

[JAVA TUTORIAL](#)[#INDEX POSTS](#)[#INTERVIEW QUESTIONS](#)[RESOURCES](#)[HIRE ME](#)[DOWNLOAD ANDROID APP](#)[CONTRIBUTE](#)**SIGN UP TO RECIEVE OUR UPDATES VIA EMAIL****SUBSCRIBE**[HOME](#) » [INTERVIEW QUESTIONS](#) » [SCALA INTERVIEW QUESTIONS](#)

# Scala Interview Questions

APRIL 2, 2018 BY [RAMBABU POSA](#) — [12 COMMENTS](#)

Before reading this post, please go through my previous post at "[Scala Basic Interview Questions and Answers](#)" to get some basic knowledge about Scala Language.

In this post, we are going to discuss about some more Scala Interview Questions which are useful for some experienced Scala Developers.

**Note:-** As this list has already become very large, I'm going to deliver another post with remaining Questions and Answers. Please refer that post at: "[Scala Intermediate and Advanced Interview Questions and Answers](#)"

## Table of Contents [\[hide\]](#)

### 1 [Scala Interview Questions](#)

#### [1.1 Scala Interview Questions and Answers](#)

[1.2 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?](#)

[1.3 What is the use of Auxiliary Constructors in Scala? Please explain the rules to follow in defining Auxiliary Constructors in Scala?](#)

[1.4 What are the differences between Array and ArrayBuffer in Scala?](#)

[1.5 What is case class? What is case object? What are the Advantages of case class?](#)

[1.6 What is the difference between Case Object and Object\(Normal Object\)?](#)

- 1.7 When compare to Normal Class, What are the major advantages or benefits of a Case-class?
- 1.8 What is the usage of `InstanceOf` and `asInstanceOf` methods in Scala? Is there anything similar concept available in Java?
- 1.9 How do you prove that by default, Case Object is Serializable and Normal Object is not?
- 1.10 Difference between Array and List in Scala?
- 1.11 What is the difference between "val" and "lazy val" in Scala? What is Eager Evaluation? What is Lazy Evaluation?
- 1.12 What is the Relationship between `equals` method and `==` in Scala? Differentiate Scala's `==` and Java's `==` Operator?
- 1.13 Difference between Scala's Inner class and Java's Inner class?
- 1.14 What is Diamond Problem? How Scala solves Diamond Problem?
- 1.15 Why Scala does NOT have "static" keyword? What is the main reason for this decision?
- 1.16 What is the use of "object" keyword in Scala? How to create Singleton objects in Scala?
- 1.17 How to define Factory methods using object keyword in Scala? What is the use of defining Factory methods in object?
- 1.18 What is apply method in Scala? What is unapply method in Scala? What is the difference between apply and unapply methods in Scala?
- 1.19 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?
- 1.20 How do we declare a private Primary Constructor in Scala? How do we make a call to a private Primary Constructor in Scala?
- 1.21 Does a Companion object access private members of it's Companion class in Scala?
- 1.22 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?
- 1.23 What is object in Scala? Is it a singleton object or instance of a class?
- 1.24 What is a Companion Object in Scala? What is a Companion Class in Scala? What is the use of Companion Object in Scala?
- 1.25 How to implement interfaces in Scala?
- 1.26 What is Range in Scala? How to create a Range in Scala?
- 1.27 How many values of type `Nothing` have in Scala?
- 1.28 How many values of type `Unit` have in Scala?
- 1.29 What is a pure function?
- 1.30 In FP, What is the difference between a function and a procedure?
- 1.31 What are the major differences between Scala's Auxiliary constructors and Java's constructors?
- 1.32 What is the use of 'yield' keyword in Scala's for-comprehension construct?
- 1.33 What is guard in Scala's for-comprehension construct?
- 1.34 How Scala solves Inheritance Diamond Problem automatically and easily than Java 8?
- 1.35 In Scala, Pattern Matching follows which Design Pattern? In Java, 'instanceof' operator follows which Design Pattern?

