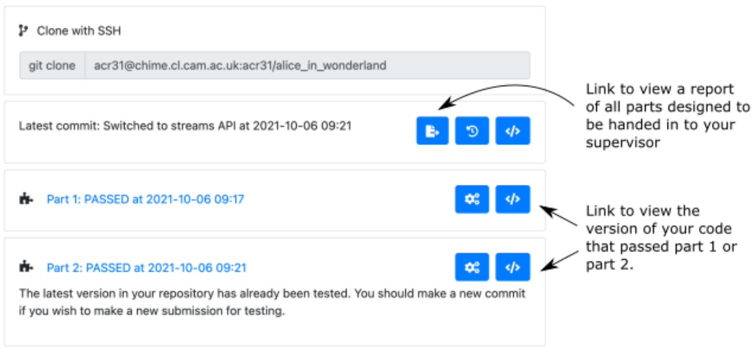
OOP Examples Sheet

Professor Andrew Rice
Michaelmas 2021
(Material originally written by Dr Robert Harle)

These exercises follow the notes and are intended to provide material for all three supervisions. For the majority of students this course has two challenges: the first is understanding the core OOP concepts; the second is applying them correctly in Java. The former can be addressed through traditional academic study; the latter requires a combination of knowledge and experience. To get the most out of this course you will need to put in effort programming *from scratch* (not just copy/pasting from StackOverflow..!).

Questions marked (**W**) are warm up questions. They mostly ask you to regurgitate the notes just to check your core understanding. Questions marked (*****) are intended to be more time consuming and/or challenging. Consult your supervisor for an appropriate set for you.

Practical exercises are mostly using the Chime service introduced in lectures. Chime can provide you with a report summarising the changes you made for each part of the exercise. I recommend you convert this to PDF and hand it in to your supervisor for marking.


Clone with SSH

```
git clone acr31@chime.cl.cam.ac.uk:acr31/alice_in_wonderland
```

Latest commit: Switched to streams API at 2021-10-06 09:21

Part 1: PASSED at 2021-10-06 09:17

Part 2: PASSED at 2021-10-06 09:21

The latest version in your repository has already been tested. You should make a new commit if you wish to make a new submission for testing.

Link to view a report of all parts designed to be handed in to your supervisor
Link to view the version of your code that passed part 1 or part 2.

For coding questions not on Chime consider starting your own repository on Github or the university Gitlab[1] service and sharing an appropriate link with your supervisor.

Workload expectations The department suggests that you should spend 3–4 hours doing work for a supervision. You should expect to spend more than that for this course! The reason for this is that this course includes 12 hours of timetabled practical classes in addition to 12 hours of lectures. This is therefore, 4 hours of lectures and another 4 hours of practical work for each supervision. Therefore your expectation should be to spend around 7–11 hours on work for each supervision. Please note that it is impossible to be precise with this number: some of you have considerable previous experience of the skills and concepts in this course and some of you do not.

Supervision 1

Video: Tasks with Chime

1. **(W)** Set up SSH, git, and IDE and your account on Chime

2. **(W)** Complete the Fibonacci task on Chime

(a) Why would one argue that the provided tests for Fibonacci are not sufficient?

(b) Why would it be a bad idea to write a test that times whether FibonacciTable runs faster than Fibonacci?

[1] `https://gitlab.developers.cam.ac.uk/`

3. What does instruction coverage mean as a test coverage metric? Give an example showing how 100% instruction coverage does not mean your program is bug free.

Video: Side-effects and void; Immutable values, returning and printing

4. **(W)** Compare and contrast a typical functional langauage and a typical imperative language.

5. Write Java code to test whether your Java environment performs tail-recursion optimisations or not.

Video: Control flow

6. Write a static function lowestCommon that takes two long arguments and returns the position of the first set bit in common, where position 0 is the LSB. If there is no common bit, the function should return -1. For example lowestCommon(14,25) would be 3. Your solution should use at least one break statement.

