Lecture Material

Lecture Material and Lecture Delivery Analysis

Lecture 1:

- Very nice and simple program to begin with. Introducing the basic programming techniques again to the class. The students seemed to engage with the examples well.
- For this lecture and the following lectures, really nice delivery of working through the slides and handwriting abstracted examples. From the perspective of someone with little critical thinking experience it's much easier to grasp programming concepts due to this method of teaching. I really appreciated this aspect compared to lectures of mine rather than just reading off of the slides.
- Another aspect of the delivery I appreciated was the process of how the solution was arrived at. Working through the variables needed for the solution > methods > classes gives a really tangible approach to handling large tasks, as this is something I don't think someone without a programming background should be expected to know.
- Replit is the perfect approach to programming for these students in my opinion, notably for this year. A very frustrating part of programming classes in general and online is the setup. It shouldn't be the focus for the students this early on. Also really convenient is the ability to be able to view their work and to allow for starter code.
- Fair practical assignments. On the easier side of things as the course accelerates greatly in the next few weeks but very reasonable expectations for their prerequisite abilities.

<u>Homework Assignments</u>

Dice Game

Dice class

Make a Dice class with:

It should have 1 variable:

- faceUp – an integer;

It should have 2 methods:

- rollDice; /* returning a random value 1-6, stored in faceUp */
- reportDice; /* reports the dice's current value: faceUp*/

Make a controlling/main DiceGame and let it:

- Create a dice;
- Roll it and show the result, saying if it is greater than 3;

Dice Game Stretch Goal

Use the Dice class already created;

Make a controlling/main DiceGame and let it:

- Create an array of 12 rolls;
- Show the results;
- See if, between all rolls, all the numbers from 1-6 have been rolled.
- If so, report 'You're a winner' and end.
- Otherwise repeat until 'You're a winner'
- You may have to up or lower the '12' to get winner status! See how you do

Lecture 2:

- Fundamentals covered in this lesson. A lot of little random topics covered in the lesson together but all really necessary, picking up from the previous module and what was missed/needed to be reviewsed. There's not much new content in this lecture mostly just a review. Not necessary but could also include switch statements which would tie nicely in with the do/while and the for loop explanations, and wouldn't require that much more effort.
- Scanner is very well demonstrated with plenty of examples. Very thorough coverage with explanation of a lot of edge cases such as the buffer etc. Definitely the right time to introduce this concept.
- Very fair tasks for homework. Not much thought to put into the material of the calculations and therefore more so concentrating on this week's content.

<u>Homework Assignments</u>

Dice Stretch Goal incl. Do/While

Finish off the Dice game (the stretch goal) version and remove all the duplicate code that results from the 'while…do' construct;

Temperature Conversion

- Create a class Temp that has 3 values: one for Fahrenheit, one for Celsius and one for Kelvin. It should have methods that allow a user of objects of that class to insert a value in any three of these degrees and update the other 2 values accordingly. It should also allow reporting of all three values.
- Create a coordinating (Conversion) class that asks the user for the unit they want to convert, takes in that unit and value and converts to the other units in the class.

Temperature, Weight and Height Conversion

- Create a Weight class that has 2 values: one for lbs and one for kgs.
- Create a Height class that has 2 values: one for inches and one for cms.
- Enhance the Conversion class to let the user select Temp, Weight or Height and do the conversions using any of the three classes
- Allow the user to do more conversions until they are satisfied
- Maximize encapsulation in your code
- Keep a count of the total number of conversions done, and report it at the end of the execution

Lecture 3:

