**What is Primary Constructor? What is Secondary or Auxiliary Constructor in Scala? What is the purpose of Auxiliary Constructor in Scala? Is it possible to overload constructors in Scala?**

Scala has two kinds of constructors:

* Primary Constructor
* Auxiliary Constructor

**Primary Constructor**  
In Scala, Primary Constructor is a constructor which is defined with class definition itself. Each class must have one Primary Constructor: Either Parameter constructor or Parameterless constructor.

Example:-

|  |  |
| --- | --- |
| 1 | class Person |

Above Person class has one Zero-parameter or No-Parameter or Parameterless Primary constructor to create instances of this class.

|  |  |
| --- | --- |
| 1 | class Person (firstName: String, lastName: String) |

Above Person class has a two Parameters Primary constructor to create instances of this class.

**Auxiliary Constructor**  
Auxiliary Constructor is also known as Secondary Constructor. We can declare a Secondary Constructor using ‘def’ and ‘this’ keywords as shown below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Person (firstName: String, middleName:String, lastName: String){    def this(firstName: String, lastName: String){        this(firstName, "", lastName)    }  } |

**What is the use of Auxiliary Constructors in Scala?Please explain the rules to follow in defining Auxiliary Constructors in Scala?**

In Scala, The main purpose of Auxiliary Constructors is to overload constructors. Like Java, We can provide various kinds of constructors so that use can choose the right one based on his requirement.

**Auxiliary Constructor Rules:**

* They are like methods only. Like methods, we should use ‘def’ keyword to define them.
* We should use same name ‘this’ for all Auxiliary Constructors.
* Each Auxiliary Constructor should start with a call to previous defined another Auxiliary Constructor or Primary Constructor. Otherwise compile-time error.
* Each Auxiliary Constructor should differ with their parameters list: may be by number or types.
* Auxiliary Constructors cannot call a super class constructors. They should call them through Primary Constructor only.
* All Auxiliary Constructors call their Primary Constructor either directly or indirectly through other Auxiliary Constructors.

**NOTE:-** If you want to learn about Scala’s Constructors, please refer my Scala posts at: [Primary Constructor](http://www.journaldev.com/9810/scala-primary-constructor-indepth) and [Auxiliary Constructor](http://www.journaldev.com/9821/scala-auxiliary-constructors-in-depth).

**What are the differences between Array and ArrayBuffer in Scala?**

Differences between Array and ArrayBuffer in Scala:

* Array is fixed size array. We cannot change its size once its created.
* ArrayBuffer is variable size array. It can increase or decrease it’s size dynamically.
* Array is something similar to Java’s primitive arrays.
* ArrayBuffer is something similar to Java’s ArrayList.

**What is case class? What is case object? What are the Advantages of case class?**

Case class is a class which is defined with “case class” keywords. Case object is an object which is defined with “case object” keywords. Because of this “case” keyword, we will get some benefits to avoid boilerplate code.

We can create case class objects without using “new” keyword. By default, Scala compiler prefixes “val” for all constructor parameters. That’s why without using val or var, Case class’s constructor parameters will become class members, it is not possible for normal classes.

**Advantages of case class:**

* By default, Scala Compiler adds toString, hashCode and equals methods. We can avoid writing this boilerplate code.
* By default, Scala Compiler adds companion object with apply and unapply methods that’s why we don’t need new keyword to create instances of a case class.
* By default, Scala Compiler adds copy method too.
* We can use case classes in Pattern Matching.
* By default, Case class and Case Objects are Serializable.

**What is the difference between Case Object and Object(Normal Object)?**

* Normal object is created using “object” keyword. By default, It’s a singleton object.

|  |  |
| --- | --- |
| 1 | object MyNormalObject |

* Case Object is created using “case object” keywords.By default, It’s also a singleton object

|  |  |
| --- | --- |
| 1 | case object MyCaseObject |

* By Default, Case Object gets toString and hashCode methods. But normal object cannot.
* By Default, Case Object is Serializable. But normal object is not.

**When compare to Normal Class, What are the major advantages or benefits of a Case-class?**

The following are the major advantages or benefits of a Case class over Normal Classes:

* Avoids lots of boiler-plate code by adding some useful methods automatically.
* By default, supports Immutability because it’s parameters are ‘val’
* Easy to use in Pattern Matching.
* No need to use ‘new’ keyword to create instance of Case Class.
* By default, supports Serialization and Deserialization.

**What is the usage of isInstanceOf and asInstanceOf methods in Scala? Is there anything similar concept available in Java?**

Both isInstanceOf and asInstanceOf methods are defined in Any class. So no need import to get these methods into any class or object.

“isInstanceOf” method is used to test whether the object is of a given type or not. If so, it returns true. Otherwise returns false.

|  |  |
| --- | --- |
| 1  2  3  4 | scala> val str = "Hello"    scala>str.isInstanceOf[String]  res0: Boolean = false |

“asInstanceOf” method is used to cast the object to the given a type. If the given object and type are of same type, then it cast to given type. Otherwise, it throws java.lang.ClassCastException.

|  |  |
| --- | --- |
| 1  2 | scala> val str = "Hello".asInstanceOf[String]  str: String = Hello |

In Java, ‘instanceof’ keyword is similar to Scala’s ‘isInstanceOf’ method. In Java, the following kind of manual type casting is similar to Scala’s ‘asInstanceOf’ method.

|  |  |
| --- | --- |
| 1  2 | AccountService service = (AccountService)   context.getBean("accountService"); |

**How do you prove that by default, Case Object is Serializable and Normal Object is not?**

Yes, By Default, Case Object is Serializable. But normal object is not. We can prove this by using isInstanaceOf method as shown below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | scala> object MyNormalObject  defined object MyNormalObject    scala> MyNormalObject.isInstanceOf[Serializable]  res0: Boolean = false    scala> case object MyCaseObject  defined object MyCaseObject    scala> MyCaseObject.isInstanceOf[Serializable]  res1: Boolean = true |

**Difference between Array and List in Scala?**

* Arrays are always Mutable where as List is always Immutable.
* Once created, We can change Array values where as we cannot change List Object.
* Arrays are fixed-size data structures where as List is variable-sized data structures. List’s size is automatically increased or decreased based on it’s operations we perform on it.
* Arrays are Invariants where as Lists are Covariants.