Video: Primitive types

7. **(W)** Identify the primitives, references, classes and objects in the following Java code:

```
double d = 5.0;
int i[] = {1,2,3,4};
LinkedList<Double> l = new LinkedList<Double>();
Double k = new Double();
Tree t;
float f;
Computer c = null;
```

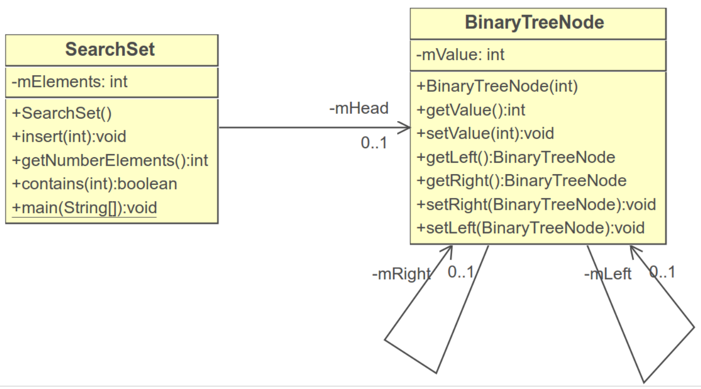
Video: Objects and classes

8. You met the idea of linked lists in FoCS. Complete the first part of the ‘Classic collections’ task on Chime.

(a) How is an empty list represented?
(b) How does the `toString` method implement the list traversal?
(c) Given that you are making use of the `addFirst` method why would it be considered better style to use a static factory method to construct a list rather than doing all the work in the constructor?
(d) Give the UML class diagram for your code.

9. **(*)** In mathematics, a *set* of integers refers to a collection of integers that contains no duplicates. You typically want to insert numbers into a set and query whether the set contains numbers. One approach is to store the numbers in a binary search tree.

(a) The diagram below represents this approach. Implement it in Java, using the names of entities to decide what they should do. Provide appropriate tests using JUnit.



(b) The BinaryTreeNode class can be reused for other solutions. Create a class FunctionalArray that uses BinaryTreeNode to create a functional array of ints. Your class should have a constructor that creates a tree of a given size (passed as an argument); a void set(int index, int value) method; and a int get(int index) method. You should make the functional array zero-indexed to match java's normal arrays (i.e. the first element has index 0). Requests for indices outside the limits should result in an exception.

Video: Constructors

10. **(W)** Explain why this code prints 0 rather than 7.

```
public class Test {
  public int x=0;
  public void Test() {
    x=7;
  }
  public static void main(String[] args)  {
    Test t = new Test();
    System.out.println(t.x);
  }
}
```

Video: Encapsulation; Access modifiers

11. What are the benefits of using private state in conjunction with public getter and setter methods? Are there any disadvantages?

12. Python does not offer a private access modifier and everything is public. Instead, programmers should follow the convention that a variable name should be prefixed with an underscore ('_') to signal it should not be accessed or edited externally. Discuss the advantages and disadvantages of this approach compared to Java.

Video: Immutability

13. Complete the Matrices programming task on Chime

(a) Why is it necessary for the assertions to have the form assertThat(actual).isWithin(tolerance).of(expected);?
(b) Why have the static factory methods been put in the Shapes class rather than Matrix?

14. This question considers representing a 2D vector in Java.

(a) Develop a mutable class Vector2D to embody the notion of a 2D vector based on floats (do not use Generics). At a minimum your class should support addition of two vectors; scalar product; normalisation and magnitude.
(b) What changes would be needed to make it immutable?
(c) Contrast the following prototypes for the addition method for both the (i) mutable, and (ii) immutable versions.
   - `public void add(Vector2D v)`
   - `public Vector2D add(Vector2D v)`
   - `public Vector2D add(Vector2D v1, Vector2D v2)`
   - `public static Vector2D add(Vector2D v1, Vector2D v2)`

(d) How can you convey to a user of your class that it is immutable?

Video: Parameterised types

15. **(W)** Explain in detail why Java's Generics do not support the use of primitive types as the parameterised type? Why can you not instantiate objects of the template type in generics (i.e. why is new T() forbidden?)

16. **(W)** Complete the second part of the 'Classic collections' task (using generics) on Chime.

