Workshop 3

Table of Contents

Information Hiding and Visibility

1. What does the term information hiding refer to? How do we achieve information hiding?
    - information hiding refers to the process of making class attributes or methods private to that class: this allows you to ensure that an object of that class is only modified in the intended manner. Doing this produces a clean interface (without extraneous/unnecessary helper methods/attributes), improving code maintainability and reliability.
    - every non-final attribute should be private (except in very unusual circumstances)
        - final attributes can't be changed in a way that can break the implementation; they are constant
    - encapsulation is one way of implementing information hiding
2. How do we perform information hiding (control the visibility) of our attributes and methods?
    - In Java, this is implemented with visibility modifiers.
        - `public`: available/visible everywhere
        - `private`: only available within a class; not inherited
        - `protected`: only visible within class, subclass, all classes in the same package, and subclasses in other packages
        - `default`: no modifier specified;
            * can be accessed within other classes in the same package, but not from outside the package
    - good practice: typically most attributes are `private`, most methods are `public`
        - methods that are only helpers can be `private`

3. What happens if we don't specify a visibility modifier?
    - default access
4. What does it mean for a class to be immutable?
    - immutable: a class has no methods that change instance variables
    - i.e. its state can't be modified
    - immutable classes make it simpler to use objects, and keep track of state of program
    - immutability is **not** the same as `final`; immutability is a property of a class

Casting and Wrapper Classes

1. What is the difference between implicit and explicit typecasting?
    - explicit: when you specify a typecast: e.g. `(int)'a';`
    - implicit: when type is coerced by Java, without explicit statement by the programmer, based on the behaviour requested; e.g. an `int` variable concatenated with a `String` (using +) will be coerced to a `String`
2. What is boxing and unboxing?
    - boxing is the process of placing a primitive within its wrapper class to extend its functionality
    - unboxing is the reverse of this process
3. Is boxing/unboxing done implicitly or explicitly?
    - modern versions of Java do automatic boxing/unboxing (i.e. implicit)

Strings

1. What is the expected output of the following code snippet?

```
String description = "Elliot Alderson";
description.replace("E", "e");  // doesn't alter description
description.replace("A", "a");  // doesn't alter description
description += ", 28";          // "Elliot Alderson, 28"
System.out.println(description);
// "Elliot Alderson, 28"
```

2. How do we check if two strings are equal?
    - using the `String.equals()` method, e.g. `a.equals(b)`

Arrays

1. What is an array?
    - fixed-length list with a homogeneous/uniform data type
        - NB in Java, not guaranteed to be contiguous block of memory; consider ”
    - in Java an array is an object
2. How do you declare and create an array?
    - `<type>[] arr = new <type>[<size>]{<init values>}`
    - e.g. `int[] nums = new int[5]{1, 2, 3, 4, 5};`
3. How do you perform the following operations with arrays?
    - Finding length: `arr.length`
    - Printing: `Arrays.toString(arr)`, note you will need to import `java.util.Arrays`
    - Sorting: `Arrays.sort(arr)`
    - Copying: `Arrays.copyOf(arr)`
    - Checking logical equality `Arrays.equals(a, b)`
4. How many instances of Person are created by this code snippet?

```
Person[] group = new Person[10];
```

0: It just creates an array that can store 10 references to `Person` instances These would have to be created manually e.g.

```
for (int i; i < group.length; i++) {
    group[i] = new Person(/*args*/);
}
```

NB Arrays in Java are clunky. They have a lot of limitations and are pseudo-objects There are much better options

IDE’s

1. What is an IDE? Why are they useful?
    - Integrated Development Environment: combines a set of tools for developing and managing software e.g. build, syntax highlighting, suggestions, warnings, debuggers, can connect to databases, tools to manage projects, file watchers, integrate with version control (e.g. git)

- Useful because all these tools are in one place and work together, allowing you to stay organised and not disrupt your workflow by changing between software

2. What is a project?
    - a project is a construct of an IDE that contains a set of files that define something you are working on (typically corresponding to a real-world project)
    - it will group together your source code, maintain configuration for your build, version control, etc.
3. Create a new project in IntelliJ, write some code, and run it. Your tutor will demonstrate how to do this.

Use IntelliJ to write solutions to these exercises.

1. Implement classes to represent the channels that air on TV. A channel has a name, and airs up to 5 shows throughout the day. A show has a name, and an air time (in 24 hour time). A channel can be queried to
   nd out what show is playing at a given time. If there is no show at that time, it should return null. Similarly, a channel can be queried to find out when a given show is playing. If the show is not on the channel, it should return null.
2. Add toString methods to the Channel and Show classes, and modify your program to print the channels and shows that have been added.
3. Add functionality to ensure that when channels are repeated, the channel stores each show (up to 5), not just one.

Implementation

1. To get a feel for your IDE, implement some solutions to tutorial problems (this week or last) in IntelliJ.
2. Aim to have finished the Grok worksheets by the end of this week.

Extra Questions

New content covered in this section is not examinable. 1. A linked list is a linear data structure that represents a collection of elements. A linked list is a collection of nodes. In a singly linked list (the most basic form of a linked list) each node consists of data, and a reference to the next node in the linked list. The last node will always point to a null reference.

```
graph LR
    1 --> 3
    3 --> 7
```

Figure 1: Visualisation of a singly-linked list

Create a class Node that represents a node in a singly linked list that stores integer values. 2. Build upon the design of the last question and implement a SinglyLinkedList class with methods that enable the following operations: - Add new value to end (public void add(int value)) - Check if value is present (public boolean contains(int value)) - Get number of elements (public int size()) This is not quite a ‘complete’ implementation of a singly linked list, so go ahead and feel free to challenge yourself and implement missing methods.