Java Notes

Camel Naming convention, second words must be capitals
Final modifier means the value of that variable cannot be changed.

Four Categories for Primitive Types

Integer Types

byte (-128 <> 127) – 8 Bits
short (-32768 <> 32767) – 16Bits
int (-2,147,483,648 <> 2,147,483,647) – 32Bits
long (……. You get it 2^63) – 64Bits

Floating Point Types

Float () – 32 Bits
Double ()- 64 Bits

Chat Types

Stores a single Unicode character
Literal value placed between quotes (Can use Unicode etc)

Boolean Type (Known)

Memory Allocation

```
int firstValue = 100;
int otherValue = firstValue;
firstValue = 50;
otherValue = 70;
```

Method Overloading

- Can have same name as another method, just needs a different parentheses

Control Flow Statements

- While
    - Will continue unless you add in a statement
    - **Continue** skips rest of code
    - **Break** will stop the loop

Reading User Input

- Scanner (java built in class) allows you to read in user input
- To handle enter key issue add an empty nextline() call

Object Orientated Programming

Classes

- Enable you to have a powerful user defined type
- Public (Unrestricted access to that class)
- Private (No other classes can access it)
- Protected (Allows classes in that package to access it)
- Allow us to create variables that are accessible anywhere in the class (Member Fields)
    - Specify access modifiers
    - General rule is to use private (Encapsulation (hides fields/methods from public access))
- To distinguish between fields in method type "This" to refer to a field in a class
- **Contructors**
    - You can set all the parameters to you want in one hit
    - Created automatically by java
    - You can have multiple contructors that contain a diff number of params.
    - The other constructors will call the main one.

Objects

- State
    - Age, Number of eggs etc
    - Software = fields
- Behaviour
    - Printing something, outputting sound etc
    - Methods

This vs super

- **Super** is used to access/call the parent class members
- **This** is used to call the current class members
- **Constructor chaining**
    - Don't repeat the same code
    - Make constructors call each other

Method Overriding vs Overloading

- **Method overloading**
    - Means proving two or more separate methods in the class with the same name but different parameters
    - Can also be treated as overloaded in a subclass of that class
    - Follow the rules:
        - Same name
        - Different parameters
- **Method Overriding**
    - Means defining a method in a child class that already exists in the parent class with the same signature
    - Known as **Runtime Polymorphism**
        - Because that method that is going to be called is decided at runtime by the JVM
    - Follow the rules:
        - Same name and arguments
        - Return type can be a subclass of the return type in the parent class
        - It cant have a lower access modifier
        - Constructors and private methods and final methods cannot be overridden
        - Only inherited methods can
        - A subclass can use super.MethodName(). To call the superclass version of an overridden method.

Static Methods

- Declared using a static modifier
- Static methods can access instance methods and instance variables directly.
- The are usually used for operations that don't require any data from an instance of the class (from 'this')
    - Cant us **this** keyword
- Whenever you see a method that does not use instance variables (Objects) that method should be declared as static

Instance Methods

- Belong to an instance of a class
- To use an instance method we have to instantiate the class first by using the "new" keyword
    - Can access directly:
        - Instance variables
        - Static methods
        - Static variables

Should a method be static?

- Does it use any fields (Instance variables or instance methods)
    - **Yes**
        - Be an instance method
    - **No**
        - Probably a static method

Static Variables

- Also known as static member variables
- Every instance of that class shares the same static variable
- If changes are made to it, it will effect all other instances
- Not used very often but can be useful
    - Declare a scanner as a static variable, that way all methods can access it directly.

Instance Variables

- Don't use the **static** keyword
- Instance variables are also known as fields/ member variables
- Instance variables belong to an instance of a class

Composition – Has a relationship with it (Parts of the greater model)

- Doesn't extend but does require.
- The monitor has a resolution class as part of its object
- A new class that has classes in its constructor
    - E.g. a PC that has a monitor, case, motherboard classes
    - Multiple inheritance (only extend one)

Encapsulation

- Preventing class/code from accessing the innerworkings of a class
- Give you more control/ change things without breaking code
- Fields within a class aren't accessible
- Making things less abusable

Polymorphism

- Allows an object to take on many forms.
- Only useful making classes within one java file if they are small and compact/not reused
- Assigning different functionally depending on what method is generated
- Movie (Loop through different movies)

- Complex functionality can be built into it.