NOTE:- If you are not sure about Invariant and Covariant, please read my next post on Scala Interview Questions.

**What is the difference between “val” and “lazy val” in Scala? What is Eager Evaluation? What is Lazy Evaluation?**

As we discussed in my Basic Scala Interview Questions, “val” means value or constant which is used to define Immutable variables.

There are two kinds of program evaluations:

* Eager Evaluation
* Lazy Evaluation

Eager Evaluation means evaluating program at compile-time or program deployment-time irrespective of clients are using that program or not.

Lazy Evaluation means evaluating program at run-time on-demand that means when clients access the program then only its evaluated.

The difference between “val” and “lazy val” is that “val” is used to define variables which are evaluated eagerly and “lazy val” is also used to define variables but they are evaluated lazily.

**What is the Relationship between equals method and == in Scala? Differentiate Scala’s == and Java’s == Operator?**

In Scala, we do NOT need to call equals() method to compare two instances or objects. When we compare two instances with ==, Scala calls that object’s equals() method automatically.

Java’s == operator is used to check References Equality that is whether two references are pointing to the same object or not. Scala’s == is used to check Instances Equality that is whether two instances are equal or not.

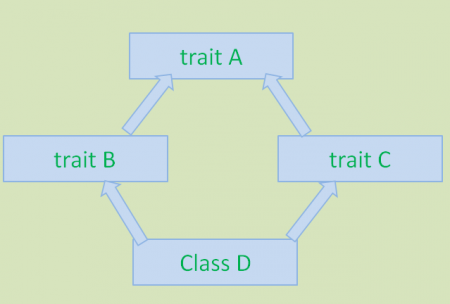
**Difference between Scala’s Inner class and Java’s Inner class?**

In Java, Inner class is associated with Outer class that is Inner class a member of the Outer class.  
Unlike Java, Scala treats the relationship between Outer class and Inner class differently. Scala’s Inner class is associated with Outer class object.

**What is Diamond Problem? How Scala solves Diamond Problem?**

A Diamond Problem is a Multiple Inheritance problem. Some people calls this problem as Deadly Diamond Problem.

In Scala, it occurs when a Class extends more than one Traits which have same method definition as shown below.



Unlike Java 8, Scala solves this diamond problem automatically by following some rules defined in Language. Those rules are called “Class Linearization”.

**Example:-**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | trait A{    def display(){ println("From A.display")  }  }  trait B extends A{    override def display() { println("From B.display") }  }  trait C extends A{    override def display() { println("From C.display") }  }  class D extends B with C{ }    object ScalaDiamonProblemTest extends App {      val d = new D      d display  } |

Here output is “From C.display” form trait C. Scala Compiler reads “extends B with C” from right to left and takes “display” method definition from lest most trait that is C.

**NOTE:-** See my post on “Scala Traits in Depth” to know this with clear explanation.

**Why Scala does NOT have “static” keyword? What is the main reason for this decision?**

As we know, Scala does NOT have “static” keyword at all. This is the design decision done by Scala Team.

The main reason to take this decision is to make Scala as a Pure Object-Oriented Language. “static” keyword means that we can access that class members without creating an object or without using an object. This is completely against with OOP principles.

If a Language supports “static” keyword, then that Language is not a Pure Object-Oriented Language. For instance, as Java supports “static” keyword, it is NOT a Pure Object-Oriented Language. But Scala is a Pure Object-Oriented Language.

**What is the use of “object” keyword in Scala? How to create Singleton objects in Scala?**

In Scala, object keyword is used the following purposes:

* It is used to create singleton object in Scala.

|  |  |
| --- | --- |
| 1 | object MySingletonObject |

Here, MySingletonObject becomes singleton object automatically.

* object keyword is used to define Scala Applications that is executable Scala programs.

|  |  |
| --- | --- |
| 1  2  3  4  5 | object MyScalaExecutableProgram{     def main(args: Array[String]){         println("Hello World")     }  } |

When we define main method in object as shown above (its same as main() method in Java), it becomes automatically as a executable Scala program.

* It is used to define static members like static variables and static methods without using ‘static’ keyword.

|  |  |
| --- | --- |
| 1  2  3  4 | object MyScalaStaticMembers{    val PI: Double = 3.1414    def add(a: Int, b: Int) = a + b  } |

By def PI variable and add methods will become as static members. That means we can call them without creating a separate object like MyScalaStaticMembers.add(10,20).

* It is used to define Factory methods. Please see my next question about this.

**How to define Factory methods using object keyword in Scala? What is the use of defining Factory methods in object?**

In Scala, we use ‘object’ keyword to define Factory methods. The main purpose of these Factory methods in Scala is to avoid using ‘new’ keyword. Without using ‘new’ keyword we can create objects.

