ECS414U Assessment Booklet
January 2018

Learning Outcomes
On passing the module you will be able to:

- tackle more complicated programming problems than in the first semester
- decompose programming problems using principles of object orientation
- use most of the object-oriented programming concepts when writing programs
- test and debug simple programs to ensure they work correctly
- explain programming constructs and argue issues related to programming

The assessment, which consists of coursework and an exam, aims to facilitate your development of these skills and test that you have gained them.

Coursework
This consists of several components including programming exercises and a mid-term test.

> **Aim:** The aim of the coursework is to test on both theoretical and practical Object Oriented programming skills.

Final Exam
This will be an electronic exam, in exam conditions at a computer laboratory. The final exam will be during the normal exam period.

> **Aim:** The aim of the exam is to ensure you have gained the programming skills necessary to succeed in further courses.

Marks: out of the 100 possible marks for this module, 40 are coursework marks and 60 are the final exam marks.

==========

ASSESSMENT

Your coursework marks:
please read carefully below so to understand how your coursework marks will be calculated

The coursework consists of the following components:

1. **A series of short programming exercises (worth 7 marks: each level worth 1 mark)**
2. **Three tests: the first one worth 4 marks, the second one worth 6 marks and the third one worth 10 marks**
3. **A programming mini-project and mock exam worth 13 marks in total: first three levels of mini-project worth 1 mark each, next four levels worth 2 marks each + 2 marks from the mock exam.**

Important note: only students getting at least 1 mark in the mock exam are eligible for level 6 of the mini-project and only students getting 2 exam marks in the mock exam are eligible for level 7 of the mini-project (so marks for level 6 and 7 of mini project are void unless student meets necessary requirements in mock exam).

The mock exam marks are awarded as follows: 1 mark for students getting between 50% and 80% in the mock exam, 2 marks for students getting > 80%

Maximum marks for the coursework = 40 (= 7+4+6+10+13)

FINAL EXAM:

The remaining 60 marks of the module will be awarded in the final exam.

To pass the module you need to get at least 30 marks in the final exam and 40 marks overall (i.e. combining exam and coursework).

The final exam for the module will be held during the normal exam period. It will be an electronic exam, held in a computer laboratory, where you will have to write code.
The exam is (largely) automatically marked, you will be awarded marks if your code compiles and executes correctly, and returns the required values.

==========

Assessment Deadlines

Coursework deadlines /assessment are in the following weeks:

- for labs exercises you will get the mark for exercise N only if you are assessed for exercise N within week N +5 of term, so for having the mark for exercises 1 you need to be assessed for exercises 1 by week 6 of term (1+5=6). For exercise 2 that will be week 7 and so on. After that deadline you will not be awarded marks for that exercise
- for the miniproject the level N has to be assessed by week N+9, so level 1 by week 10, level 2 by week 11, level 3 by week 12 etc. So if you haven't been assessed until week 12 you can still be assessed but only get marks from level 3 on. (to be precise you need to be assessed within week Min{12,N+9} because there is no assessment after week 12 of term)
- only two exercises or two mini-project levels can be assessed in each lab per student. If students want to be assessed for more than that they need to be assessed by the module leader.
- recorded lab marks are published on the module page few days after the labs, so in week N you should see published your marks up to week N-1.
- **Students can only get their programs assessed in their allocated lab times.**
- **Students can only do the three in term tests in their allocated lab times.**

You are expected to get programs completed and marked well before the above final deadlines. Aim to get at least one finished each week to give yourself the best chance of a high mark.
Tutors/Demonstrators will mark at most 2 programs for each student in any week, INCLUDING WEEK 12. For example, they will mark only one short exercise and one level at the mini project, or only two assessed exercises, or only two levels at the mini project.

disclaimer: the above rules guarantee the right of students of being assessed and the assessment being recorded and counted only if they comply with the above rules

Short Programming Exercises

Each of the seven short programming exercises counts for a single coursework mark .
They are intended to be relatively simple, developing your understanding of the constructs introduced in a single lecture. You must complete each to a satisfactory standard and get it assessed before moving on to the next. **You may have a particular program assessed more than once** if it was not up to the standard required on the first attempt. Your tutor will explain any problems when he/she assesses it. You may also be asked questions about your program, or be asked to make modifications to prove that you understand it. When the program has been completed, and you have demonstrated your understanding of it, to a satisfactory level, the tutor will sign the appropriate level.