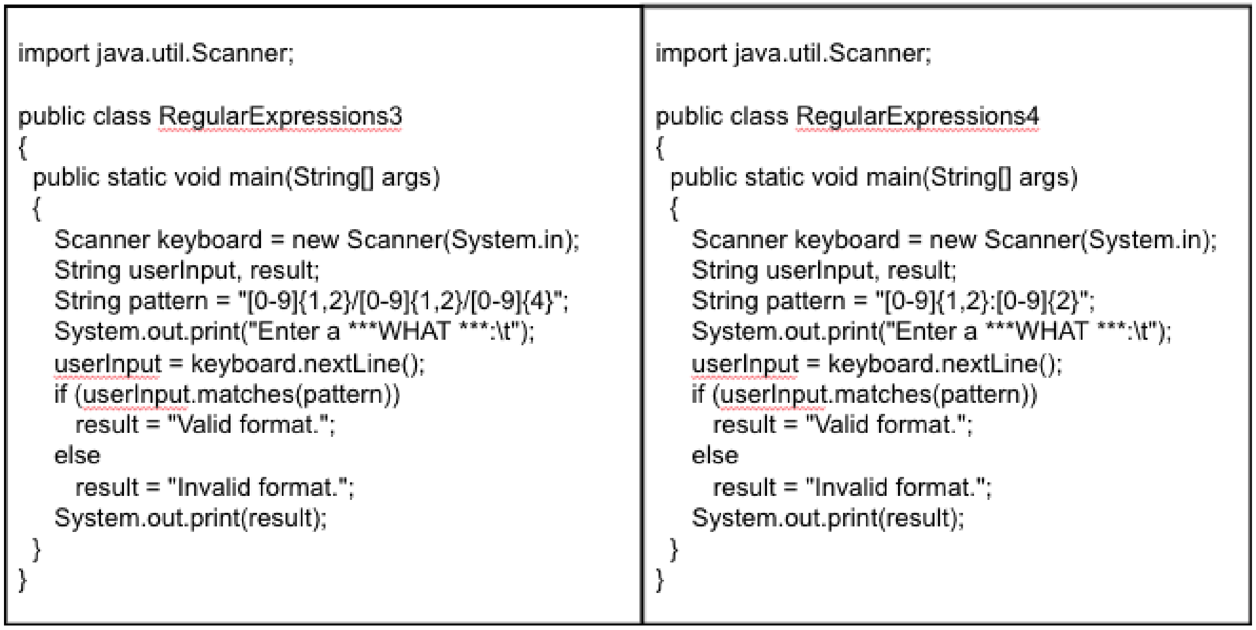
- Very important and fundamental object principals static, encapsulation. Very basic and a good time to set up for OOP.
- Also included in this lecture is an odd arrangement of remaining topics; date objects, pass by reference/value and input validation. Again a mismatch of topics but necessary.
- Pass by validation is overall very well explained but the corresponding written/drawn live explanation is a little harder to follow. It's a foreign concept for the students to understand but not as difficult a concept to grasp as portrayed.
- The input validation is also an interesting topic to spend as much time on. This explanation also bleeds into the next lesson. Certainly interesting on my part learning something new about in

detail error checking etc. but the fine details are something these students I imagine could do without.

- Probably on the difficult side of things for the assignments. 3 problems they'll definitely spend a lot of time on.

<u>Homework Assignments</u>

Error Handling Final Example



```java
import java.util.Scanner;

public class RegularExpressions3
{
  public static void main(String[] args)
  {
     Scanner keyboard = new Scanner(System.in);
     String userInput, result;
     String pattern = "[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}";
     System.out.print("Enter a ***WHAT ***:\t");
     userInput = keyboard.nextLine();
     if (userInput.matches(pattern))
        result = "Valid format.";
     else
        result = "Invalid format.";
     System.out.print(result);
  }
}
```

```java
import java.util.Scanner;

public class RegularExpressions4
{
  public static void main(String[] args)
  {
     Scanner keyboard = new Scanner(System.in);
     String userInput, result;
     String pattern = "[0-9]{1,2}:[0-9]{2}";
     System.out.print("Enter a ***WHAT ***:\t");
     userInput = keyboard.nextLine();
     if (userInput.matches(pattern))
        result = "Valid format.";
     else
        result = "Invalid format.";
     System.out.print(result);
  }
}
```

Credit Card Structure Validation & Luhn Validation

e.g., Credit Card Number 4567123474567836

Step 1 and Step 2:

```
      (4 * 2) + (6 * 2) + (1 * 2) + (3 * 2) + (7 * 2) + (5 * 2) + (7 * 2) + (3 * 2)
=        8    +   12   +    2    +    6    +   14   +   10   +   14   +    6
=        8    +   3    +    2    +    6    +   5    +    1   +    5   +    6
=                          36
```

Step 3: 5 + 7 + 2 + 4 + 4 + 6 + 8 + 6 = 42
Step 4: 36 + 42 = 78

Step 5: 78 modulus 10 = 8 thus invalid

You have been requested to assist with the development of a solution to check if a credit card number is valid. You are required to write a method named luhnFormula() that will accept an array of integers. You should (i) first check that the values in this array have passed the criteria in Table 1 and then (ii) that in each location of this array is an integer in the range 0 to 9. The luhnFormula() method should apply the steps 1 to 5 to the values in this array and return true is these digits represents a valid credit card number otherwise it should return false.