**To define Factory methods:**  
We can use apply method to define Factory methods in Scala. If we have Primary Constructor and Multiple Auxiliary constructors, then we need to define multiple apply methods as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Person(val firstName: String, val middleName: String, val lastName: String){    def this(firstName: String, lastName: String){      this(firstName,"",lastName)    }  }  object Person{    def apply(val firstName: String, val middleName: String, val lastName: String)          = new Person(firstName,middleName,lastName)      def apply(val firstName: String, val lastName: String)          = new Person(firstName, lastName)  } |

Now we can create Person objects without using new keyword or with new keyword upto your wish.

|  |  |
| --- | --- |
| 1  2  3 | val p1 = new Person("Scala","Java")  or  val p1 = Person("Scala","Java") |

**What is apply method in Scala? What is unapply method in Scala? What is the difference between apply and unapply methods in Scala?**

In Scala, apply and unapply methods play very important role. They are also very useful in Play Framework in mapping and unmapping data between Form data and Model data.

In simple words,

* apply method: To compose or assemble an object from it’s components.
* unapply method: To decompose or dis-assemble an object into it’s components.

**Scala’s apply method:**  
It is used to compose an object by using its components. Suppose if we want to create a Person object, then use firstName and laststName two components and compose Person object as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Person(val firstName: String, val lastName: String)    object Person{    def apply(firstName: String, lastName: String)          = new Person(firstName, lastName)  } |

**Scala’s unapply method:**  
It is used to decompose an object into its components. It follows reverse process of apply method. Suppose if we have a Person object, then we can decompose this object into it’s two components: firstName and laststName as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Person(val firstName: String, val lastName: String)    object Person{    def apply(firstName: String, lastName: String)          = new Person(firstName, lastName)        def unapply(p: Person): (String,String)          = (p.firstName, p.lastName)  } |

**How does it work under-the-hood, when we create an instance of a Class without using ‘new’ keyword in Scala? When do we go for this approach? How to declare private constructors in Scala?**

In Scala, when we create an instance of a Class without using ‘new’ keyword, internally it make a call to appropriate apply method available in Companion object. Here appropriate apply method means that matched with parameters.

**When do we choose this option:** When we need to provide private private constructor and we need to avoid using ‘new’ keyword, we can implement only apply method with same set of parameters and allow our class users to create it without new keyword.

**How do we declare a private Primary Constructor in Scala? How do we make a call to a private Primary Constructor in Scala?**

In Scala, we can declare a private Primary Constructor very easily. Just define a Primary Constructor as it is and add ‘private’ just after class name and before parameter list as shown below:

|  |  |
| --- | --- |
| 1  2  3  4 | class Person private (name: String)  object Person{   def apply(name: String) = new Person(name)  } |

As it’s a private constructor, we cannot call it from outside. We should provide a factory method (that is apply method) as shown above and use that constructor indirectly.

**Does a Companion object access private members of it’s Companion class in Scala?**

Generally, private members means accessible only within that class. However Scala’s Companion class and Companion Object has provided another feature.

In Scala, a Companion object can access private members of it’s Companion class and Companion class can access it’s Companion object’s private members.

**What is the main design decision about two separate keywords: class and object in Scala? How do we define Instance members and Static members in Scala?**

In Scala, we use class keyword to define instance members and object keyword to define static members. Scala does not have static keyword, but still we can define them by using object keyword.

The main design decision about this is that the clear separation between instance and static members. Loosely coupling between them. And other major reason is to avoid static keyword so that Scala will become a Pure-OOP Language.

**What is object in Scala? Is it a singleton object or instance of a class?**

Unlike Java, Scala has two meanings about ‘object’. Don’t get confuse about this, I will explain it clearly. In Java, we have only one meaning for object that is “An instance of a class”.

* Like Java, the first meaning of object is “An instance of a class”.

|  |  |
| --- | --- |
| 1  2  3 | val p1 = new Person("Scala","Java")  or  val p1 = Person("Scala","Java") |

* Second meaning is that object is a keyword in Scala. It is used to define Scala Executable programs, Companion Objects, Singleton Objects etc.

**What is a Companion Object in Scala? What is a Companion Class in Scala? What is the use of Companion Object in Scala?**

In simple words, if a Scala class and object shares the same name and defined in the same source file, then that class is known as “Companion Class” and that object is known as “Companion Object”.

When we create a Class by using Scala “class” keyword and Object by using Scala “object” keyword with same name and within the same source file, then that class is known as “Companion Class” and that object is known as “Companion Object”.

Example:-  
**Employee.scala**

|  |  |
| --- | --- |
| 1  2 | class Employee{ }  object Employee{ } |

In Scala, The main purpose of Companion Object is to define apply methods and avoid using new keyword in creating an instance of that Companion class object.

**How to implement interfaces in Scala?**

As we know from Java background, we use interface to define contact.

However, there is no interface concept in Scala. Even, Scala doesn’t have interface keyword. Scala has a more powerful and flexible concept i.e. trait for this purpose.

**What is Range in Scala? How to create a Range in Scala?**

Range is a Lazy Collection in Scala. Range is a class available in ‘scala’ package like ‘scala.Range’. It is used to represent a sequence of integer values. It is an ordered sequence of integers.  
 **Example:-**

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> 1 to 10  res0: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)    scala> 1 until 10  res1: scala.collection.immutable.Range = Range(1, 2, 3, 4, 5, 6, 7, 8, 9) |

**How many values of type Nothing have in Scala?**

In Scala, Nothing type have no values that is zero. It does not have any values. It is a subtype of all Value classes and Reference classes.

**How many values of type Unit have in Scala?**

In Scala, Unit is something similar to Java’s void keyword. It is used to represent “No value exists”. It has one and only one value that is **()**.

**What is a pure function?**

A pure function is a function without any observable side-effects. That means it returns always same results irrespective how many times we call it with same inputs.

A pure function always gives same output for the same inputs.

For Example:-

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> 10 + 20  res0: Int = 30  scala>  scala> 10 + 20  res0: Int = 30 |

Here “+” a pure function available in Int class. It gives same result 30 for same inputs 10 and 30, irrespective how many times we call it.

