# Scala Design Patterns

https://www.youtube.com/watch?v=tv-_1er1mWI&t=460s

## 1. Singleton

Scala leverages the `object` keyword for singletons.

```
object Database {
  def connect(): Unit = println("Connected to database")
}
```

UML: A Singleton UML diagram has a single class with a static accessor.

## 2. Factory Method

Scala uses companion objects for simple factories.

```
trait Animal
class Dog extends Animal
class Cat extends Animal

object AnimalFactory {
  def create(kind: String): Animal = kind match {
    case "dog" => new Dog()
    case "cat" => new Cat()
    case _ => throw new Exception("Unknown")
  }
}
```

UML: Factory Method has a Creator abstract class/interface and concrete implementations.

### 3. Builder

Builder patterns can be modelled using chained methods in Scala.

```
case class Person(name: String, age: Int)

class PersonBuilder {
  private var name = ""
  private var age = 0
  def setName(n: String): this.type = { name = n; this }
  def setAge(a: Int): this.type = { age = a; this }
  def build(): Person = Person(name, age)
}
```

UML: Builder separates object construction from its representation.

## Structural Patterns

### 4. Adapter

Scala's implicits make adapters simple.

```
class OldPrinter { def printOld(msg: String): Unit =
println(s"Old: $msg") }
class NewPrinter { def print(msg: String): Unit =
println(s"New: $msg") }

implicit class Adapter(printer: OldPrinter) {
  def print(msg: String): Unit = printer.printOld(msg)
}
```

UML: Adapter has an adaptee, and an adapter wrapping it to provide a desired interface.

### 5. Decorator

Scala traits with `extends`/`with` can provide decorators.

```
trait Printer { def print(msg: String): Unit }
class ConsolePrinter extends Printer { def print(msg:
String): Unit = println(msg) }
trait TimestampDecorator extends Printer {
  abstract override def print(msg: String): Unit =
super.print(s"${System.currentTimeMillis}: $msg")
}
val printer = new ConsolePrinter with TimestampDecorator
printer.print("Hello")
```

UML: Decorator wraps a class and adds responsibility.

## 6. Composite

Scala's case classes excel for composites.

```
sealed trait FileSystemElement { def getSize: Int }
case class File(name: String, size: Int) extends
FileSystemElement { def getSize = size }
case class Directory(name: String, children:
List[FileSystemElement]) extends FileSystemElement {
  def getSize = children.map(_.getSize).sum
}
```

UML: Composite has Component, Leaf, and Composite classes.

# Behavioral Patterns

## 7. Strategy

Pass functions or traits to abstract behavior.

```
trait PaymentStrategy { def pay(amount: Double): Unit }
class CreditCard extends PaymentStrategy { def
pay(amount: Double): Unit = println(s"Paid $amount with
Credit Card") }
```

```
class PayPal extends PaymentStrategy { def pay(amount:
Double): Unit = println(s"Paid $amount with PayPal") }

def checkout(strategy: PaymentStrategy, amount: Double):
Unit = strategy.pay(amount)
```

UML: Strategy pattern relates a Context to an abstract Strategy.

## 8. Observer

Scala event handling can use function lists or libraries.

```
class Event[T] {
  private var subscribers = List[T => Unit]()
  def subscribe(f: T => Unit): Unit = subscribers ::= f
  def publish(event: T): Unit =
subscribers.foreach(_(event))
}
```

UML: Observer has a central subject and multiple observers.

## 9. Command

Functions as objects are the norm.

```
trait Command { def execute(): Unit }
class PrintCommand(msg: String) extends Command { def
execute(): Unit = println(msg) }
```

UML: Command links an invoker, command, and receiver.

## More Resources & Complete Examples

- Large set of GoF patterns (with UML and Scala code): GitHub examples
- Full project with categorization: Scala Design Patterns on GitHub
- More on Scala-specific implementations and when to use them: Pavel Fatin Blog

For every pattern above, you can find a clear diagram and extended code sample at Refactoring.Guru (often in Java or pseudocode, but examples are easily adaptable to Scala). Many GitHub repositories listed provide both UML models and well-documented Scala code for every design pattern.