## Scala Interview Questions

In this section, we are going to list out all Scala Intermediate Interview Questions and in next section we will discuss them in detail.

1. What is Primary Constructor? What is Secondary or Auxiliary Constructor in Scala?

2. What is the use of Auxiliary Constructors in Scala? Please explain the rules to follow in defining Auxiliary Constructors in Scala?
3. What are the differences between Array and ArrayBuffer in Scala?
4. What is case class? What is case object? What are the Advantages of case class?
5. What is the difference between Case Object and Object(Normal Object)?
6. When compare to Normal Class, What are the major advantages or benefits of a Case-class?
7. What is the usage of `isInstanceOf` and `asInstanceOf` methods in Scala? Is there anything similar concept available in Java?
8. How do you prove that by default, Case Object is Serializable and Normal Object is not?
9. Difference between Array and List in Scala?
10. What is the difference between "val" and "lazy val" in Scala? What is Eager Evaluation? What is Lazy Evaluation?
11. What is the Relationship between `equals` method and `==` in Scala? Differentiate Scala's `==` and Java's `==` Operator?
12. Difference between Scala's Inner class and Java's Inner class?
13. What is Diamond Problem? How Scala solves Diamond Problem?
14. Why Scala does NOT have "static" keyword? What is the main reason for this decision?
15. What is the use of "object" keyword in Scala? How to create **Singleton** objects in Scala?
16. How to define Factory methods using object keyword in Scala? What is the use of defining Factory methods in object?
17. What is apply method in Scala? What is unapply method in Scala? What is the difference between apply and unapply methods in Scala?
18. 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?
19. How do we declare a private Primary Constructor in Scala? How do we make a call to a private Primary Constructor in Scala?
20. Does a Companion object access private members of it's Companion class in Scala?
21. 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?
22. What is object in Scala? Is it a singleton object or instance of a class?
23. What is a Companion Object in Scala? What is a Companion Class in Scala? What is the use of Companion Object in Scala?
24. How to implement interfaces in Scala?
25. What is Range in Scala? How to create a Range in Scala?
26. How many values of type `Nothing` have in Scala?
27. How many values of type `Unit` have in Scala?
28. What is Range in Scala? How to create a Range in Scala?
29. In FP, What is the difference between a function and a procedure?
30. What are the major differences between Scala's Auxiliary constructors and Java's constructors?
31. What is the use of 'yield' keyword in Scala's for-comprehension construct?
32. What is guard in Scala's for-comprehension construct?
33. How Scala solves Inheritance Diamond Problem automatically and easily than **Java 8**?
34. In Scala, Pattern Matching follows which **Design Pattern**? In Java, 'isinstanceof' operator follows which Design Pattern?

## Scala Interview Questions and Answers

In this section, we will pickup each and every question from above list and discuss in-detail with suitable examples(if required). If you want to understand these concepts in-depth with examples, please go through my previous posts in Scala Tutorials section.

### 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:-

```
class Person
```

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

```
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:

```
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](#) and [Auxiliary Constructor](#).

## 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.

```
object MyNormalObject
```

- Case Object is created using "case object" keywords. By default, It's also a singleton object

```
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.

```
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.

```
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.