**In FP, What is the difference between a function and a procedure?**

Both are used to perform computation, however they have one major difference in Functional Programming world.

A function is a computation unit without side-effect where as a Procedure is also a computation unit with side-effects.

**What are the major differences between Scala’s Auxiliary constructors and Java’s constructors?**

Scala’s Auxiliary constructor is almost similar to Java’s constructor with few differences.

Compared to Java’s constructors, Auxiliary constructors have the following few differences:

* The auxiliary constructors are called using “this” keyword.
* All auxiliary constructor are defined with the same name that is “this”. In Java, we use class name to define constructors.
* Each auxiliary constructor must start with a call to a previously defined auxiliary constructor or the primary constructor.
* We use ‘def’ keyword to define auxiliary constructors like method/function definition. In Java, constructor definition and Method definition is different.

**What is the use of ‘yield’ keyword in Scala’s for-comprehension construct?**

We can use ‘yield’ keyword in Scala’s for-comprehension construct. ‘for/yield’ is used to iterate a collection of elements and generates new collection of same type. It does not change the original collection. It generates new collection of same type as original collection type.

For example, if we use ‘for/yield’ construct to iterate a List then it generates a new List only.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> val list = List(1,2,3,4,5)  list: List[Int] = List(1, 2, 3, 4, 5)    scala> for(l <- list) yield l\*2  res0: List[Int] = List(2, 4, 6, 8, 10) |

**What is guard in Scala’s for-comprehension construct?**

In Scala, for-comprehension construct has an if clause which is used to write a condition to filter some elements and generate new collection. This if clause is also known as “Guard”.

If that guard is true, then add that element to new collection. Otherwise, it does not add that element to original collection.

**Example:-** For-comprehension Guard to generate only Even numbers into new collection.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> val list = List(1,2,3,4,5,6,7,8,9,10)  list: List[Int] = List(1, 2, 3, 4, 5 , 6 , 7 , 8 , 9 , 10)    scala> for(l <- list if l % 2 =0 ) yield l  res0: List[Int] = List(2, 4, 6, 8, 10) |

**How Scala solves Inheritance Diamond Problem automatically and easily than Java 8?**

If we use Java 8’s Interface with Default methods, we will get Inheritance Diamond Problem. Developer has to solve it manually in Java 8. It does not provide default or automatic resolution for this problem.

In Scala, we will get same problem with Traits but Scala is very clever and solves Inheritance Diamond Problem automatically using Class Linearization concept.

**In Scala, Pattern Matching follows which Design Pattern? In Java, ‘isinstanceof’ operator follows which Design Pattern?**

In Scala, Pattern Matching follows Visitor Design Pattern. In the same way, Java’s ‘isinstanceof’ operator also follows Visitor Design Pattern.

**What is the current latest version of Scala? What is the major change or update in Scala 2.12?**

Current Scala’s stable is 2.11.7. It supports Java SE 7.

The major change or update in Scala 2.12 version is that it supports Java SE 8 or later versions only. Scala 2.12 is not a binary compatible with the 2.11.x series. It’s still in Mile Stone Builds only.

**What is Option in Scala? What are Some and None? What is Option/Some/None Design Pattern in Scala?**

In Scala, Option is used to represent optional values that is either exist or not exist.  
Option is an abstract class. Option has two subclasses: Some and None. All three (Option, Some and None) are defined in “scala” package like “scala.Option”.

Option is a bounded collection in Scala, which contains either zero or one element. If Option contains zero elements that is None. If Option contains one element, that is Some.

Some is used to represent existing value. None is used to represent non-existent value.  
**Example:-**

|  |  |
| --- | --- |
| 1 | def get(val index: Int): Option[String] |

Let us assume that this method is from List. This method has a return type of Option[String]. If List contains elements, this get method returns “Some[String]” element available in that index position. Otherwise, it returns “None” (that is no elements)

Some is a case class and None is an Object. As both are case class/object, we can use them in Pattern Matching very well.

The combination of all these three definitions is known as Option/Some/None Design Pattern in Scala.

**What is Either in Scala? What are Left and Right in Scala? Explain Either/Left/Right Design Pattern in Scala?**

In Scala, Either is an abstract class. It is used to represent one value of two possible types. It takes two type parameters: Either[A,B].

It exactly have two subtypes: Left and Right. If Either[A,B] represents an instance A that means it is Left. If it represents an instance B that means it is Right.

This is known as Either/Left/Right Design Pattern in Scala.

**What is the equivalent construct of Scala’s Option in Java SE 8? What is the use of Option in Scala?**

Scala’s Option is similar to Java SE 8’s Optional. Java SE 8 has introduced a new utility class Optional to represent existing or non-existing of some value. Optional is available in java.util package.

Both Scala’s Option and Java SE 8’s Optional are used to represent optional values. Both are used to avoid unwanted null checks and NullPointerException.

**What are the Advantages of Functional Programming (FP) or Advantages of Pure Functions?**

The following are the Advantages of Functional Programming (FP) or Advantages of Pure Functions:

* More Modular
* Easier to understand Or Easier reason about
* Easier to test
* Less prone to bugs
* Easier to reuse
* Easier to Parallelism and generalize

**What are the Popular Scala-Based Frameworks to develop RESTful Web Services or REST API?**

There are many Scala-Based Framework to develop RESTful Web Services. Most popular frameworks are:

* Play Framework

In Play, we call REST API URLs as routes. We place all routes at once place in Play framework. It is a stateless web framework to develop REST API easily.

* Scalatra Framework

It is very simple and easy Scala-based web framework to develop REST API

* Spray Framework

It is very concise and built on top of Akka framework so it’s better to develop REST API using Actor Model.