Note the **marks are not just for presenting a correct program but for convincing the assessor that you have achieved the learning outcomes** – i.e. you have the required skills and understanding. If you cannot answer questions on your program or cannot make simple changes on the spot, then you have not reached the required level

Note: A requirement for passing the short programming exercises is to **demonstrate that you can write fully working programs that contain a main method**. Relying purely on BlueJ to demonstrate the functionality of your programs, without having a main method will not get you a pass.

At the end of some exercises, you'll see a section headed "Optional Extras". This describes more advanced features of Java that could be used in each exercise, or in some cases give example written exam questions. As the name suggests, these are optional and not marked, so you can pass each exercise without attempting them. However, the extra programming ideas will help you gain confidence at programming, and the exam questions will provide useful revision and preparation for your exams.

==========

Short Assessed Programming Exercise 1: Basic objects

To pass this exercise you must demonstrate that you:
Understand the concept of objects and classes
Are able to work with instance methods, instance variables and constructors

Consider the following class:

```
class Counter {
   private int count;

   public Counter(int initialCount) {
     count = initialCount;
   }

   public Counter() {
     count = 0;
   }

   public void increment() {
     count += 1;
   }

   public void reset() {
     count = 0;
   }

   public int getValue() {
     return count;
   }
}
```

Save this class in a file `Counter.java`. In the same directory, create a simple class (e.g. `TestCounter.java`) that does the following things:

1. declares a `Counter` variable named `ctr` and initializes it to contain a `Counter` object with instance variable `count` set to 0
2. increments the instance variable of the `ctr` object
3. prints the value stored in the instance variable of the `ctr` object
4. changes the value of the instance variable of the `ctr` object to zero

To have this exercise assessed it is essential to demonstrate that you have both classes, `Counter.java` and `TestCounter.java`
Hint: Use the classes `Account.java` and `TestAccount.java` from the lecture to guide you in this exercise.

When your program works have it assessed. If you are having problems talk to your TA about it.
Optional Extras (not assessed)
Write an instance method `decrease(int n)` of the `Counter` class. The method decreases the value of `count` by a number equal to `n`.
Then, assume that `Counter` objects can not hold negative values in their instance variables. Insert some control structure in `TestCounter.java`, that prints a warning if ever the value of the instance variable of the `ctr` object becomes negative. The value of the instance variable of the object should then remain at its last positive value.

For example, if the value is 100 and we decrease the value by 150, the output might be "`Warning: Illegal operation on Counter object, object can not hold negative values`" and the value would remain 100.

Short Assessed Programming Exercise 2: Getters/setters and simple calculations

To pass this exercise you must demonstrate that you are able to
Write some basic code that implements instance methods
Write a simple class that invokes instance methods and does basic calculations based on them
Write code that implements getter and setter methods

Write an `Account` class with

1. a constructor `Account(double initialBalance)`
2. instance variables `balance` and `interestRate`
3. methods `deposit(double amount)` (to change the balance by adding an amount), `withdraw(double amount)` (to change the balance by subtracting an amount), `getBalance()` (to print the balance), `setInterest(double rate)` (to change the interest rate) and `computeInterest(int n)` (to compute the balance+interest after n months).

You can use the code for the class `Account` from Week 1 lecture to get you started. You can assume that `interestRate` corresponds to an annual interest rate.

Create a new class in a new java file, e.g. `TestAccountInterest.java`
This new class should:
create two different `Account` objects, under two different names and different initial balances (e.g. `account1` at 500, `account2` at 100)
compute the balances (including the interests) of the two accounts after 12 and 24 months

Note: the formula for calculating the compound interest is the following:

balance after n months= balance * (1+interest) "raised to the power" (n/12)
(to compute "a raised to the power b" in Java you can use Math.pow(a,b))

for example for an initial balance of 100 and interest at 10% (remember 10% is 0.1) we have:

`computeInterest(12)=110, computeInterest(24)=121, computeInterest(36)=133`

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras (not assessed)
Get the program to print how many months it would take, on the current interest rate, for the balance of the accounts to reach a specific amount of money, e.g. 2000.

Write a mortgage calculator that given an amount to borrow, an interest rate and a mortgage term (e.g. 10, 15, 20, 25 years) returns the monthly repayment

==========

Short Assessed Programming Exercise 3: Class Methods
To pass this exercise you must demonstrate that you are able to
Write a program that takes command-line arguments
Write code that implements a simple class method (distinct form the main method)

Write a class containing the main method and another class methods (so there are two class methods in this class).
The main method calls the other class method and passes to it as arguments three command line arguments converted to double. The class method then returns the middle number of the three in a specific output format.
For example, if the class is in the file `middle.java`:

```
java middle 5 6.3 3
```

should print on screen "`5 is between 3 and 6.3`".
If you use BlueJ to test your program, remember that command line arguments are provided inside the bracket and inside double quotes, i.e. {"5","6.3","3"}.

