Object Oriented Development

# Core Java Syntax Exercise

What does this exercise cover?   
  
This exercise will give you a chance to practise using the Java language and the basics of Java syntax.

# How long will the walkthrough take to complete?

2-3 hours

# What should you have already completed?

Java Day 1 Walkthrough and associated pre-requisites

# What do you need?

In order to complete this exercise you will need:

* Java Development Kit 1.6 or above
* Eclipse IDE Kepler or above
* Subversion

# Instructions

Open Eclipse and starts a new **Java** project called **Exercise1SyntaxPractise**

# Setting Up

Create a package called *com.fdmgroup.*

You will create a number of sub-packages within this one, one for each exercise.

# Some Useful Tips

Remember to use Eclipses’ auto-complete and shortcuts where appropriate.

Save often. It forces Eclipse to compile your code. If it isn’t saved, you can’t run it.

## Code

Remember that all code in Java must appear inside a method, unless you are declaring an object or variable.

Further, remember that one method *cannot* see the contents of another method.

Java is *case sensitive*.

To print something out to the command line use:

System.out.println(“Text to print out”);

# Running Your Code

There are a number of ways to run your code:

* Ctrl-F11
* Right-click in your main method *or* your class file and select Run As 🡪 Java Application
* Press the green *play* button at the top of Eclipse (this will only work if you have used one of the above methods at least once – it re-runs the most recently executed command)

# Task 1

*General coding– flow of control. No requirement for methods or arrays.*

*“Write a program that prints the numbers from 1 to 100. But for multiples of three, print “Fizz” instead of the number and for the multiples of five, print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”.*

First create a sub-package: *com.fdmgroup.fizzbuzz.task1.*

Create a class called FizzBuzz inside your package and give it a *main* method. All of your code should appear inside this *main* method for this task.

Once you are happy with your solution move onto Task 2.

## Tip

For those of you who have already completed this exercise using JavaScript, think about what you have done. The logic will be the same, but here you will be using Java syntax instead.

# Task 2

*Creating & using methods*

## Structure

Create another sub-package: *com.fdmgroup.fizzbuzz.task2.*

This package will contain a number of class files for this exercise:

* Client (which will contain a *main* method)
* FizzBuzzRunner (which will contain your fizz-buzz code)

## FizzBuzzRunner

Inside FizzBuzzRunner, create a method called fizzBuzz that takes an int and returns no value.

public void fizzBuzz(int number){

//code

}

## Client

Within your main method, create an *instance* of FizzBuzzRunner:

FizzBuzzRunner runner = new FizzBuzzRunner();

Now make a call to the fizzBuzz method of your runner object, passing it a number.

runner.fizzBuzz(100);

## Basic Functionality

Copy your code from Part 1 into your fizzBuzz method and modify it so that it will count up to the input number only.

Satisfy yourself that it runs as expected for various different number values by changing the int passed in from *main*.

## Adding Additional Methods for Modularity

Create 3 additional methods:

private boolean fizz(int num)

Takes a number and, if a multiple of 3, returns true, otherwise returns false:

private boolean buzz(int num)

Takes a number and, if a multiple of 5, returns true, otherwise returns false:

private boolean fizzbuzz(int num)

Takes and number and if a multiple of both 3 *and* 5 returns true otherwise returns false:

## Amend method “fizzBuzz”

Amend your code inside the method fizzBuzz to use the new methods you have just created, rather than performing all of the logic itself.

Once you are happy with your solution move onto Task 3.

## Tip

Again, think about how you completed this exercise using JavaScript, if you have done so. The logic will be the same, but here you will be using Java syntax instead.

# Task 3

*Objects, arrays and MVC*

## Structure

Create a new package: *com.fdmgroup.pensions.task3*

Our package will contain a number of classes (see class diagram on next page):

* Client – will set up and execute our system   
  + Should contain a *main* method only   
    public static void main(String[] args)
* View – will output information to the Command Line   
  + Should contain two methods called *printEligible and printIneligible*

public void printEligible(Person person)

public void printIneligible(Person person)

* PensionController – will invoke our business logic and direct to the correct method in *View*
  + Should contain a method called *handlePensions*

public void handlePensions(Person[] people)

* Person – will represent a *Person* within our system   
  + Should have 3 String attributes: *firstname, lastname* and *dateOfBirth*
  + Should have associated *get* and *set* methods for each attribute
* PensionLogic – will perform our *Business Logic*
  + Should contain a method called *isPensionable*

public boolean isPensionable(Person person, String year)

Our simple system will be designed with MVC architecture in mind (we will discuss this in more detail later on in the course).

## Object Setup

Within your *main* method create a *Controller* object and 3 Person objects with details set as below:

|  |  |  |
| --- | --- | --- |
| **First Name** | **Last Name** | **Date of Birth** |
| John | Smith | 28/06/1993 |
| Jane | Doe | 05/01/1950 |
| Fred | Bloggs | 12/12/1949 |

Now put these 3 Person objects into a Person array and pass it through to PensionController.

## PensionController

*PensionController* should loop over the array of *Person* objects and make a call to the *isPensionable* method in *PensionLogic* for each person in turn, checking for the year 2014.

If pensionable returns **true**, the controller should pass the person through to the *printEligible* method of *View.* Otherwise it should pass the *Person* to the *printIneligible* method of view.

## PensionLogic

Your *isPensionable* method should take a Person object and a year in the form of a String and should calculate whether the person is eligible for their pension in the given year that was passed in.

*For the purposes of this exercise the qualification age should be taken as 65.*

The calculation might look at the last 4 digits of the person’s DOB and calculate their age. It would then compare their age against the qualification age.

The function should return **true** if they might qualify in the year provided to the function and **false** if they do not.

With the data provided Fred Bloggs is the only one who might qualify during the year 2014.

Hint: You may need to use the *parseInt* method of the *Integer* wrapper class. Consult the Java API for more information.

## View

Each of the methods in the *View* object should simply take a *Person* object.

*printEligible* and *printIneligible* should print out, respectively:

<person’s full name> may qualify for a pension this year.

<person’s full name> is not old enough yet.