* Lift Framework

It allows routing using Pattern Matching concept.

**What is the best Framework to generate REST API documentation for Scala-based applications?**

Swagger is is the best tool for this purpose. It is very simple and open-source tool for generating REST APIs documentation with JSON for Scala-based applications.

* If we use Play with Scala to develop your REST API, then use play-swagger module for REST API documentation.
* If we use Spray with Scala to develop your REST API, then use spray-swagger module for REST API documentation.

**Like Hibernate for Java-based applications, What are the Popular ORM Frameworks available to use in Play/Scala based applications?**

Like JPA, Hibernate and Toplink etc ORM Frameworks for Java-based applications, There are many ORM frameworks to use in Play/Scala based applications.  
Popular ORM frameworks for Play/Scala based applications:

* Slick
* Anorm
* SORM(Scala ORM)
* Squeryl

**What is the best tool to develop Play/Scala applications to persist data in MongoDB NoSQL data store?**

ReactiveMongo is the best Scala Driver to develop Play/Scala applications to persist data in MongoDB NoSQL data store. It supports fully non-blocking and asynchronous I/O operations.

**Popular clients who are using Play and Scala to develop their applications?**

Thousands of clients are using Play and Scala in Production. The following list is the more popular clients who are using Play and Scala actively.

* LinkedIn
* The Guardian
* Ocado
* LuchidChart
* GOV.UK

**What is the best language to use with Play framework: Scala or Java?**

Play 2 is completely written in Scala. If we use Java with Play framework, we need to face many issues because Java does not support full FP features.

Scala is the best option to use with Play framework to develop Highly Scalable, Better Performance with Concurrency/Parallelism and Low latency applications, because:

* Play 2 is completely written in Scala.
* It supports full FP features.
* It is more expression language than Java.
* It supports Akka Actor model very easily
* It supports some new OOP feature like Traits.
* Play’s built-in templates are developed in Scala

**How Scala supports both Highly Scalable and Highly Performance applications?**

As Scala supports Multi-Paradigm Programming(Both OOP and FP) and uses Actor Concurrency Model, we can develop very highly Scalable and high-performance applications very easily.

**What are the available Build Tools to develop Play and Scala based Applications?**

The following three are most popular available Build Tools to develop Play and Scala Applications:

* SBT
* Maven
* Gradle

**What is SBT? What is the best Build Tool to develop Play and Scala Applications?**

SBT stands for Scala Build Tool. Its a Simple Build Tool to develop Scala-based applications.

Most of the people uses SBT Build tool for Play and Scala Applications. For example, IntelliJ IDEA Scala Plugin by default uses SBT as Build tool for this purpose.

**What are the available Unit Testing, Functional Testing and/or BDD Frameworks for Play and Scala Based applications?**

The following are most popular available Unit Testing, Functional Testing and/or BDD Frameworks for Play/Scala Based applications:

* Spec2
* ScalaTest
* ScalaCheck
* Mokito

**What is the best Code-coverage tool available for Play and Scala based applications?**

SCoverage is the Code-coverage tool for Play and Scala based applications.

SCoverage stands for Scala Code-coverage tool. It has three separate plug-ins to supports the following build tools:

* SBT
* Maven
* Gradle

**What is the best Scala style checker tool available for Play and Scala based applications?**

Like Checkstyle for Java-Based Applications, Scalastyle is best Scala style checker tool available for Play and Scala based applications.

Scalastyle observes our Scala source code and indicates potential problems with it. It has three separate plug-ins to supports the following build tools:

* SBT
* Maven
* Gradle

It has two separate plug-ins to supports the following two IDEs:

* IntelliJ IDEA
* Eclipse IDE

**Which IDEs support Play and Scala-Based Applications Development and how?**

The following two popular IDEs support Play and Scala-Based Applications Development:

* IntelliJ IDEA
* Eclipse IDE

They support by using Scala Plugins like Eclipse IDE has a Scala IDE for Eclipse to support Play and Scala-Based Applications Development.

IntelliJ IDEA has a plug-in like “Scala Plugin for IntelliJ IDEA” to support “Scala, SBT and Play 2 Framework” based applications.

 What is the default Unit and Functional Testing Framework for Play? What is the default Build Tool for Play? What is the Default Template Engine for Play? What is the built-in Web Server available in Play Framework?

Play Framework’s default Unit and Functional Testing Framework is Spec2. It is very easy to test Play/Scala based applications using Spec2 Framework.

Play Framework’s Default built-in template is “Twirl”. It was developed in Scala. By using these templates, we can develop Play/Scala based applications very easily.

The Built-in or Default Web Server available for Play Framework is Netty Server.

**Why Scala is better than Java? What are the advantages of Scala over Java (Java 8)? Compare to Java What are the major advantages or benefits of Scala?**

Because Scala supports the following extra features, it is better than Java 8:

* Full FP Features
* More Expression Language
* Pattern Matching
* Better support for Akka Actor Model
* Automatic resolution for Inheritance Diamond Problem with Traits
* Asynchronous and Non-blocking IO programming using Akka Framework
* Fully Reactive Streaming API

**What is an Anonymous Function In Scala? What is a Function Literal in Scala? What are the advantages of a Anonymous Function/Function Literal in Scala?**

Anonymous Function is also a Function but it does not have any function name. It is also known as a Function Literal.  
**The advantages of a Anonymous Function/Function Literal in Scala:**

* We can assign a Function Literal to variable
* We can pass a Function Literal to another function/method
* We can return a Function Literal as another function/method result/return value.