The program should also output an appropriate error message if not exactly three command-line arguments are supplied. For example

```
java middle 5 3
```

could print on screen "`Error`".

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras (not assessed)

If you know about exceptions the program should also handle the case where any of the command-line arguments is not a number. For example, `java middle 5 6.3 xcvf` should trigger an exception and write on screen "input not valid".

Add a further class method that, in the case not exactly three command-line arguments are supplied will write on screen an appropriate error message (i.e. repack the code you already written in a different method called appropriately)

Short Assessed Programming Exercise 4: Inheritance

To pass this exercise you must demonstrate that you are able to explain inheritance and
Write the code for basic classes
Write code that makes use of inheritance

Create a `Vehicle` class containing three instance variables of type `double`: `horsepower`, `aerodynamics` and `weight`. The class should also have getter methods for the three instance variables.

Create a `SportCar` class extending the `Vehicle` class; Sport cars have an additional double variable `topspeed`. All sportscars have `aerodynamics`=0.5. The class should have a getter method for the new instance variable.

You will need to create three instances of `SportCar:`

The first has `horsepower`=200, `weight`=1500, `topspeed`=220
The second has `horsepower`=100, `weight`=1000, `topspeed`=170
The third has `horsepower`=135, `weight`=1100.2, `topspeed`=173

(Notice that these will need to be created in the class `TestConsumption` below, and not using the BlueJ interface)

Create now a `TestConsumption` class with a `main` method that creates these three new vehicles, and prints their fuel consumption according to the formula:
(1000+(`weight`/5))*(`topspeed`/100)*(`aerodynamics`*`horsepower`)/10000

(Notice for the car enthusiast: this is not a realistic formula)

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras (not assessed)
Create a further subclass of `Vehicle` called `Motorcycle`; think of appropriate values for the parameters of this subclass, compute `TestConsumption` on instances of this subclass

==========

Short Assessed Programming Exercise 5: Polymorphism

To pass this exercise you must demonstrate that you are able to explain polymorphism and
Write code that implements basic classes
Write code that makes use of inheritance and polymorphism

Extend the previous exercise by creating a `Van` class that extends the `Vehicle` class; a van object has an additional double variable `carryweight`. The class should have a getter method for the new instance variable.

Add to the `Vehicle` class a method with signature:
`public double acceleration()`
this method returns the acceleration of the vehicle according to the formula
(100/`horsepower`)*`aerodynamics`*`weight`/100

The method `acceleration` is overridden in the Van class by the formula
(100/`horsepower`)*(`aerodynamics/2`)*`weight`/100

Write a program which computes the acceleration (from 0-60mph) for the three sports cars defined in the previous exercise, as well as for an instance of Van with `horsepower`=100, `aerodynamics`=0.9, `weight`=3500, and `carryweight`=160.4.

Note that the solution has to use the `acceleration` method in a polymorphic way.
(Notice for the car enthusiast: this is not a realistic formula)

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras (not assessed)
Extend the above including the class `Motorcycle`; check polymorphism is still working, also extend the program with dynamic binding, e.g. creation of vehicles of random sub-type.

Short Assessed Programming Exercise 6: Graphical User Interfaces and The Abstract Window Toolkit (AWT)

To pass this exercise you must demonstrate that you are able to
Write code that uses the AWT
Create a simple GUI
Include appropriate comments in the code, sufficient for the assessor to understand your coding

Create a simple GUI for the `Vehicle` class.
The GUI should have buttons for creating new `SportCar` and new `Van` objects, and should display the values of the instance variables of these vehicles in textareas in the GUI.
The GUI should also calculate and display in appropriate textareas the fuel consumption and acceleration of the `Vehicle` objects that have been added by the user, using the formulas given in the previous two exercises.

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras

Give the GUI a more professional look. For example, you can add options to sort the objects on the GUI display based on their fuel consumption (eco-friendly option) or based on their acceleration. Feel free to add more functionality to the GUI.