Make sure you test it rigorously

Each type of credit card begins with a prefix or range of prefixes and is of a certain length. Table 1 shows the details of two commonly used credit card types.

Card Type	Prefix(es)	Length
MasterCard	51-55	16
Visa	4	13 or 16

Table 1: Details of two commonly used credit card types

In addition to these rules a credit card number is subjected to the Luhn formula to determine if it is valid. The following are the steps of this formula:

1. Take each odd positioned digit and multiply it by 2. If the result of multiplying one such digit by 2 is a 2-digit number these digits should be summed so that the result is a single digit.
2. Add results of all multiplications in Step 1.
3. Sum all even positioned digits.
4. Add the result of Step 2. to the result of Step 3.
5. Modulus the result of Step 4. with 10. If the result is 0 this indicates that the credit card number is valid.

Palindromic Number Check (SOLVED)

"A palindromic number or numeral palindrome is a number that remains the same when its digits are reversed." Examples of palindromic numbers are 1234321, 777, 1 and 66. You are required to write a Java application program that determines whether or not a number is a palindrome (cf. Figure 3).
Your program should accept one argument from the end user. A valid argument is a number in the range 1 to 999999999 (inclusive). An appropriate error message should be displayed in each of the following situations:

1. the user does not supply an argument,

2. the format of the user input (supplied as the argument) is incorrect and
3. the input value provided (for a value supplied as the argument) is outside the desired range.

If the user supplies valid input your main() method should call a method named checkIfPalindrome(). The checkIfPalindrome() method should return true if the user input is a palindromic. Otherwise it should return false. Your main() method should display a message stating whether or not the user input is palindromic.

Lecture 4:

- Really good topics covered in this lesson; "this" keyword, overloading constructors, immutable objects and classes, error handling and try catch finally testing. Great time to introduce all of these topics.
- There's a lot of new content in this lesson, any one of the object principles taught in this lesson could be taught previous to maintain a manageable learning curve.
- Very interesting addition to the lesson is the inclusion of the date class. If students are struggling/dedicating too much time to this concept the course could make do without it.
- No homework assignments received, I imagine the first of the continuous assessment assignments were given out. Couldn't see this in the slides or the lecture.

Lecture 5:

- The course work certainly gets a lot more challenging in this section for OOP. I think for students with limited OOP knowledge at all they will find these 4 weeks very challenging and jam packed.
- Topics covered in this lecture were: abstraction and encapsulation, API's, the relationship between classes and the "super keyword". Overall good topics to deliver in the introduction that I would feel are the most basic. This was well planned the topics to deliver first.
- Sufficient worked examples in the slides.
- A lot of new abstract topics to be introduced this week are definitely a step up from the weeks before. Over all manageable considering the content yet to come.
- Fair exercises to look through. Will take the students a while to work through but to understand the future concepts they should spend a good while working through the basics.

Homework Assignments

Beyond Designing the Loan Class

Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000).
-loanDate: Date	The date this loan was created.
+Loan()	Constructs a default Loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+getLoanDate(): Date	Returns the date of the creation of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+getMonthlyPayment(): double	Returns the monthly payment of this loan.
+getTotalPayment(): double	Returns the total payment of this loan.

Your job to implement a loan class: You can choose between one where interest rate is defined as above or one where it is defined as a string where the '%' is included. But you have to make the interface/contract the same.

Use your implementation to set up 4 loans, 2 with default values, one for 2000 at a rate of 3.5% and one for 3 years (all other values defaulted)

Then use your loan classes to calculate monthly and total payment for each loan.I'm not too concerned about the formula you use to do this (but it has to be reasonable!)

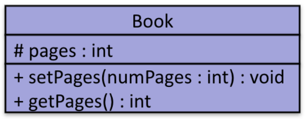
Again note how different implementations (interest rate calculations – and thus interest rate calculation changes, representation of rate) don't screw up the client software.

Class of Cards and Comparing

- Implement a new class card which should have a suit and value field, and should be able to print itself in a nice format (Ace of Spades)
- Implement a hand of cards that has an array of 5 cards, randomly selected from a "deck".
- It should have functionality to check itself for pairs, triplets, a flush and a straight
- Implement a controlling class that deals 2 hands and evaluates them, where a flush beats a straight, beats triplets, beats pairs.
- The controlling class should announce a winner.