**What is an Higher-Order Function (HOF)?**

Higher Order Function (HOF) is also a function but which performs one, two or both of the following things:

* Take other functions as arguments
* Return functions as their results

**What are the differences between Case class and Normal Class?**

Case class is also a class, however when we compare it with normal class, it gives the following extra features or benefits:

* By default, Case-class constructor parameters are ‘val’. We don’t need to declare parameters with ‘val’.
* By default, Case-class constructor parameters become class fields.
* These methods are added automatically: toString, equals, hashCode, copy. apply and unapply.
* It automatically gets Companion object.
* No need to use ‘new’ keyword to create instance of Case Class.
* Easy to use in Pattern Matching.

All these features are added by Scala Compiler at compile-time. It is not possible with normal class.

**What are the advantages of Play/Scala stack to develop web applications?**

The following are the major advantages of Play/Scala stack to develop web applications:

* Open Source

Play is an Open-source free-software framework to develop web applications.

* Better Productivity

Play framework’s Auto-reload feature improves Developer Productivity. No need to build, deploy and test our changes. Just do our changes and refresh the page to see our changes.

* Stateless and Easy to Develop REST API

Play is HTTP based stateless model to serve web requests so it is very easy to develop REST API or RESTful Web Services.

* Better Error-Handling

If we develop our web application using Play framework,it informs all errors in the browser in very useful format. It shows error message, the file location, line number where error occurred, highlighting the code-snippet to understand the error very easily.

* High Performance and Better Scalability With Reactive

Play framework is developed by following Reactive design patterns and it is built on top of Netty sever to utilize Non-blocking IO Feature. Because of this feature, we can develop very highly Scalable and performance applications very easily.

* Easy to Extend

Play is very flexible framework and supports developing plug-ins very easy to extend it’s features and functionality.

* Highly Concurrency and Better Parallelism

As both Scala and Play supports Functional Programming, it is very easy to develop Highly Concurrency and Better Parallelism applications very easily because FP supports Immutability, Pure Functions (Functions without side-effects), Pattern Matching, Actor Model etc.

* Better Reusability, Easy to Test and More Modular

As both Scala and Play supports Functional Programming, we can develop more modular and reusable applications. It is also very easy to test more modular applications.

**What are the Java’s OOP constructs not supported by Scala? What are the Scala’s OOP constructs not supported by Java? What are the new OOPs constructs introduced by Scala, but not supported by Java?**

Java’s OOP constructs, which are not supported by Scala:

* There is no interface concept in Scala
* There is no Enum concept in Scala

Scala’s OOP constructs, which are not supported by Java:  
OR  
The new OOPs constructs introduced by Scala, but not supported by Java:

* Scala Traits
* Solving Inheritance Diamond Problem automatically.

**What is call-by-name? Does Scala and Java support call-by-name? What is the difference between call-by-value and call-by-name function parameters?**

Call-by-name means evaluates method/function parameters only when we need them or we access them. If we don’t use them, then it does not evaluate them.

Scala supports both call-by-value and call-by-name function parameters. However, Java supports only call-by-value, but not call-by-name.

**Difference between call-by-value and call-by-name:**  
The major difference between these two are described below:

* In Call-by-name, the function parameters are evaluated only whenever they are needed but not when the function is called.
* In Call-by-value, the function parameters are evaluated when the function is called.
* In Call-by-value, the parameters are evaluated before executing function and they are evaluated only once irrespective of how many times we used them in that function.
* In Call-by-name, the parameters are evaluated whenever we access them and they are evaluated each time we use them in that function.
* Scala Syntax Differences

Call-by-value:

|  |  |
| --- | --- |
| 1 | def myFunction(a: Int, b: Int) { } |

Here both a and b are Call-by-value parameters to myFunction.

Call-by-name:

|  |  |
| --- | --- |
| 1 | def myFunction(a: Int, b: => Int) { } |

Here both a is a Call-by-value parameter and b is Call-by-name to myFunction.

**What are the popular MVC frameworks for Scala Language to develop Web Applications?**

The following are the most popular MVC frameworks available for Scala Language to develop Web Applications:

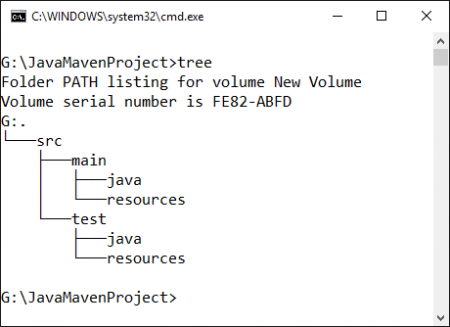
* Play Framework
* Scalatra Framework
* Spray Framework
* Lift Framework

**What are major differences between Java-Based and Scala-Based Maven Project’s structure?**

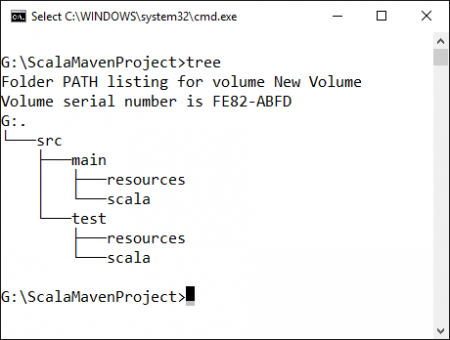
Most of the Java-based projects uses Maven as their Build tool. However, most of the people uses SBT as build tool to develop their Scala-based applications, but Some Teams uses Maven too as a build tool to develop their Scala-based applications.

Maven folder structure for both Java-based and Scala-based projects is almost same just one folder name change as shown in the below diagrams: java and scala folder names.

**Java-based Maven Project Folder Structure:**



**Scala-based Maven Project Folder Structure:**



**What is Extractor in Scala? What is the difference between Constructor and Extractor in Scala? What is the use of Extractor in Scala?**

Not only in Java and Scala, in almost all OOP languages Constructor is used to create (or assemble) an object or an instance of a Class using it’s parameters (or components). Extractor is quite opposite to Constructor.

