Functional vs OO programming

Table of contents

1. **History**
   1. **Functional languages**
   2. **Object Oriented languages**
   3. **Current state of the industry**
2. **Functional programming elements**
   1. **Immutability**
   2. **Functions**
   3. **Higher order functions**
   4. **Purity and side effects**
   5. **Referential transparency**
   6. **Benefits/disadvantages**
3. **Object Oriented programming elements**
   1. **Classes**
   2. **Inheritance**
   3. **Encapsulation**
   4. **Polymorphism**
   5. **Dynamic binding**
   6. **Relations -> passing state and modifying it internally**
   7. **Benefits/disadvantages**
4. **Functional Best Practices**
   1. **Function composition**
   2. **Side effect free**
   3. **Separation of pure/impure code**
   4. **ADTs and separating data from functions**
5. **Object Oriented Best Practices**
   1. **SOLID**
      1. **Single Responsibility Principle**
      2. **Open/Closed Principle**
      3. **Liskov Substitution Principle**
      4. **Interface segregation Principle**
      5. **Dependency Inversion Principle**
   2. **YAGNI**
   3. **High cohesion/low coupling**
   4. **Interfaces instead of implementation**
6. **Functional Design Patterns**
   1. **What it’s all about? Functions, Functions, Functions**
   2. **Variance/Covariance/Contravariance**
   3. **Type classes**
   4. **Functors**
      1. **Functor**
      2. **Bifunctor**
      3. **Multifunctor**
      4. **Profunctor**
      5. **Applicative Functor**
   5. **Arrow**
   6. **Monads**
      1. **Monad**
      2. **Free Monad**
      3. **IO Monad**
      4. **State Monad**
      5. **Costate Monad**
      6. **Comonad**
7. **Functional Design Patterns in Action**
   1. **Scala Cats**
   2. **Scalaz**
   3. **Haskell Standard Library**
8. **Object Oriented Design Patterns**
   1. **Builder**
   2. **Factory**
   3. **Singleton (??)**
   4. **Bridge**
   5. **Decorator**
   6. **Chain of responsibility**
   7. **Command**
   8. **Interpreter**
   9. **Memento**
   10. **Observer**
   11. **State**
   12. **Strategy**
   13. **Visitor**
9. **Object Oriented Design Patterns in Action**
   1. **Java Swing/FX**
   2. **Java Spring**
10. **Web Programming**
    1. **FRM – functional relational mapping**
       1. **What is it**
       2. **Slick**
       3. **Examples**
       4. **Why?**
    2. **ORM – objection-relational mapping**
       1. **What is it**
       2. **JDBC/JPA/Hibernate**
       3. **Examples**
       4. **Why?**
    3. **Web Services**
       1. **Functional Web services**
          1. **Scala with Akka**
          2. **Scala with Play**
          3. **Example – mail service or something similar**
       2. **Object Oriented Web Services**
          1. **Java with Spring**
          2. **Example – mail service or something similar**
    4. **Controllers** 
       1. **Functional Controllers**
          1. **Scala with Akka**
          2. **Example – how and come**
       2. **OO Controllers**
          1. **Java with Spring**
11. **Distributed systems**
    1. **History**
    2. **Current state of the industry – why is it needed**
    3. **Why choose Functional over OOP**