SESSION 9 - INTRODUCTION TO OBJECT ORIENTATED PROGRAMMING

DIGITAL JERSEY CODING PROGRAMME

WHAT IS IT?

- Design philosophy based on Objects and Classes. We organise software as a collection of objects that consist of both data and behaviour
- Models the world as a series of messages that pass between objects, rather than a set of predefined procedures
- OOP encourages
  - Modularisation - where are application can be decomposed into modules
  - Software reuse - Where an application can be composed from existing and new modules

WHY USE IT?

- In a world where you would never need to change code once written you don't need it.
- But something will always need to change, and when it does you want making changes to be as easy and painless as possible
- Object orientated design is about managing dependencies

OBJECTS

- An object is an instance of a class
- A class is the description of a concept, and an object is the realisation of this description
- There is one set of plans (the class), but there could be *n* number of objects created from that class
- Objects have their own identity and are independent of each other (e.g if we had a Television class, changing the channel on one instance of that class (an object) does not change the channel on all Televisions (objects)

CLASSES

- Blueprint / plan / template that describes the details of an object
- Have two components
  - Attributes - Instance variables in Ruby (@)
  - Methods (def)
- S.O.L.I.D is the acronym for the most well known principles of OOD design
  - Single responsibility principle (SRP)
  - Open-closed
  - Liskov Substitution
  - Interface Segregation
  - Dependency inversion

DESIGN CLASSES THAT DO ONE THING

- A class should do the the smallest possible useful thing, it should have a unique responsibility
- Why?
  - Applications that are easy to change consist of classes that are easy to reuse
  - A class that has more than one responsibility is difficult to reuse. You can't reuse some (but not all) of the behaviour

MANAGING DEPENDENCIES

- An object depends on another object if, when one object changes, the other might be forced to change in turn
- An object is dependent on another when it knows:
  - The name of another class
  - The name of a method it intends to send to another class
  - The arguments required, and the order of those arguments

MANAGING DEPENDENCIES

- Inject Dependencies
- Isolate dependencies
- Remove argument order dependencies
- Explicitly define defaults
- Depend on objects that are less likely to change

INTERFACES

- Classes should expose as little of themselves as possible
- We only need to understand the public interface that the class gives us
- Exposed methods comprises the *public interface* of a class
- A kitchen has a public interface (menu), it does many things but these are not exposed they are private

INTERFACES

- Public interface
  - Reveal a classes primary responsibility
  - Are expected to be invoked by others
  - Will not change much
  - Are documented and tested (ideally)
- Private Interface
  - Handle implementation details
  - Are not expected to be sent to other objects
  - Can change whenever
  - Are unsafe for others to depend on

DUCK TYPING

- The idea that it is not what an object is (its class) but what it does
- As long as the object responds to a message it doesn't matter what that object does

INHERITANCE

- Subclasses inherit the behaviour of their parents
- Ruby will first check the current class for the method, then it will works its way up through the superclasses

MORE ACRONYMS

- DRY (Don’t repeat yourself)
- LoD (Law of Demeter)

```
        hover-bg">
      <a href="#">
        <div class="hover-text">
          <h4>Logo Design</h4>
          <small>Branding</small>
          <div class="clearfix"></div
          <i class="fa fa-plus"></i>
        </div>
        <img src="images/portfolio/02.j
        img-responsive" alt="...">
      </a>
    </div>
  </div>
</div>

<div class="col-sm-6 col-md-3 col-lg-3 branding
  <div class="portfolio-item">
    <div class="hover-bg">
      <a href="#">
        <div class="hover-text">
          <h4>Logo Design</h4>
          <small>Branding</small>
          <div class="clearfix"></div
```

- SIMPLICITY IS KING
- CLASSES SHOULD HAVE A SINGLE RESPONSIBILITY
- SOFTWARE DEVELOPERS LIKE ACRONYMS

KEY TAKEAWAYS

- FINISH THE RUBY EXERCISES ON GITHUB
- READ OBJECT ORIENTATATED DESIGN IN RUBY BY SANDI METZ
- SIGN UP TO CODE WARS AND DO AS MANY RUBY EXERCISES AS POSSIBLE