In Scala, Extractor is used to decompose or disassemble an object into it’s parameters (or components).

In Scala, apply method is a Constructor. Internally, Extractor uses unapply method to decompose an objects into it’s parts (or parameters). In Scala, Extractor is mainly used in Pattern Matching concept. We will discuss Pattern Matching concept soon.

**What is the use of ‘???’ in Scala-based Applications?**

This ‘???’ three question marks is not an operator, a method in Scala. It is used to mark a method which is ‘In Progress’ that means Developer should provide implementation for that one.

This method is define in scala.PreDef class as shown below:

|  |  |
| --- | --- |
| 1 | def ??? : Nothing = throw new NotImplementedError |

If we run that method without providing implementation, then it throws ‘NotImplementedError’ error as shown below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> def add(a:Int, b:Int) : Int = ???  add: (a: Int, b: Int)Int    scala> add(10,20)  scala.NotImplementedError: an implementation is missing |

**Explain the main difference between List and Stream in Scala Collection API? How do we prove that difference? When do we choose Stream?**

In Scala, both List and Stream are from Collection API and works almost similar. Both are Immutable collections.

However, there is one main difference between List and Stream in Scala Collection API: That is List elements are evaluated Eagerly and Stream elements are evaluated Lazily that means when we access them.

|  |  |
| --- | --- |
| 1  2 | scala> var list1 = List(1,2,3,4)  list1: List[Int] = List(1, 2, 3, 4) |

Here we can observe that all List elements evaluated at the time of creating List object. However, if we do same thing on Stream, we cannot see all elements. We can see only first evaluated element and remaining elements are evaluated lazily as shown below:

|  |  |
| --- | --- |
| 1  2 | scala> var s1 = Stream(1,2,3,4)  s1: scala.collection.immutable.Stream[Int] = Stream(1, ?) |

When we want Lazy collection to evaluate elements only when we access them then it’s better to use Stream.

**What is the difference between :: and #:: in Scala? What is the difference between ::: and #::: in Scala?**

In Scala Collection API,

* **::** and **:::** are methods available in List class.
* **#::** and **#:::** are methods available in Stream class
* In List class, **::** method is used to append an element to the beginning of the list.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> var list1 = List(1,2,3,4)  list1: List[Int] = List(1, 2, 3, 4)    scala> list1 = 0 :: list1  list1: List[Int] = List(0, 1, 2, 3, 4) |

* In List class, **:::** method is used to concatenate the elements of a given list in front of this list.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> var list1 = List(3,4,5)  list1: List[Int] = List(3, 4, 5)    scala> val list2 = List(1,2) ::: list1  list2: List[Int] = List(1, 2, 0, 1, 2, 3, 4) |

* In Stream class, **#::** method is used to append a given element at beginning of the stream. Only this newly added element is evaluated and followed by lazily evaluated stream elements.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> var s1 = Stream(1,2,3,4)  s1: scala.collection.immutable.Stream[Int] = Stream(1, ?)    scala> s1 = 0 #:: s1  s1: scala.collection.immutable.Stream[Int] = Stream(0, ?) |

* In Stream class, **#:::** method is used to concatenate a given stream at beginning of the stream. Only this newly added element is evaluated and followed by lazily evaluated stream elements.

|  |  |
| --- | --- |
| 1  2  3  4  5 | scala> var s1 = Stream(1,2,3,4)  s1: scala.collection.immutable.Stream[Int] = Stream(1, ?)    scala> val s2 = Stream(-1,0) #::: s1  s2: scala.collection.immutable.Stream[Int] = Stream(-1, ?) |

* :: method works as a cons operator for List class and #:: method words as a cons operator for Stream class. Here ‘cons’ stands for construct.
* ::: method works as a concatenation operator for List class and #::: method words as a concatenation operator for Stream class.

**If I want to become a Fullstack Scala Developer, which technology stack I should learn?**

If you want to become a Fullstack Scala Developer, you should learn the following technology stack:

* Scala 2.11.7
* Play 2.4.6 Framework
* Akka 2.3 Framework
* One Build Tool: SBT/Maven
* One JS Framework: CoffeeScript/JavaScript
* One IDE: IntelliJ IDEA 15/ Eclipse IDE 4.x
* One TDD & BDD Framework: ScalaTest,Spec2,ScalaCheck,Mockito
* Micro Services with Play and Scala
* SCoverage
* Scalastyle
* Functional Programming Design Patterns
* Machine Learning with Scala

NOTE:- In Scala, Extractor follows Extractor Design Pattern. If you want to learn it in depth, please go through my Scala Tutorial (Most of the posts follows this pattern: Scala xxxx In Depth where xxxx is a concept like Extractor).

### 1. Why Sequential Programming is Bad?

These are the main drawbacks of Sequential Programming Model:

* Blocking Programming
* Wasting of Valuable Resources
* In-Efficient and In-Effective Use of Valuable Resources
* Synchronous Programming Model
* Less Through Put
* Bad Performance
* More Waiting time
* Slow Response to the Client Requests

### 2. Why Concurrency Programming is Good?

These are the main advantages of Concurrency Programming Model:

* Supports Non-Blocking Programming Model
* Efficient and Effective Use of Valuable Resources
* Asynchronous Programming Model
* More Through put
* Better Performance
* No Waiting time
* Instance Response to the Client Requests

### 3. What are the two main categories of Concurrency Models available in JVM?

There are two main categories in Concurrency Models:

* Shared Sate Concurrency Model
* No-Shared Sate Concurrency Model

1. Actors Based Concurrency Model
2. Future/Promise Concurrency Model

### 4. What are the main Advantages of Shared State Concurrency Model?

This Shared State Concurrency Model have the following few advantages:

