1.Trait

Thursday, September 26, 2024 12:21 PM

What is a trait? When to use?
Short: A trait is like an interface but can hold concrete methods and fields.
Long: In Scala, a trait is a reusable unit of behavior that can be added to classes. Unlike interfaces in Java, traits can have concrete methods and fields. Traits are used to define object characteristics and allow for mixing behavior. You use traits when you want to share functionality between classes without inheritance.
Example:

```
trait Flyable {
    def fly(): Unit = println("Flying")
}
class Bird extends Flyable
```

2.Abstract Class

Thursday, September 26, 2024 10:00 PM

What is an abstract class?
Short: A class that cannot be instantiated and may contain abstract and concrete methods.
Long: An abstract class is a class that cannot be instantiated directly. It may include abstract methods (without implementation) and concrete methods. Abstract classes are used when you want to provide base functionality that can be extended by subclasses. Unlike traits, a class can extend only one abstract class, but multiple traits.

3.Java Interface and Traits

Thursday, September 26, 2024 10:02 PM

What is the difference between a Java interface and a Scala trait?
Short: Traits can have both concrete methods and fields, while Java interfaces can only have abstract methods (before Java 8).
Long: Scala traits differ from Java interfaces in that they can hold both concrete and abstract methods as well as fields. Java interfaces (before Java 8) could only contain abstract methods. Scala traits also support multiple inheritance, meaning a class can mix in multiple traits.

4.Singleton

Thursday, September 26, 2024 10:04 PM

What is a singleton?
Short: A class with only one instance throughout the application.
Long: A singleton in Scala is an object that has only one instance. It is used for single-instance requirements, like managing configurations or logging. In Scala, the object keyword is used to define a singleton.

5.Higher-order Function

Thursday, September 26, 2024 10:06 PM

What is a higher-order function?
Short: A function that takes or returns another function.
Long: A higher-order function is a function that either takes another function as an argument or returns a function as its result. They are commonly used in functional programming to apply functions dynamically or build more complex behavior.
Example:
def applyTwice(f: Int => Int, x: Int): Int = f(f(x))

6.Closure Function

Thursday, September 26, 2024 10:08 PM

What is a closure function?
Short: A function that captures variables from its enclosing scope.
Long: A closure function is a function that captures and references variables from its surrounding scope, even after that scope has closed. Closures are commonly used when a function needs to access non-local variables.
Example:

```
val factor = 3
val multiplier = (x: Int) => x * factor
```

7.Companion Object

Thursday, September 26, 2024 10:12 PM

What is a companion object? What are the advantages?
Short: A singleton object that shares the same name as a class and can access its private members.
Long: A companion object is a singleton object in Scala that shares the same name as a class and is defined in the same file. It can access the class's private fields and methods. The main advantage is that it allows for functionality that applies to the class as a whole (e.g., factory methods) without using static methods like in Java.
Example:

```
class MyClass(val name: String)
object MyClass {
   def apply(name: String): MyClass = new MyClass(name)
}
```

8.NILvsNullvsNonevsUnit

Thursday, September 26, 2024 10:15 PM

Nil vs Null vs null vs Nothing vs None vs Unit?
Short: Different representations of absence, emptiness, or failure in Scala.
Long:
Nil: Represents an empty list.
Null: Represents a reference that points to no object.
null: A keyword representing a null reference.
Nothing: The bottom type; no value is of this type.
None: A safe alternative to null, used with Option.
Unit: Represents no meaningful value, equivalent to void in Java.

9.Pure Function

Thursday, September 26, 2024 10:18 PM

What is a pure function?
Short: A function that has no side effects and always returns the same output for the same input.
Long: A pure function is a function where the output depends solely on the input parameters and has no side effects (i.e., it does not alter state outside the function). Pure functions are important in functional programming because they are easier to reason about and test.