```
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 isInstanceOf method as shown below:

```
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.

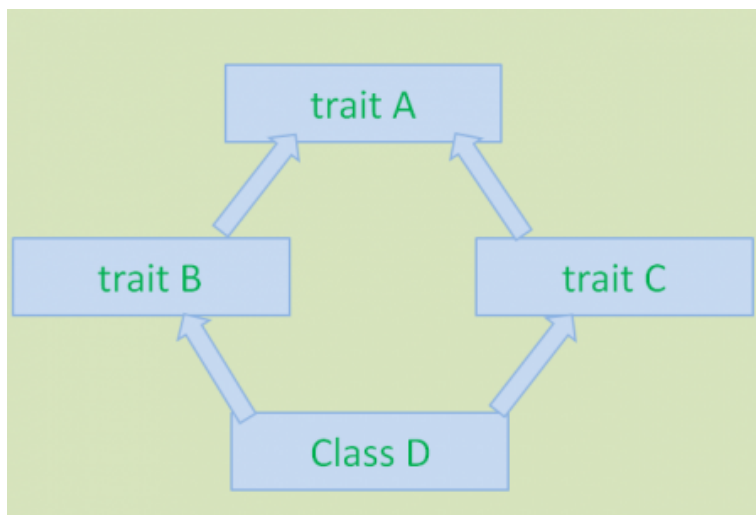
Unlike Java, Scala treats the relationship between Outer class and Inner class differently. Scala's Inner class is associated with Outer class object.

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



A Diamond Problem is a Multiple Inheritance problem. Some people call 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:-

```
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" from trait C. Scala Compiler reads "extends B with C" from right to left and takes "display" method definition from left 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.

```
object MySingletonObject
```

Here, MySingletonObject becomes singleton object automatically.

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

```
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.

```
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.

```
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.

```
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 lastName two components and compose Person object as shown below.

```
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 lastName as shown below.

```
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 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:

```
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".

```
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**

```
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 contract.

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:-

```
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:-

```
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 20, 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.

```
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.

```
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, 'instanceof' operator follows which Design Pattern?

In Scala, Pattern Matching follows Visitor Design Pattern. In the same way, Java's 'instanceof' operator also follows Visitor Design Pattern.

That's it all about "Scala Intermediate Interview Questions and Answers". We will discuss some Advanced Scala Interview Questions and Answers in my coming posts.

Please drop me a comment if you like my post or have any issues/suggestions.

### « PREVIOUS

Java SE 8 Interview Questions and Answers (Part-2)

### NEXT »

Scala Advanced Interview Questions and Answers

### About Rambabu Posa

Rambabu Posa have 13 years of RICH experience as Sr Agile Lead Java/Scala/BigData/NoSQL Developer. Apart from Java, he is good at Spring4, Hibernate4, MEAN Stack, RESTful WebServices, NoSQL, BigData Hadoop Stack, Cloud, Scala, Groovy Grails, Play Framework, Lagom Framework and ConductR, TDD, BDD,Agile,DevOps and much more. His hobbies are Developing software, Learning new technologies, Love Walking, Reading Books, Watching TV and obviously sharing his knowledge through writing articles on JournalDev.

FILED UNDER: [INTERVIEW QUESTIONS](#), [SCALA](#)

## Comments

### Dora says

JULY 4, 2017 AT 11:35 PM

Hi,

i found your articles very helpful not only for interview preparation but to understand the concepts.

for the isInstanceOf question listed above, i got the below result thats different from your answer

```
val str = "hello"
```

```
str: String = hello
```

```
str.isInstanceOf[String]
```

```
res0: Boolean = true
```

[Reply](#)

### sahil says

FEBRUARY 22, 2017 AT 10:17 PM

Sir,

I am new to Scala but found your topics extremely helpful.I would be grateful if you please guide me to learn scala or provide me some source to get acquainted well with scala in easy and lucid manner.

Thanks

Sahil

[Reply](#)

### Rambabu Posa says

FEBRUARY 24, 2017 AT 2:04 PM

Thanks for reading my posts. You can find n number of online Scala resources or Read any good Scala book.

Ram

[Reply](#)

**Hendi Santika says**

NOVEMBER 20, 2016 AT 5:25 PM

Excellent way of explaining all questions answer. One of the best blog ever visited for interview questions.

Thanks A lot.

[Reply](#)

**Sumit Kumar says**

JUNE 23, 2016 AT 6:05 AM

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(val firstName: String, val 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(val firstName: String, val lastName: String)  
      = new Person(firstName, lastName)  
    def apply(val firstName: String, val lastName: String): (String,String)  
      = (firstName, lastName)  
  }  
}
```

please remove " val " from apply method otherwise it won't compile

its a interview question please make sure you code snippet should compile properly.

please reply back if i am wrong.