Key Concepts

1. Inheritance
    a. (**extends** keyword) allows you to use the state behaviour of the other
    b. Super – call the constructor from the class we're extending from
        i. **Super**.**function**(will call the main classes method)
    c. You can add extra parameters to the inherited constructor
    d. Overide – will override a method in the super class
    e.
2. Polymorphism
3. Encapsulation
4. Composition

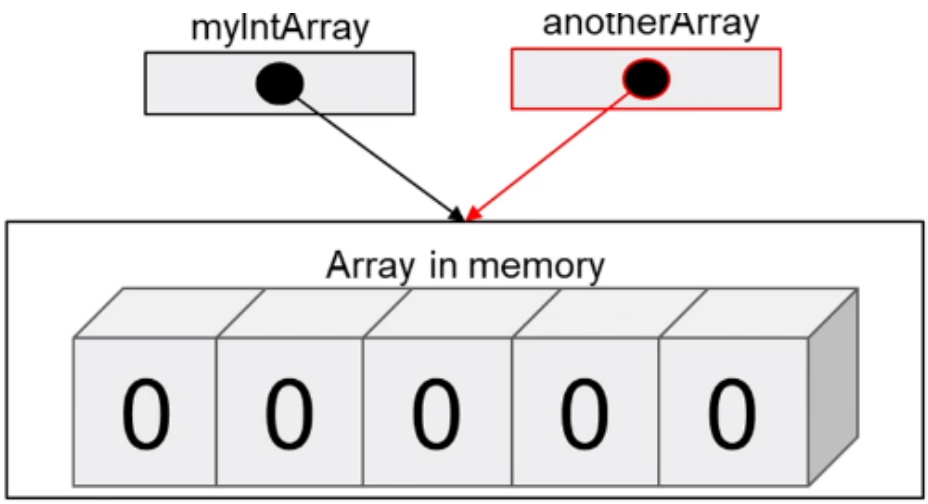
Arrays

- An array is a data structure that allows us to store multiple values of the same type into a single variable
- Indexed from 0.. n-1

Reference Types vs Value Types

- If both variables point to the same object they'll but update the same value

```
int[] myIntArray = new int[5];
int[] anotherArray = myIntArray;
```



Autoboxing

- Converting a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:
    - Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
    - Assigned to a variable of the corresponding **wrapper class**.

Unboxing:

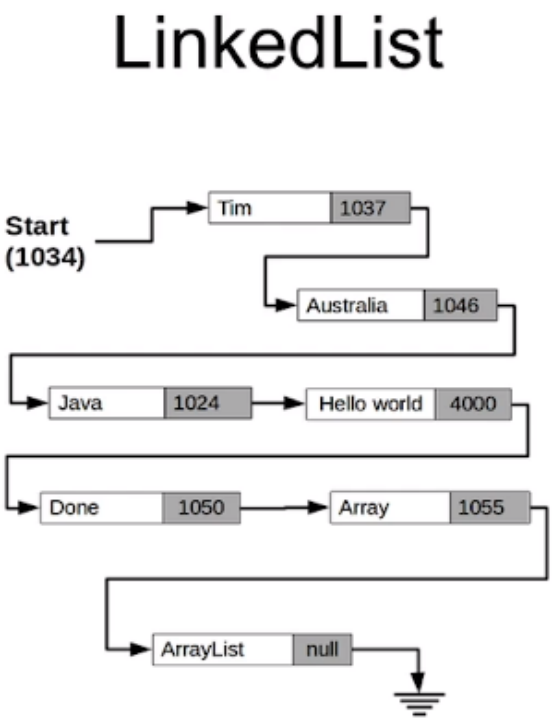
- Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example conversion of Integer to int. The Java compiler applies unboxing when an object of a wrapper class is:
    - Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
    - Assigned to a variable of the corresponding **primitive type**.

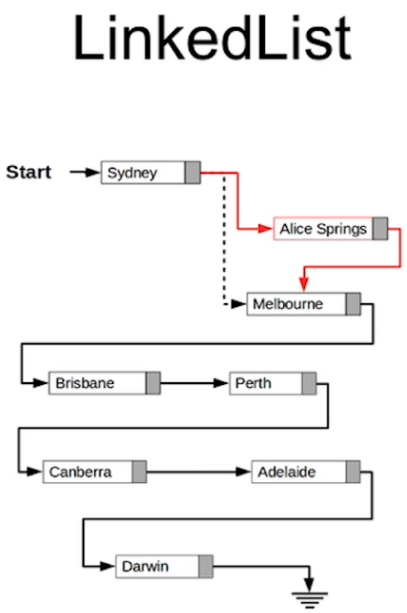
LinkedList

- Java can calculate the address of other integers
- Not efficient to add /remove items from an array List if it has lots of members as other members need to be moved. (Lots of manipulation)
- Each element in the list holds a link to the item that follows it.

LinkedList

Index	Address	Value
0	100	34
1	104	18
2	108	91
3	112	57
4	116	453
5	120	68
6	124	6



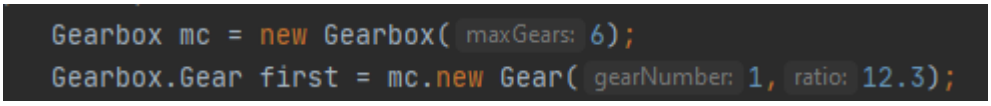


- Make Alice Spring point to Melbourne, Sydney points to Springs.
- If you remove an object java will collect it during garbage collection.