17. Complete the Sorting task on Chime.

(a) When implementing Merge Sort you will need some temporary space for the merging part of the algorithm. You will find that you are unable to create new arrays of type T i.e. if you try `T[] temp = new T[10]` you will get a compile error. You can use a plain Object array or a new ArrayList⟨T⟩ instead. Bearing in mind the concept of type-erasure why does this fail to compile? Why does Object[] or ArrayList⟨T⟩ work?

18. **(*)** Research the notion of *wildcards* in Java Generics. Using examples, explain the problem they solve.

Video: References and pointers

19. **(W)** Pointers are problematic because they might not point to anything useful. A `null` reference doesn't point to anything useful. So what is the advantage of using references over pointers?

20. **(W)** Draw some simple diagrams to illustrate what happens with each step of the following Java code in memory:

```
Person p = null;
Person p2 = new Person();
p = p2;
p2 = new Person();
p = null;
```

Video: Summary

21. What would you say are the three most important concepts from the content for this supervision?

22. Give one question that you would like to discuss in the supervision.

Supervision 2

Video: Argument passing

23. **(W)** Explain the result of the following code.

```
public static void add(int[] xy,int dx, int dy) {
  xy[0]+=dx;
  xy[1]+=dy;
}

public static void add(int x,int y,int dx, int dy) {
  x=x+dx;
  y=y+dy;
}

public static void main(String[] args) {
  int xypair[] = {1,1};

  add(xypair[0], xypair[1], 1, 1);
  System.out.println(xypair[0]+" "+xypair[1]);

  add(xypair,1,1);
  System.out.println(xypair[0]+" "+xypair[1]);
}
```

Video: Inheritance; Inheriting fields and methods

24. **(W)** A student wishes to create a class for a 3D vector and chooses to derive from the Vector2D class (i.e. public void Vector3D extends Vector2D). The argument is that a 3D vector is a "2D vector with some stuff added". Explain the conceptual misunderstanding here.

25. **(W)** If you don't specify an access modifier when you declare a member field of a class, what does Java assign it? Demonstrate your answer by providing minimal Java examples that will and will not compile, as appropriate.

26. Suggest UML class diagrams that could be used to represent the following. Think carefully about the directions of and multiplicities on your arrows.

(a) A shop is composed of a series of departments, each with its own manager. There is also a store manager and many shop assistants. Each item sold has a price and a tax rate.
(b) Vehicles are either motor-driven (cars, trucks, motorbikes, electric bikes) or human-powered (bikes, skateboards,scooters). All cars have 3 or 4 wheels and all bikes have two wheels. Every vehicle has an owner. Some vehicles must have road tax.

27. Consider the Java class below:

```
package questions;

public class X {
  MODIFIER int value = 3;
};
```

Another class Y attempts to access the field value in an object of type X. Describe what happens at compilation and/or runtime for the range of MODIFIER possibilities (i.e. public, protected, private and unspecified) under the following circumstances:

(a) Y subclasses X and is in the same package;
(b) Y subclasses X and is in a different package;
(c) Y does not subclass X and is in the same package;
(d) Y does not subclass X and is in a different package.

Video: Abstract classes and abstract methods; Subtype polymorphism

28. **(W)** Explain what is meant by (dynamic) polymorphism in OOP and explain why it is useful, illustrating your answer with an example.

29. A programming language designer proposes adding 'selective inheritance' whereby a programmer manually specifies which methods or fields are inherited by any subclasses. Comment on this idea.

30. A Computer Science department keeps track of its CS students using some custom software. Each student is represented by a Student object that features a pass() method that returns true if and only if the student has all 20 ticks to pass the year. The department suddenly starts teaching NS students, who only need 10 ticks to pass. Using inheritance and polymorphism, show how the software can continue to keep all Student objects in one list in code without having to change any classes other than Student.

31. Complete the Chess practical exercise on Chime

(a) Which are the parts of the program which are poorly designed with switch statements for pieces?
(b) In what ways is your new code easier to maintain than previously? What drawbacks have arisen from this new approach?