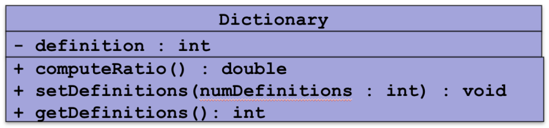
Book, Dictionary and Word Relationship.

Mess around with visibility between the classes. What can be accessed and where?

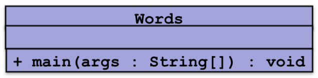


```
public class Book
{
  protected int pages = 1500;

  // mutator method
  public void setPages(int numPages)
  {
    pages = numPages;
  }
  // accessor method
  public int getPages()
  {
    return pages;
  }
}
```


Dictionary
- definition : int
+ computeRatio() : double
+ setDefinitions(numDefinitions : int) : void
+ getDefinitions(): int

```
public class Dictionary extends Book
{
    private int definitions = 52500;
    public double computeRatio()
    {
       return definitions/pages;
    }
    public void setDefinitions(int numDefinitions)
    {
       definitions = numDefinitions;
    }
    public int getDefinitions()
    {
       return definitions;
    }
}
```



```
public class Words
{
  public static void main(String [] args)
  {
    Dictionary webster = new Dictionary();
    System.out.println("Number of pages: "+
webster.getPages());
    System.out.println("Number of definitions: "+
webster.getDefinitions());
    System.out.println("Definitions per page: "+
webster.computeRatio());
  }
}
```

- Words class instantiates an object of class Dictionary, which is derived from a class called Book
- in the main() method, 3 methods are invoked through the Dictionary object (two declared locally in the Dictionary class and one that is inherited from the Book class)

Lecture 6:

- New topics cover protected keyword, inheritance, abstract classes, overriding methods, and binding. I think these topics are very new and different, and could've used the time set to reviewing the previous lectures topics. I really think that this lecture in particular, requires a lot more time to dedicate to the material and just to go through a bit slower.
- Abstract classes, overriding methods, binding could all do with a lot more of a practical programming example to be worked through. A brand new concept for new programmers to OOP to understand at the very beginning.
- The assignments are easy enough given the difficulty of the material. They don’t cover all that was displayed in class and therefore a lot of what was delivered may get forgotten.
- The 2nd assignment delivered this week is a fair assessment of the key principles to take away from OOP. A lot of the aspects aren’t covered in the assessments which may be intentional for difficulty purposes.

<u>Homework Assignments</u>

Relationship Between Classes Exercise

Using the class descriptions just derived, Ask the user for the radius for a circle, the length of a square and the height/width of a rectangle.
Get the total area of each. Make sure that every method (including constructors) in your classes contains a printout showing what is being executed.
(Lets predict now but) Using those printout statements make sure you are aware of which method you are in as the program executes

Array of Geometric Objects Abstract Classes

As per the class definitions we have just talked about, create an array of 3 GeometricObjects shapes and ask the user to specify the nature of each of these shape ('R', 'C' or 'S')
Depending on the shape also ask the user for the radius, length/width or length) and create them in the array
Calculate the total area of the 3 shapes.
Again, use print statements to illustrate the execution sequence of all of the code for yourself

Assignment 2

- Create a circleBase class (possibly off of GeometricObject?). It should have a private colour and protected radius. It should have get/set methods for radius and colour and an abstract method for calculating surface area.
- It should have a default constructor (radius=1) and a constructor that takes an int, specifying the radius;
- Create 2 sub-classes: Cone and Cylinder;
- They should have an additional field: height, with get/set methods. They should have a method for calculating volume.
- They should have 2 constructors: default (radius =1, height=1) and a constructor that takes 2 ints, specifying the radius and the height;
- There should be a driver class that makes a cone and a cylinder (of whatever radius, height you wish) and reports back on their volume and surface area. (staging goal: week 1)(Note: not all the fields may have been specified)
- Each of the sub-classes should implement a comparable interface.
- The compareTo method should compare the shapes based on their surface area.
- Use this to compare the 2 shapes you made last week, to see which is bigger. (Staging goal week 2)
- Create an array of 5 elements. Populate it with 5 cylinders/cones of your user's choice (whether they are cones or cylinders and they can also specify the height, radius);
- Sort this array so that you can report on the biggest and smallest;

Lecture 7

- Again, a lot of time is dedicated to the review of the previous lesson which I wish was given more of this time. I think students may get a bit lost between this week and the previous week in particular, especially due to this week building on the past.

- New material covered in this lecture: interfaces, abstract classes vs interfaces, default methods in interfaces, good time to deliver this material and a good number of topics to introduce at once.
- Could do with a few more examples where interfaces are simplified as well as the common java interfaces examples used in the slides.

