellite, and this has no special parent class. Provide implementations of the takePhoto method in each case so that the method prints out some message that is representative of the nature of the photographer (perhaps the Artist says "say cheese" and the satellite prints "I can read your license plate".
  - Create a class that contains a main method. In the main method, create an array of Photographer, and use this to refer to an Artist and a Satellite. Iterate over the array and ask each element of the array to take a photograph.
  - Solution: PhotoInterfaceLab

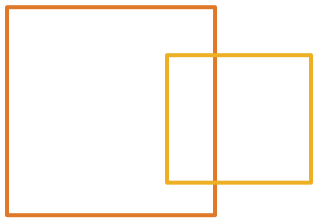
Lab 6

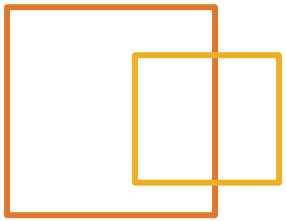




- Exceptions
  - You are going to add code to your Person class constructor to check that the birthDay and birthMonth passed in are valid. If not, you should throw a InvalidDate exception.
  - Which means that you are first going to have to create a new Exception class called InvalidDateException.

Lab 7





- Packages
  - Put your [*]Person classes and your application class into separate packages, e.g. *com.yourbiz.hr.domain.Person*, and *com.yourbiz.hr.app.MainApplication*.
  - Which means that you are first going to have to create a new Exception class called InvalidDateException.

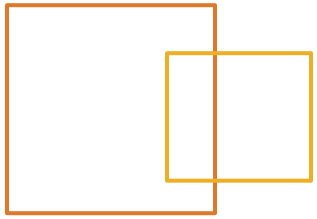
Lab 8

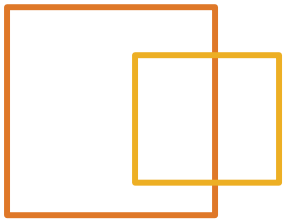
- Description: Use the Mixer class in the **MixerLab** project as your starting point. Refactor the Mixer so that the frequency map becomes type safe. The frequency map should contain <String, Integer> as its map structure. The List should contain <String> as its element types.
- You should get rid of all the “raw type” warnings that Eclipse gives you.
- Duration: 15 minutes

Lab 8.5 (optional)

- Description: The `getFrequencyMap` 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.
- Duration: 15 minutes

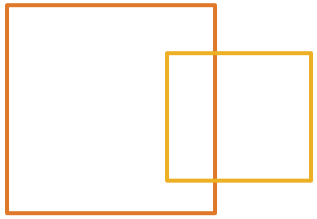
Lab 9

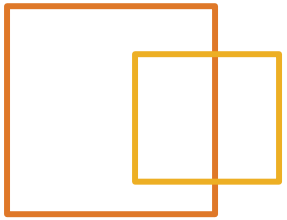




- I/O
  - Write a program that prompts the user for a file name, and attempts to display information about the file (size, read/write permissions, parent directory) and the contents of the file (as text) on the console. If the file does not exist, loop round and prompt the user to re-enter the filename, otherwise exit.
  - Solution: SimpleFileLab/SimpleFileLabToo

Lab 10





- Networking
  - Write a File Information Server that will provide the following service:
    - On establishing a connection, it should read a filename from the client.
    - If the requested file exists, it should send back basic file information to the client.
    - If the file does not exist, send an appropriate error message.
    - You should be able to reuse code from the I/O lab and the NetworkServer and NetworkClient projects.
  - Write a File Information Client that will prompt the user for filenames, then establish a connection the server and request information about the file.
  - Solution: NetworkFileLab