* May be fast (If Developer is good in Understanding and Writing Multi Threading code)
* Using this model for longer time

### 5. What are the main Drawbacks of Shared State Concurrency Model?

This Shared State Concurrency Model have the following drawbacks:

* More difficult to write and understand the correct code
* Lot of boiler plate code
* Lots of Synchronization code to avoid race conditions
* Does not scale much in Distributed applications
* Tough to avoid Deadlock issues
* Tough to avoid Starvation issues
* Tough to avoid Live Locks
* Debugging is very tough
* Unit testing is bit tough
* More Resources utilization
* Very low level Concurrency Programming
* Lot of code to get and release Locks
* A Thread cannot create another Thread
* Threads are Heavy Weight
* Does not provide better Performance
* Supports only Concurrency, does NOT support True Parallelism.
* Does not support Non-Blocking functionality
* Context Switching overhead to support Concurrency

### 6. What are the main Advantages of Akka/Scala Actors Concurrency Model or Scala's Future/Promise Model?

This Actor Based Concurrency Model have the following few advantages:

* High Level Concurrency Programming
* No boiler plate code
* Easy to write Highly Scalable code
* Supports Distributed applications very well
* By default, Actors are Distributed
* More Type Safe
* No race conditions or dead lock issues
* Easy to avoid Starvation and Live Locks
* No Shared State
* Less Resources utilization
* An Actor can create or manager another Actor
* Unit Testing is very easy
* Easy to write and understand correct code
* No Locks and No Synchronization code
* Actors are Light weight
* Supports High Performance
* Supports both Concurrency and True Parallelism very well
* Supports Non-Blocking functionality
* No Context Switching overhead

**7. How Akka Actor Concurrency Model works?**

In Akka/Scala Framework, Actor Concurrency model works

* By using Actors but not Objects
* Each Actor has it's MailBox
* No Direct communication between Actors
* One Actor send messages to another Actor Asynchronously
* Asynchronous Message Passing
* No Share State that means Actors never share their state with other Actors
* No Locking techniques. No competition to get locks
* An Actor's state can only change by itself
* Actor processes it's messages one by one only. That means one message at a time.

**8. What are the main Drawbacks of Actors Based Concurrency Model?**  
  
In Akka/Scala Framework, Actor Concurrency model have the following few drawbacks or disadvantages or limitations:

* When our application needs shared state
* When our application needs Synchronous Communication
* When we cannot break our problem into smaller problems

Only in the above few cases, we cannot use Actor model.

**9. How do we implement Shared State Concurrency Model in Java?**  
  
In Java, we implement Thread-based Shared state concurrency model by using

* Shared Mutable state
* Locking mechanism

**10. How do we implement Locking Mechanism of  Shared State Concurrency Model in Java?**

In Java, we implement Locking Mechanism of Shared state concurrency model by using one of the following two things

* Synchronization Block
* Locking API

### 11. Why do we need to implement Locking Mechanism or Synchronization Block for Shared State Concurrency Model in Java?

In Java, we implement Locking Mechanism or Synchronization Block for Shared state concurrency model to get the following two things

* Execute the code in the correct order
* To get correct visibility
* To avoid Data Inconsistency issues
* To avoid Dead Lock issues
* To avoid Starvation
* To avoid Live Locks

### 12. What is a Future in Scala?

In Scala, a Future is an object which represents a computation unit. It have some value at some point in future.  
  
In other words, it is a programming construct in Scala used to write Concurrency programs very easily.  
  
  
When a Future finishes it's computation, we call it as Future is Completed. However it may be completed either successfully or unsuccessfully.  
  
If a Future is completed successfully, it has some computed value.  If its failed or completed unsuccessfully, it has some error.  
  
**NOTE:-** Please go through my "[**Scala Future and Promise In Depth**](http://rams4java.blogspot.co.uk/2017/02/scala-future-and-promise-api.html)" tutorial to understand them very well.

**Example-1:-**

myFutureObj.onComplete {

case Success(value) => // do something

case Failure(error) => // Log error

}

### 13. What is a Promise in Scala?

In Scala, a Future is represents a Computation unit. A Promise is used to execute that Computation unit and write the result into a Future.  
  
That mans, Promise is used to compute a value asynchronously and Future is used to read that value. One Promise is associated with only one Future.

**NOTE:-** Please go through my "[**Scala Future and Promise In Depth**](http://rams4java.blogspot.co.uk/2017/02/scala-future-and-promise-api.html)" tutorial to understand them very well.

### 14. Scala Future and Promise API?

Please go through my "[**Scala Future and Promise In Depth**](http://rams4java.blogspot.co.uk/2017/02/scala-future-and-promise-api.html)" tutorial to understand them very well.

### 15. Differences between Scala Future and Promise?

**In Scala,**Future and Promise have the following differences

* Promise is used to compute a value
* Future is used to read that value
* Promise is used write the value
* Future is read-only object.
* Once Promise is completed, we cannot change it's value. It become Immutable.
* Once Future is completed, we cannot change it's value. It become Immutable.
* Future is for Querying data and Promise is for Writing Data.
* In Simple words, Future is getter and Promise is setter.

### 16. What is ExecutionContext in Scala?

**In Scala, as we discussed** a Future is represents a Computation unit. A Promise is used to execute that Computation unit and write the result into a Future.  
  
ExecutionContext is a place where our Future's Computation unit is executed.  
Scala Language gives us a ready-made static global ExecutionContext in scala.concurrent.ExecutionContext.global.  
  
  
**NOTE:-**In simple words, ExecutionContext is something similar to a Thread Pool (Java Guys also can understand this).