==========

Short Assessed Programming Exercise 7: Data structures - File I/O

To pass this exercise you must demonstrate that you are able to
Write code that uses Generics Data Structures e.g. ArrayLists
Write code that performs basic file I/O operations
Include appropriate comments in the code, sufficient for the assessor to understand your coding

Write a program that reads a text file of a specific format, stores fields of each line in an ArrayList (you have to use Generics), and is able to perform a basic calculation from the ArrayList. More specifically, consider the following text file of capital cities, their population and the country in which they are:

London 7500000 UK
Rome 2700000 Italy
Paris 6000000 France
Berlin 4500000 Germany
Athens 3800000 Greece
Budapest 1700000 Hungary
Lisbon 2800000 Portugal
Madrid 5600000 Spain
Oslo 1300000 Norway
Copenhagen 1200000 Denmark

(note that there are 3 items per line, separated by a single space).

The program should read each line, and store the population field only in a ArrayList (you have to use Generics).
Your program then, given a command line argument, should be able to answer how many capitals have a population of equal to, or larger than, the command-line argument.

For example, for the above sample list, if the command-line argument is 5000000 (5 million), the answer should be 3.

When your program works have it assessed. If you are having problems talk to your TA about it.

Optional Extras

Modify the program so that it takes two command-line arguments, and returns the number of cities with population within the bounds defined by the two command line arguments.

Programming mini-project

You must write a mini-project program. It should be chosen to demonstrate your understanding of and ability to use the different constructs covered in the course. Possible programs are given below. **You choose one of these projects.** You may wish to keep notes on your progress week by week in a log book (for example including a copy of the current program listing and a description of what it does and how it works, or what does not work).

Mini-projects are divided in 7 levels. The marks awarded for each level are as follows:
first three levels worth 1 mark each, next four levels worth 2 marks each+ 2 marks from mock exam (see coursework marks explanation in page 2 for details)

You are expected to develop your program in stages – modifying your earlier version (that has been marked and achieved a given level of proficiency) for the next level version. Once your program has reached a level (see below) you should get it marked. If you are confident then you may wish to skip getting some levels marked. However, all mini-projects must be marked **at least two times**, and the level achieved confirmed in writing. For each level you will be required to state what your program does, the grade that you believe the program should obtain and also explain why it deserves that grade. As with the short programs you may be asked questions on how it works etc – the mark is for you convincing the tutor that you have met the learning outcomes not just for presenting a correct program.

Whole programs submitted at the end of the term for which intermediate stages have not previously been marked will not be accepted.

In the following pages are example programs with indicative grades for different levels of development. However, your program does not have to do exactly as outlined in the descriptions for each level, these are just guidelines to help you through the task: use your imagination (though obey specific restrictions)! The more original is your project the better it will be.
What matters is that you demonstrate you can use the different object oriented concepts (objects, inheritance, polymorphism, etc.) and programming constructs like ArrayLists, file I/O, GUIs, etc., have achieved all the other criteria for a given level as described rather than the precise thing you have achieved. All students' programs should be different in what they do.
What matters is how many criteria you have achieved in your program.
Please avoid to cooperate with other students on the mini project. The more we see similarity in projects the more suspicious that will look the lower the marks awarded...

If you want to do a mini-project not suggested in this booklet then you need to talk to the module leader and him to agree to the topic. If it is not agreed, it will not be accepted. Whatever the topic you should sketch a 7 levels scale of achievement similar to the ones for the suggested topics. So if you think of a mini-project topic non suggested in the following pages you should write down a 7 levels scale of achievement similar to the ones for the suggested topics to show to the module leader for approval. Only students with some previous Object Oriented programming experience are encouraged to original suggestions.

==========

Suggestion 1: A stock market investor simulation

Write a program that simulates an investor buying and selling shares in a stock market. The investor needs tools to display his portfolio and buy and sell shares. Shares value can go up and down so un underlying simulation of the market should be implemented

Level 1 –
Includes **screen output** and **keyboard input** and **basic classes**.
There are java source files for at least three major classes in the program.
Good source comments and code indentation is expected for all implemented parts of the code
Example: The program reads and prints the names of shares and their basic characteristics.

Level 2 – Includes methods and variables for at least three major classes, and all constructions above.
At least 3 major methods fully implemented and working for each class
Example: As above, but also the notion of the buying and sell shares is shown.

