Exercise 4 - Inheritance and Traits

Objective

The objective of this exercise is to expand on the classes we have been looking at in the previous exercise by using inheritance and traits.

References

Chapter four slides: 68-93

Overview

Standard Inheritance

1. Start a new worksheet
2. Create an Animal class.
   * The class should be abstract
   * The constructor should take in a name and an age
   * Override the toString method
   * Provide an abstract method called move that returns a string.
3. Create three subclasses of the animal class, cat, dog and rabbit
   * Your classes should call the parent constructor
   * Implement the move class for each animal differently (return a different string. For example, a rabbit could return “hop”)
   * Override the toString method for the class
4. Create some cat, dog and rabbit classes to test your inheritance. Test the toString methods.
5. Copy in your bank account class from the previous exercise (we don’t need the companion object)
6. Create a SavingsAccount which inherits from account
   * A savings account has a percentage interest rate and an addInterest method which increase the balance by that percentage

For example: If the account had a balance of £100 and an interest rate of 10% the addInterest method would increase the balance by 10% to £110

1. Create a current account which has an overdraft facility
   * Override the withdraw method to take into account the overdraft as well as the current balance in the account.

Traits

Traits can be applied to classes when the class is declared, or when a new object is created from that class. We will be looking at both of these situations.

1. Create a ‘vocal’ abstract trait that has a single method, makeNoise. This should return a string value
2. Add this trait to the animal class. All subclasses of animal will now need to implement the methods in the vocal trait. Add some implementation for these.

Example: abstract class Animal(…) extends Vocal { … }

1. Call the makeNoise method on your cat, dog and rabbit objects.
2. Remove the vocal trait from animal and add it directly to the Dog class. We can now call makeNoise on dog objects via the trait, but the others have become standard methods. Comment out the method in the cat and rabbit classes
3. Create a new cat object and add the Vocal trait at declaration time (rather than in the class description)
   * Does it work? Why not?
   * Create a concrete Vocal trait (provide some implementation for the makeNoise method) and apply it to a cat object at declaration time.
   * Call the makeNoise method on your new cat object
4. Implement the Logger trait
   * The logger trait should have a single method, Log which takes in a string and does nothing with it (slide 82 has an example)
5. Create a ConsoleLogger trait that inherits from the Logger trait
   * The console logger should print the messages to the screen
6. Add the Logger trait to your BankAccount class. Add some Log calls where you would like output to be.
7. Create a new BankAccount object and add the ConsoleLogger to it at declaration time. Test by calling your methods and seeing if the output is printed to the screen.
8. Now create a different logger, the TimeStampLogger
   * This works out the current date and time and prints this before printing the message to the screen
   * Hint: there are ways of getting the current date in scala, but you can also call classes and methods in Java as well!
9. Create a bank account that uses both the ConsoleLogger and TimeStampLogger

def b1 = new BankAccount(“123”, 100) with ConsoleLogger with TimeStampLogger

Do some deposit and withdraw calls and look at the output, then swap the order of traits and repeat this. When all the traits have the same parent trait the version of the method called is the one added to the class last, so in the example above, the TimeStampLogger method would be called, rather than the ConsoleLogger.

We can use this to override functionality in classes and traits even at object creation time.

1. Finally, we’re going to have a look at a built in trait called App.
   * Create a new scala class **(not a worksheet!)**
   * Change the class definition to be an object that extends App
   * Write anything you like in the body of the object
   * The App trait takes any commands in the body of the object and runs them as if they were in a methods, or being typed into the repl. Add as many scala commands as you like and then run it to see the output.

Hint: To see output you will need println() statements

Your object should look something like this:

object Ex4 extends App{  
  
 *println*("Anything here is run as individual statements due to the app trait")  
  
}