10.SBT

Thursday, September 26, 2024 10:20 PM

What is SBT, and how have you used it?
Short: SBT is Scala's build tool, used for compiling, running, and testing Scala code.
Long: SBT (Simple Build Tool) is the default build tool for Scala. It handles project dependencies, compilation, running, and testing of Scala programs. You specify your project's dependencies in the build.sbt file, and SBT automatically downloads and manages them.

11.Currying

Thursday, September 26, 2024 10:24 PM

What is currying?
Short: Transforming a function that takes multiple arguments into a series of functions that take one argument.
Long: Currying is a functional programming technique where a function with multiple parameters is transformed into a series of functions that each take one parameter. This allows for partial application of functions and more flexible function composition.

12.Currying&Higher-order Fun

Thursday, September 26, 2024 10:26 PM

Difference between currying and higher-order functions?
Short: Currying transforms a function with multiple arguments; higher-order functions take or return other functions.
Long: Currying involves converting a function that takes multiple arguments into a chain of functions, each taking a single argument. Higher-order functions either take a function as a parameter or return a function as a result. Currying is a technique, while higher-order functions are more general and encompass any function that operates on other functions.

13.Var and Val

Thursday, September 26, 2024 10:28 PM

Difference between var and val?
Short: var is mutable; val is immutable.
Long: In Scala, var is used to define a variable whose value can change (mutable), while val defines a constant or a variable whose value cannot change after it is assigned (immutable). Use val when you want to prevent reassignment and ensure immutability in your code.

14.Case Class

Thursday, September 26, 2024 10:30 PM

What is a case class?
Short: A class that automatically provides boilerplate code like equals, hashCode, and pattern matching.
Long: A case class in Scala is a special type of class that automatically generates methods such as equals, hashCode, and toString. It also supports pattern matching, making it ideal for use in functional programming. Case classes are commonly used to represent immutable data.

15.Why Case Class

Thursday, September 26, 2024 10:32 PM

Why/when to use case class?
Short: Use for immutable data models with built-in methods.
Long: Case classes are useful when you need to represent immutable data. They automatically provide useful features like toString, equals, hashCode, and pattern matching, reducing boilerplate code. Case classes are commonly used in data modeling, especially in applications that rely on functional programming principles.
Example:
case class Person(name: String, age: Int)

16.Case&Normal Class

Thursday, September 26, 2024 10:33 PM

Difference between case class and normal class?
Short: Case classes provide immutability and pattern matching, while normal classes don’t.
Long: A case class automatically provides methods like equals, hashCode, toString, and copy, and supports pattern matching. Normal classes do not provide these features out of the box. Case classes are immutable by default, while normal classes allow for mutable state unless explicitly defined as immutable.

17.Type Hierarchy

Thursday, September 26, 2024 10:36 PM

Scala type hierarchy?
Short: All types in Scala inherit from Any, which has two subtypes: AnyVal (for value types) and AnyRef (for reference types).
Long: Scala's type hierarchy starts with Any, the root of all types. Any has two direct subclasses: AnyVal, which includes all value types (e.g., Int, Boolean), and AnyRef, which represents reference types (e.g., user-defined classes). The hierarchy also includes special types like Nothing (the bottom type) and Null.

18.Partially Applied Function

Thursday, September 26, 2024 10:37 PM

What are partially applied functions?
Short: A function that is applied with some arguments fixed in advance.
Long: A partially applied function is a function where some of the arguments are fixed (partially applied) and a new function is returned, which takes the remaining arguments. This is useful for reducing redundancy and simplifying code, as well as for function composition.

19.Tail Recursion

Thursday, September 26, 2024 10:39 PM

What is tail recursion?
Short: A recursive function where the final result is the result of the recursive call, optimized for memory.
Long: Tail recursion occurs when a recursive function makes its final result the result of its last recursive call. Scala can optimize tail-recursive functions to execute in constant stack space, preventing StackOverflowError. This makes tail recursion memory efficient for deeply recursive functions.
Example:

```
def factorial(n: Int, acc: Int = 1): Int = {
   if (n <= 1) acc
   else factorial(n - 1, n * acc)
}
```