32. In the second of the shape drawing examples in lectures, we relied on the existence of an instanceof operator that allowed us to check which class the object really was. The built-in ability to do this is called *reflection*. However, not all OOP languages support reflection. Show how we could modify the shape classes to allow us to determine the true type without using relection.

Video: Interfaces

33. **(W)** Explain the differences between a class, an abstract class and an interface in Java.

34. An alternative implementation of a list to a linked list uses an array as the underlying data structure rather than a linked list.

(a) Write down the asymptotic complexities of the array-based list methods.
(b) Complete part 3 of the 'Classic collections' (adding interfaces) task on Chime.
(c) When adding items to an array-based list, rather than expanding the array by one each time, the array size is often doubled whenever expansion is required. Analyse this approach to get the asymptotic complexities associated with an insertion.

35. **(*)**

(a) Complete parts 4 and 5 of the 'Classic collections' (queues) task on Chime.
(b) State the asymptotic complexities of the approach.

Video: Multiple inheritance

36. Imagine you have two classes: `Employee` (which embodies being an employee) and `Ninja` (which embodies being a Ninja). You need to represent an employee who is also a ninja (a common problem in the real world). By creating only one interface and only one class (`NinjaEmployee`), show how you can do this *without* having to copy method implementation code from either of the original classes.

Video: Object initialisation

37. **(W)** Write a small Java program that demonstrates constructor chaining using a hierarchy of three classes as follows: `A` is the parent of `B` which is the parent of `C`. Modify your definition of `A` so that it has exactly one constructor that takes an argument, and show how `B` and/or `C` must be changed to work with it.

Video: Object destruction and garbage collection

38. The lectures described the main mechanism to mark objects that can be deleted (or to mark objects that cannot be deleted), but not the deletion process. Compare and contrast the following approaches in terms of performance. Your answer should consider the costs associated with marking and deleting.

(a) The *mark-sweep* schemes delete the marked objects, leaving their memory available. A list of free memory chunks is maintained.
(b) The *mark-sweep-compact* schemes delete the objects and then compact the memory by moving surviving objects to be adjacent to each other (i.e. no free memory between them).
(c) The *mark-copy* schemes maintain two memory regions, one of which is active. During the marking process, surviving objects are copied into the other region. Once marking is complete, references are upated to the second region, which becomes active, and the first region is deleted.

39. It is often recommended that we make classes immutable wherever possible. How would you expect this to impact garbage collection?

40. What are the issues with using finalizers?

41. **(*)** In Java `try`...`finally` blocks can be applied to any code—no `catch` is needed. The code in the `finally` block is *guaranteed* to run after that in the `try` block. Suggest how you could make use of this to emulate the behaviour of a destructor (which is called immediately when indicate we are finished with the object, not at some indeterminate time later).

42. By experimentation or otherwise, work out what happens when the following method is executed.

```
public static int x() {
   try {return 6;}
   finally { ... }
}
```

Video: Summary

43. What would you say are the three most important concepts from the content for this supervision?

44. Give one question that you would like to discuss in the supervision.

Supervision 3

Video: Collections

45. **(W)** Using the Java API documentation or otherwise, compare the Java classes Vector, LinkedList, ArrayList and TreeSet.

46. **(*)** Java provides the List interface and an abstract class that implements much of it called AbstractList. The intention is that you can extend AbstractList and just fill in a few implementation details to have a Collections-compatible structure. Write a new class CollectionArrayList that implements a mutable Collections-compatible Generics array-based list using this technique. Comment on any difficulties you encounter.

Video: Iterators

47. Write a Java program that calculates the average (mean) for a list of integers. Provide three implmentations: 1) using a regular for-loop; 2) using a for-each loop; 3) using an iterator. What are the pros and cons of each?

Video: Comparing objects

48. **(W)** Explain why the following code excerpts behave differently when compiled and run (may need some research):

```
String s1 = new String("Hi");
String s2 = new String("Hi");
System.out.println( (s1==s2) );

String s3 = "Hi";
String s4 = "Hi";
System.out.println( (s3==s4) );
```

49. **(W)** Complete part 6 of the ‘Classic collections’ task on Chime.