Homework Assignments

Complete the Circle1 Code

```
public class Circle1 extends GeometricObject implements Comparable<Circle1>
{
   private int radius;

   public Circle1()
   {
    radius=1;
   }

   public Circle1(int rad)
   {
     radius=rad;
   }

   public int getRadius()
   { return radius; }

   public double getArea()
   {
    double area = 3.14*radius;
    return area;
    }

   public int compareTo(Circle1 C)
   {
    //Blah-de-Blah: compare them based on their area
    }
}
```

Complete the Circle1 code:
- Implement the compareTo() method

Make a driver programme that:
- Makes four Circle1 objects;
- And compares them to one another;
- printing out the largest;

Lecture 8
- Comparable and file input and output both covered in a lot of detail. Two main focusses of the class chosen and covered well with edge cases and plenty of examples in the slides and worked examples in replit.
- Very good demonstrastinos of both using HUE. Fair excesrsie to complete reflecting the work presented in the lectures.

Homework Assignments

Editing Text Files:

Create a file of the following format:
- Jim,Buckley,jim.buckley@ul.ie,086 1877760
- Mary,Williams,m.Williams@lero.ie,0887775434
- ...

Create a programme that allows users to:
- Change a record’s email address;

- Add a new record;
- Delete a record.

Lecture 9

- New JavaFX content. Really in detail step by step installation and setup with Maven. I followed the steps myself and it worked perfectly.
- A good time to get the students set up on an IDE as it's important they familiarise themselves with one. When lectures return in person this will become a lot easier to manage with students' issues and should not requie the full 4 hour demonstration.
- Good explanation of the uses and functionality of JavaFX. Examples worked through well and are very easy to follow. Custom pane and horizontal box examples covered at the end of the lecture should be demonstrated slower due to their dependencies. Students will find these classes a bit more challenging.

Homework Assignments

Setup IntelliJ with Maven and JavaFX successfully

Follow the recording and set up the environment as shown in the lecture

Review the Classes Displayed During Lectures

Work through the classes that were displayed in the lecture. Edit them in the IDE

Lecture 10

- The 2 hour lecture video was not included in the resources mp4 so I couldn't view. I'd imagine the other classes for JavaFX were covered as Ashish mentioned the JavaFX was assigned 4 hours of lectures. An appropriate amount of time to spend on this topic as a lot of the first lecture will probably be dedicated to trouble shooting with IDEs.

Lecture 11

- Data structures covered in this week's lecture. A nice and very practical topic to end on, easy for the students to get a grasp on compared to OOP.
- Well explained data structures and the relationship between them all in the hierarchy. The lecture concentrated a lot on just arraylists and linked lists which was beneficial as they're more commonly used for intermediate programmers, and it's equally important for them to be familiarised with some lesser used data structures.
- Could do with some practical examples of what some of the data structures are used for in programming problems such as stacks and queues.
- Towards the end of the lecture a lot of the material was run through very quickly.
- Good topic to introduce to allow the students to broaden their Java knowledge and to guide them towards learning more about data structures.
- In this lecture and the following lecture, I think that the material gets a lot less intense and it is a nice conclusion to the course.

Homework Assignments

Plurals and Capitalised List of Word

Not compulsory -
Write a program that reads a file and displays
the words of that file as a list.
- First display all words.
- Then display them in reverse order.
- Then display them with all plurals (ending in "s") capitalized.
- Then display them with all plural words removed.

List of Numbers Manipulation

Not compulsory -
Write a program that reads a file full of numbers and
displays all the numbers as a list, then:
- Prints the average of the numbers.
- Prints the highest and lowest number.
- Filters out all of the even numbers (ones divisible by 2).

Lecture 12

- A step by step run through of the poker program.
- A good visualisation of a full program. Again building the program from just an idea to a full program. This development program I think is very useful to the students and makes the lectures more engaging.
- Doesn't cover a lot of the topics discussed in the lectures but is a useful tool for students to work through and understand a full program from start to finish.

Conclusion

Overall quite random topics included at the start and end of the course. Well delivered in an engaging and accessible manner especially for remote learning. Course material was covered most of the time in good detail, a few topics could use a few more examples extracting the concept. The course gets quite a bit more intense and difficult in the mid section, a lot of new and difficult material is covered, so the learning curve I would feel is not consistent (which may be intentional also). All topics that I would expect to be covered in this module are covered as well as a bit more.