Level 3 – At least three major program classes will be implemented, with methods working and well designed, and all constructions above
Use of **inheritance** with at least one superclass and three subclasses
Class, method and variable naming will be clear and consistent
Example: As above, but also there is a basic simulation of the stock market, though most details, commentary etc. may be very simple

Level 4 –
Polymorphism should be used in at least three subclasses, and all constructions above
At least four major program classes will be implemented, with methods working and well designed,
Comments are clear and applied to class and method level consistently
Example: As above, but the simulation is more natural, there is a running commentary of the change in value of the shares with major events reported.

Level 5 –
Use of **ArrayLists or other classes from Java's Collection Framework** in all parts of the program, and all constructions above.
Exception handling is carried out appropriately in all parts of the program
Inheritance is correctly applied to all parts of the program.
Example: As above, but all types of accounts and functionality will now be included in the simulation. The simulation is now mostly ruled by a basic **GUI**.

Level 6 –
Includes **file input and/or output**, and all constructions above
The simulation (including player movement) will be displayed on the GUI
Polymorphism will be fully implemented in all parts of the program
Example: As above, with a full GUI now controlling all aspects of the simulation; data should be read from files.

Level 7 –
Includes everything required for an A grade but also **something special** (using other more advanced constructs or algorithms, or something you just read up on yourself). Make it a program someone would really want to use!
Example: .. real historical data, moving averages,...

Suggestion 2: An adventure game

Write a program that implements a simple (yet exciting!!!) adventure game. Different groups of characters (e.g. magicians, warriors, monsters, different types of warriors, magicians, etc.) will have different characteristics, strengths and behaviour. Heroes may pick up objects on their way, can engage in fights when in close proximity to each other, etc. At least 6 characters from 3 different groups, and 2 different types of object should be included in the game.

Level 1 –
Includes **screen output** and **keyboard input** and **basic classes**.
There are java source files for at least three major classes in the program.
Good source comments and code indentation is expected for all implemented parts of the code
Example: The program reads and prints the names of the characters and their basic characteristics.

Level 2 –
Includes methods and variables for at least three major classes, and all constructions above.
At least 3 major methods fully implemented and working for each class
Example: As above, but also characters, objects, etc. are arranged on a simple layout.

Level 3 –
At least three major program classes will be implemented, with methods working and well designed, and all constructions above
Use of **inheritance** with at least one superclass and three subclasses
Class, method and variable naming will be clear and consistent
Example: As above, but also there is a basic simulation of the game, though character movement and interaction may be very simple.

Level 4 –
Polymorphism should be used in at least three subclasses, and all constructions above
At least four major program classes will be implemented, with methods working and well designed,
Comments are clear and applied to class and method level consistently
Example: As above, but game simulation is more natural, characters can move a specified distance for each turn.

Level 5 –
Use of **ArrayLists** in all parts of the program, and all constructions above.
Exception handling is carried out appropriately in all parts of the program
Inheritance is correctly applied to all parts of the program.
Example: As above, but also basic character interaction scenarios will be included (e.g. fights). A basic **GUI** will also be included showing character details, status, etc.

Level 6 –
Includes **file input and/or output**, and all constructions above
The simulation (including movement) will be displayed on the GUI
Polymorphism will be fully implemented in all parts of the program
Example: As above, with full character interaction and with full interaction with objects in the game (characters pick up objects etc.). Characters' details should be read from a file.

Level 7 –
Includes everything required for an A grade but also **something special** (using other more advanced constructs or algorithms, or something you just read up on yourself). Make it a program someone would really want to use!
Example: Use a smart algorithm to determine the results of fights, use a fancy GUI for the game. Or ...

==========

Suggestion 3: A car racing simulation

Write a program that simulates a car racing. You should include different types of cars (e.g. by brand or engine size or horsepower, etc) with different abilities (e.g. braking, straight line speed, acceleration), characteristics (e.g. aerodynamics, poor at cornering, tends to overheat) etc. Notice the project can be very difficult if you try to implement "arcade racing" where you see the race through the windscreen (you need 3D for that); a simpler simulation is looking at the race from above (you just need 2D for this)

Level 1 –
Includes **screen output** and **keyboard input** and **basic classes**.
There are java source files for at least three major classes in the program.
Good source comments and code indentation is expected for all implemented parts of the code
Example: The program reads and prints the names of cars and their basic characteristics.

Level 2 –
Includes **methods** and **variables** for at least three major classes, and all constructions above.
At least 3 major methods fully implemented and working for each class
Example: As above, but also the structure of the track is shown.