Thanks

[Reply](#)

**Rambabu Posa says**

JUNE 24, 2016 AT 4:09 AM

Thanks Sumit for catching typo errors. Updated both apply and unapply functions.

[Reply](#)

**min says**

OCTOBER 29, 2016 AT 6:57 PM

the 2nd "apply" method should be "unapply".

[Reply](#)

**min says**

OCTOBER 29, 2016 AT 7:04 PM

and the unapply method parameter type shall be the class.

[Reply](#)

**min says**

OCTOBER 29, 2016 AT 8:08 PM

so it should be something like this:

```
object Person {  
  def apply...  
  def unapply(x: Person):Option[(String, String)] = {  
    Some((x.firstName, x.lastName))  
  }  
}
```

**Rambabu Posa says**

OCTOBER 30, 2016 AT 12:47 AM

Thanks for finding the typo error. Updated, please check it and come back to me if you find any errors.

Ram

**Ramdas Bhingole says**

APRIL 28, 2016 AT 12:38 AM

Wao!

Excellent way of explaining all questions answer. One of the best blog ever visited for interview questions.

Thanks A lot.

[Reply](#)

**Bish says**

APRIL 6, 2016 AT 4:11 PM

Very nice....

[Reply](#)

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP



IMPORTANT INTERVIEW QUESTIONS

## Java Interview Questions

- > [Core Java Interview Questions](#)
- > [String Interview Questions](#)
- > [Multithreading Interview Questions](#)
- > [Collections Interview Questions](#)
- > [Exception Interview Questions](#)
- > [Java Programming Interview Questions](#)
- > [Java 8 Interview Questions Part 1](#)
- > [Java 8 Interview Questions Part 2](#)
- > [Servlet Interview Questions](#)
- > [JSP Interview Questions](#)
- > [Struts 2 Interview Questions](#)
- > [JDBC Interview Questions](#)
- > [Spring Interview Questions](#)
- > [Hibernate Interview Questions](#)
- > [JSF Interview Questions](#)
- > [Web Services Interview Questions](#)
- > [Scala Basic Interview Questions](#)

- > [Scala Intermediate Interview Questions](#)
- > [Scala Advanced Interview Questions](#)
- > [Scala Interview Questions Summary](#)
- > [Common Job Interview Questions](#)

Miscellaneous

- > [Java ClassLoader](#)
- > [String StringBuffer StringBuilder](#)
- > [Java is Pass By Value](#)
- > [Java Heap vs Stack Memory](#)

RECOMMENDED TUTORIALS

Java Tutorials

- > [Java IO](#)
- > [Java Regular Expressions](#)
- > [Multithreading in Java](#)
- > [Java Logging](#)
- > [Java Annotations](#)
- > [Java XML](#)
- > [Collections in Java](#)
- > [Java Generics](#)
- > [Exception Handling in Java](#)
- > [Java Reflection](#)
- > [Java Design Patterns](#)
- > [JDBC Tutorial](#)

Java EE Tutorials

- > [Servlet JSP Tutorial](#)
- > [Struts2 Tutorial](#)
- > [Spring Tutorial](#)
- > [Hibernate Tutorial](#)
- > [Primefaces Tutorial](#)
- > [Apache Axis 2](#)
- > [JAX-RS](#)
- > [Memcached Tutorial](#)

**Java Books for Beginners**

---

**Download Java Plugins**

---

**Free Interactive Java Tutorial**

---

**Beginners Java Script Tutorials**

---

**Java Game Development  
Tutorials**

---

**Download Java Development  
Kit**

---

**Basic Java Tutorials**

---

**Java Tutorials for Beginners**

---

© 2018 · Privacy Policy · Don't copy, it's Bad Karma · P

