Part IB Further Java Notes

1 Sockets

- Client and server sockets, called `Socket` and `ServerSocket` respectively

  1. `Socket`
     - Constructed by passing the IP address as an `InetAddress` object and the port
     - Call `getInputStream` to return the input stream from the socket
     - Call `getOutputStream` to return the output stream from the socket
  2. `ServerSocket`
     - Constructed by passing the port that the server will listen on
     - The `accept` method blocks until a connection is made and will return the socket to the newly connected client.

2 Serialisation

- To be serialised class must implement the `Serializable` interface
- On serialisation Java will represent any primitives and recursively serialise all references
- Any nonserialisable state (eg. socket) must be marked as `transient` and will not be serialised.
- The JVM will create a UID for each class by hashing the definition
  - Allows the JVM to detect that classes are the same version
  - Can declare a custom UID by adding

    ```
    private static final long serialVersionUID = ...;
    ```

  - Declaring a custom UID is a good idea as adding/ deleting a fields can still be serialised (new fields initialised to default value, removed fields ignored).
  - `ObjectInputStream` and `ObjectOutputStream` are used to serialise and deserialise objects respectively.
    - Throws a `ClassNotFoundException` if the JVM cannot find a defintion for that class

3 Reflection

- Reflection permits inspection of classes at runtime
  - Calling `getDeclaredFields` returns a list of all fields
  - `getMethod(methodName)` returns a reference to the method

4 Annotations

- Annotations provide away to add metadata to a program in a way which is accessible to the program at runtime
- Prefix annotations with `@`
- Declare an annotation using the `@interface` keyword, eg.

```
public @interface MyAnnotation {
    int something;
    String somethingElse;
            ⋮
}
```

5 Multithreading

- Threads can either extend `Thread` or implement the `Runnable` interface
  1. Extend `Thread`
     - Implement the `run` method
     - Call the `start` method to run the run method in a new thread
  2. Implement `Runnable`
     - Create a class which implements the `Runnable` interface
     - Implement the `run` method
     - Create a new instance of the class `Thread` passing the newly created class object as an argument to the contractor
     - Call start on this instance of `Thread`
- Every thread has a name, if no name is provided then one will be generated, multiple threads may have the same name
- A class can be marked as a daemon, the JVM will only exit when all threads running are daemons
- Inner classes require external variables to be final (as they copy them instead of refering to the original)

5.1 Synchronized statement

```
synchronized (object) {
    // Some statements
            ⋮
}
```

- Java associates a lock with the parameter object
- Threads must acquire the lock before entering the block and release the lock when they leave
- Syntactic sugar

```
public void something() {
    synchronized (this) {
        // Some statements
                ⋮
    }
}
```

≡

```
public synchronized void
    something() {
    // Some statements
            ⋮
}
```

- If this is done to a static method then the lock will be associated with the class not the object

5.2 Wait and notify

- Calling wait on an object which you have the lock for will release the lock and pause execution
- Calling notify will wake you one waiting thread, the notifier must then release the lock before the waiter can aqurire it.