Level 3 –
At least three major program classes will be implemented, with methods working and well designed, and all constructions above
Use of **inheritance** with at least one superclass and three subclasses
Class, method and variable naming will be clear and consistent
Example: As above, but also there is a basic simulation of the race, though most details, commentary etc. may be very simple

Level 4 –
Polymorphism should be used in at least three subclasses, and all constructions above
At least four major program classes will be implemented, with methods working and well designed,
Comments are clear and applied to class and method level consistently
Example: As above, but race simulation is more natural, there is a running commentary of the race with major events reported.

Level 5 –
Use of **ArrayLists or other classes from Java's Collection Framework** in all parts of the program, and all constructions above.
Exception handling is carried out appropriately in all parts of the program
Inheritance is correctly applied to all parts of the program.
Example: As above, but all types of cars and functionality will now be included in the simulation. The simulation is now mostly ruled by a basic **GUI**.

Level 6 –
Includes **file input and/or output**, and all constructions above
The simulation (including player movement) will be displayed on the GUI
Polymorphism will be fully implemented in all parts of the program
Example: As above, with a full GUI now controlling all aspects of the simulation; data should be read from files.

Level 7 –
Includes everything required for an A grade but also **something special** (using other more advanced constructs or algorithms, or something you just read up on yourself). Make it a program someone would really want to use!
Example: realistic animation, sound etc

Suggestion 4: A business software dashboard simulation

"Dashboard provides at-a-glance views of key performance indicators (KPIs) relevant to a particular objective or business process (e.g. sales, marketing, human resources, or production)" (from wikipedia). You should be familiar with basic business concepts for this project. You should have an underlying simulation of real world events that determine the evolution of the business in terms of materials acquired, products sold, hiring, forecasts etc...

Level 1 –
Includes **screen output** and **keyboard input** and **basic classes**.
There are java source files for at least three major classes in the program.
Good source comments and code indentation is expected for all implemented parts of the code
Example: The program reads and prints the names of basic dashboard items and their basic characteristics.

Level 2 –
Includes methods and variables for at least three major classes, and all constructions above.
At least 3 major methods fully implemented and working for each class
Example: As above, but also the notion of the hiring, firing people, buying, selling items, forecasting is shown.

Level 3 –
At least three major program classes will be implemented, with **methods working and well designed**, and all constructions above
Use of **inheritance** with at least one superclass and three subclasses
Class, method and variable naming will be clear and consistent
Example: As above, but also there is a basic simulation of the system, though most details, commentary etc. may be very simple

Level 4 –
Polymorphism should be used in at least three subclasses, and all constructions above
At least four major program classes will be implemented, with methods working and well designed,
Comments are clear and applied to class and method level consistently
Example: As above, but the simulation is more natural, there is a running commentary of the system with major events reported.

Level 5 –
Use of **ArrayLists or other classes from Java's Collection Framework** in all parts of the program, and all constructions above.
Exception handling is carried out appropriately in all parts of the program
Inheritance is correctly applied to all parts of the program.
Example: As above, but all items will now be included in the simulation. The simulation is now mostly ruled by a basic **GUI**.

Level 6 –
Includes **file input and/or output**, and all constructions above
The simulation (including player movement) will be displayed on the GUI
Polymorphism will be fully implemented in all parts of the program
Example: As above, with a full GUI now controlling all aspects of the simulation, displaying different dashboard elements; data should be read from files.

Level 7 –
Includes everything required for an A grade but also **something special** (using other more advanced constructs or algorithms, or something you just read up on yourself). Make it a program someone would really want to use!
Example: real data simulation and business transactions

==========

=======

ECS414U LABS Assessed Cover Sheet

NOTICE : THIS IS NOT THE OFFICIAL RECORD OF LAB ASSESSMENTS. THE OFFICIAL RECORD IS POSTED ON THE MODULE WEBSITE FEW DAYS AFTER EACH LAB. PLEASE CHECK THIS RECORD MATCHES THE OFFICIAL RECORD

Course Leaders: **Pasquale Malacaria, Matthew Huntbach**

Year: **2018**

Student Name: _____

Student Exam Number: _____

Lab Class Time: _____

	Date and Tutor Name and Signature		Date and Tutor Name and Signature
Short 1		Mini-proj 1	
Short 2		Mini-proj 2	
Short 3		Mini-proj 3	
Short 4		Mini-proj 4	
Short 5		Mini-proj 5	
Short 6		Mini-proj 6	
Short 7		Mini-proj 7	

This cover sheet must be handed in to the Computer Science Reception by:

12pm, last day of term

IMPORTANT: If you do not have the tutors signature on this sheet you have not got the grade.