Interfaces

- Doesn't fill methods, just declares them with their name and parameters.
- In general, interfaces facilitate the key concepts in OOP like abstraction, inheritance and polymorphism. In addition, interfaces add flexibility and re-usability for software components.
- Interfaces allow you to fix this issue by editing methods within specific classes
- RULES:
    - Fields in an interface are public, static and final implicitly**:**
    - Methods in an interface are public implicitly
    - A class can implement multiple interfaces:
    - The overriding methods cannot have more restrict access specifiers
    - Abstract classes are not forced to override all methods from their super interfaces. The first concrete class in the inheritance tree must override all methods:
    - An interface cannot extend another class
    - An interface can extend from multiple interfaces:
    - An interface can be nested within a class:
    - An interface can be nested within another interface:
    - Methods in an interface cannot be static and final
    - Since Java 8, an interface can have default methods and static methods
    - Functional interface is an interface that has only one method:

Inner Classes

- Refrain from using the same variable names
- This. Refers to inside that specific class
- Must specific the outer class first
- 

- Inner classes can be private if you don't need to access them directly

Abstract Classes

- The **abstract** class states the class characteristics and methods for implementation, thus defining a whole interface.
- Interfaces are purely abstract
- You must determine how the classes are inheriting etc
- Can have constructs/variables
- Methods can have any visibility
- Methods can be defined/contain code
- If an abstract class is subclassed and you don't want to implement all the methods, you must declare that class as abstract
- The purpose of an abstract class is to provide a common definition of a bass class that multiple derived classes can share

Interfaces vs Abstract Classes

- Use abstract classes when you want to share code among several closely related classes (Animals with fields name, age etc.)
- You want non static or non-final fields
- You want to use public/protected/private modifiers
- Declaration of methods of a class
- Interfaces form a contract between a class and the outside world which is enforced at build time by the compiler

Generics

- Generics add stability to your code by making more of your bugs detectable at compile time.
- Fortunately, some bugs are easier to detect than others. Compile-time bugs, for example, can be detected early on; you can use the compiler's error messages to figure out what the problem is and fix it, right then and there.
- Runtime bugs, however, can be much more problematic; they don't always surface immediately, and when they do, it may be at a point in the program that is far removed from the actual cause of the problem.
- Arraylist<Integer> etc…..
- Finish off section
- Team <T extends player> Means it will accept any class the extends the class player

Name Conventions

- **Packages**
  - Always lower case
  - Should be unique
  - Use your internet domain name reversed
  - Co.uk.jackghawkins
- **Interface**
  - Capitalised
  - Consider what objects implementing it will become
- **Methods**
  - mixedCased
  - often verbs
- **Constants**
  - All Upper case
  - Separate words with underscored _
  - Use final keyword
- **Variable names**
  - mixedCase
  - meaningful
  - don't used underscores

Packages

- Programmers can easily determine that classes are related
- It is easy to know where to find the classes and interfaces that can provide the functions provided by the package
- Because the package creates a new namespace, class and interface name conflicts are avoided
- Classes within the package can have unrestricted access to one another
- Java.awt.***** brings everything from that package (*)

Access Modifiers

- Only classes, interfaces and enums can exist at the top level, everything else must be included with one of these
- Public access everywhere
- **Package-private**
  - When you don't specific a keyword
- Protected is visible anywhere within that package

The Static Statement

- You know what they do at this point

The Final Statement

- You know what they do..

Collections Overview

Binary Search - O(Log n).

- Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found, or the interval is empty.
- **Collections – List**
    - Has a variety of useful methods
        - Collections.Max
        - Collections.Reverse
        - Collections.Min
        - Collections.Shuffle

```
Theatre theatre = new Theatre("Hero",8,12);
List<Theatre.Seat> seatCopy = new ArrayList<>(theatre.seats); //
shallow copy
```

```
    /* Uses CompareTo Method*/
    Theatre.Seat minSeat = Collections.min(seatCopy);
    Theatre.Seat maxSeat = Collections.max(seatCopy);
    System.out.println(minSeat.getSeatNumber());
    System.out.println(maxSeat.getSeatNumber());
}
```

- You can't use the compare method without doing more work for price and seats etc. Example to follow

Maps

- .put (adds the reference)
- (Key ,Value)
- You can use .containsKey("Method") to prevent duplicates
- If you call sout(map.put("")) it will print null as its just been done
- Loop through map by doing

```
for(String key: languages.keySet()){
    System.out.println(key + " : " + languages.get(key));
}
```

- Standard methods Remove, Replace etc

Immutable Classes

- Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable
    - The class must be declared as final (So that child classes can't be created)
    - Data members in the class must be declared as final (So that we can't change the value of it after object creation)

    - A parameterized constructor
    - Getter method for all the variables in it
    - No setters(To not have the option to change the value of the instance variables

Sets

- The set interface present in the java.util package and extends the Collection interface is an unordered collection of objects in which duplicate values cannot be stored

HashSet

- Java **HashSet** class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
- The important points about Java **HashSet** class are
    - **HashSet** stores the elements by using a mechanism called hashing.
    - **HashSet** contains unique elements only.

https://www.codejava.net/java-core/the-java-language/everything-you-need-to-know-about-interfaces-in-java

https://www.hackerrank.com/interview/interview-preparation-kit?h_l=domains&h_r=hrw&utm_source=hrwCandidateFeedback

https://www.linkedin.com/learning

Look at getting java certified after course.