50. **(W)** Write an immutable class that represents a 3D point (x,y,z). Give it a natural order such that values are sorted in ascending order by z, then y, then x

51. Write a Java class that can store a series of student names and their corresponding marks (percentages) for the year. Your class should use at least one Map and should be able to output a List of all students (sorted alphabetically); a List containing the names of the top P% of the year as well; and the median mark.

52. The user of the class Car below wishes to maintain a collection of Car objects such that they can be iterated over in some specific order.

```
  public class Car {
     private String manufacturer;
     private int age;
  }
```

(a) Show how to keep the collection sorted alphabetically by the manufacturer *without* writing a Comparator.

(b) Using a Comparator, show how to keep the collection sorted by {manufacturer, age}. i.e. sort first by manufacturer, and sub-sort by age.

53. **(*)** Write a Java program that reads in a text file that contains two integers on each line, separated by a comma (i.e. two columns in a comma-separated file). Your program should print out the same set of numbers, but sorted by the first column and subsorted by the second.

Video: Exception handling

54. **(W)** The following code captures errors using return values. Rewrite it to use exceptions.

```
public class RetValTest {
  public static String sEmail = "";

  public static int extractCamEmail(String sentence) {
    if (sentence==null || sentence.length()==0)
       return -1;  // Error - sentence empty
    String tokens[] = sentence.split(" "); // split into tokens
    for (int i=0; i< tokens.length; i++) {
      if (tokens[i].endsWith("@cam.ac.uk")) {
        sEmail=tokens[i];
        return 0; // success
      }
    }
    return -2; // Error - no cam email found
  }

  public static void main(String[] args) {
    int ret=RetValTest.extractCamEmail("My email is rkh23@cam.ac.uk");
    if (ret==0) System.out.println("Success: "+RetValTest.sEmail);
    else if (ret==-1) System.out.println("Supplied string empty");
    else System.out.println("No @cam address in supplied string");
  }
}
```

55. Write a Java function that computes the square root of a double number using the Newton-Raphson method. Your function should make appropriate use of exceptions.

56. Comment on the following implementation of pow, which computes the power of a number:

```
public class Answer extends Exception {
  private int mAns;
  public Answer(int a) { mAns=a; }
  public int getAns() {return mAns;}
}

public class ExceptionTest {
  private void powaux(int x, int v, int n) throws Answer {
    if (n==0) throw new Answer(v);
    else powaux(x,v*x,n-1);
    }

    public int pow(int x, int n) {
      try { powaux(x,1,n); }
      catch(Answer a) { return a.getAns(); }
      return 0;
    }
  }
```

Video: Covariance and contravariance

57. **(W)** Give an example of how covariant arrays in Java can create runtime errors.

Video: Inner classes and lambda

58. **(W)** Compare Inner-classes, method-local classes, anonymous inner classes and lambda functions. What general advice would you give someone who is trying to choose which one to use?

Video: Streams

59. Complete the Alice in Wonderland task on Chime

Video: Design Patterns; Decorator pattern

60. **(W)** Explain how Java uses the Decorator pattern with Reader.

Video: State pattern

61. **(W)** In lectures the examples for the State pattern looked at implementing a Fan with different speeds. Compare the different approaches taken. What were the advantages and disadvantages with them? In what sense is the final solution demonstrating the open-closed principle?

Video: Strategy pattern

62. Complete the Game of Life task on Chime

63. **(W)** Explain the difference between the State pattern and the Strategy pattern.

Video: Composite pattern

64. A drawing program has an abstract Shape class. Each Shape object supports a draw() method that draws the relevant shape on the screen (as per the example in lectures). There are a series of concrete subclasses of Shape, including Circle and Rectangle. The drawing program keeps a list of all shapes in a List<Shape> object.

(a) Should draw() be an abstract method?
(b) Write Java code for the function in the main application that draws all the shapes on each screen refresh.
(c) Show how to use the Composite pattern to allow sets of shapes to be grouped together and treated as a single entity.
(d) Which design pattern would you use if you wanted to extend the program to draw frames around some of the shapes? Show how this would work.

Video: Summary

65. What would you say are the three most important concepts from the content for this supervision?

66. Give one question that you would like to discuss in the supervision.