Senior Scala & ZIO Interview Questions with Answers

Q: What are type classes in Scala and how do they compare with OOP interfaces?

A: Type classes enable ad-hoc polymorphism, allowing you to define behavior for types without modifying them. Unlike interfaces in OOP which are tightly coupled to types, type classes in Scala allow for more flexible and decoupled design.

Q: Explain higher-kinded types and give a real-world use case.

A: Higher-kinded types abstract over type constructors (e.g., F[_]). Useful in defining abstractions like Functor or Monad.

Q: How do you handle concurrency in Scala without traditional threads?

A: Using libraries like Futures, ZIO, or Cats Effect which offer non-blocking, composable concurrency abstractions.

Q: How do implicits work in Scala 2 vs. givens in Scala 3?

A: Scala 2 uses implicits for type classes and conversions, while Scala 3 introduces 'given' and 'using' for clearer, more explicit context passing.

Q: What's the difference between Try, Either, and ZIO for error handling?

A: Try captures exceptions; Either captures typed errors; ZIO provides richer handling of both environment and error types.

Q: What are the three type parameters of ZIO[R, E, A] and what do they represent?

A: R is the environment, E is the error type, and A is the success type.

Q: How is ZIO different from cats.effect.IO?

A: ZIO has three type parameters for environment, error, and result, enabling more expressive modeling and dependency injection.

Q: How would you use ZLayer to manage dependencies in a ZIO app?

A: Create services as traits, implement them, and wrap them with ZLayer for injection. Compose them at the top-level with provide/provideSome.

Q: What are fibers in ZIO and how do they compare to threads?

A: Fibers are lightweight concurrency units managed by the ZIO runtime, cheaper than OS threads and designed for high scalability.

Q: How do you handle expected vs unexpected failures in ZIO?

A: Expected failures go in the E channel; defects go to the cause and are accessed via sandboxing.

Q: How does ZIO Test help with functional testing?

A: It allows pure, effectful testing, supporting dependency injection, time control, and mocking without side effects.

Q: How would you mock dependencies in ZIO?

A: Provide test implementations using ZLayer and inject them into the effect with provideLayer.

Q: How would you structure a large ZIO application?

A: Separate modules as traits, use ZLayer for wiring, and compose pure domain logic separately from side effects.

Q: What patterns do you follow for effectful domain logic in a purely functional way?

A: Model domain logic as pure functions, use ZIO for orchestration, and encapsulate effects at the boundary.

Q: What is ZManaged and when would you use it?

A: ZManaged is used to manage resources safely, ensuring acquisition and release, like working with file handles or DB connections.

Q: What's the difference between ZIO#fold, foldM, and foldZIO?

A: fold is for pure transformations; foldM/foldZIO are for effectful error/success handlers.

Q: What's the purpose of ZIO#refineToOrDie?

A: It narrows the Throwable to a specific exception type or dies if the exception doesn't match.

Q: How can you test time-dependent effects in ZIO?

A: Using TestClock, you can manually control and simulate the passage of time.

Q: What is a Layer in ZIO and what problems does it solve?

A: ZLayer is a managed way to handle dependencies and DI in functional applications, replacing manual wiring with safe composition.