OOPs With Scala

cerenode.io

Classes

- Classes in Scala are blueprints for creating objects
- Contain methods, values, variables, types, objects, traits, and classes which are collectively called members.

```
class Car(name: String, price: Double, engineCC: Double) {
    ...
}
```

- Primary constructor is the implicitly introduced constructor
    - It also executes all the statements inside the class

Object

- Way to implement singleton in Scala
- Can be directly accessed using the name
- Uses
  - Create utility methods
  - Defining constants

```
object Car {
    val name = “City”
    val price = 1000000
    val engineCC = 1.3
}
```

Companion Object

- Special type of Object
- Same name as Class in the source file
- Companion Class can access private fields of the companion object
- Can be used for adding functions to case classes

Case Class

- Ease the development by avoiding lot of boilerplate code
- Can use Case Classes in Pattern Matching very easily
- No need to use new operator to create instances of a case class
- Automatically adds a default `toString` implementation
- Adds `copy` method by default

```
case Car(name: String, price: Float)
```

Inheritance

- Inherit methods and parameters from another class for code reusability
- A class that extends an abstract class must provide implementation of its all abstract methods
- Scala solves Deadly Diamond of Death inheritance pattern

Abstract Class

- Used to achieve abstraction
  - Hide complex implementation details and show only functionality to the user
- Instances of abstract class cannot be created
- Can have abstract methods and non-abstract methods as well

Trait

- Encapsulates method and field definitions, which can then be reused by mixing them into classes
- A class can mix in any number of traits
- Traits can be partially implemented

Pattern Matching

- Sequence of alternatives, each starting with the keyword `case`
- Each alternative includes a pattern and one or more expressions, which will be evaluated if the pattern matches
- Arrow symbol => separates the pattern from the expressions
- Generalization of switch from C/Java

Sealed Class

- A sealed class may not be directly inherited, except if the inheriting template is defined in the same source file
- subclasses of a sealed class can inherited anywhere

val, lazy val & def

- Every `val` inside a class is evaluated when an instance is created
- Every `def` is evaluated for each and every call to the method
- Every lazy val is evaluated when the val is accessed for the first time and not during the instance creation

Package Object

- Used to group function declarations and constants that can be used across the package
- No need to import the package object