Scala Language-Functional and object-oriented programming

C.Phaneendra Reddy-201301006

INTRODUCTION

•Scala language is a general purpose programming language. Scala has full support for functional programming and a very strong static type system.Designed to be concise,many of the scala's design decisions were inspired by the criticism of the shortcomings of java.

PROBLEM

Criticisms that are solved by scala are implementation of generics, forced object oriented programming, Implementation of floating point arithematic.

Lexical Syntax

- the two modes of Scala's lexical syntax, the Scala mode and the *XML mode*.
- In Scala mode, *Unicode escapes* are replaced by the corresponding Unicode character with the given hexadecimal code
- Names in Scala identify types, values, methods, and classes which are collectively called *entities*. Names are introduced by local definitions and declerations, inheritance, import clauses, or package clauses which are collectively called *bindings*..

Identifiers, names and scopes

- There are two different name spaces, one for types and one for terms. The same name may designate a type and a term, depending on the context where the name is used.
- A binding has a *scope* in which the entity defined by a single name can be accessed using a simple name. Scopes are nested. A binding in some inner scope *shadows* bindings of lower precedence in the same scope as well as bindings of the same or lower precedence in outer scopes.

Classes and Objects:

•Classes and objects are both defined in terms of *templates*.

•A template defines the type signature, behavior and initial state of a trait or class of objects or of a single object. Templates form part of instance creation expressions, class definitions, and object definitions

Delayed Initialization:

- The initialization code of an object or class (but not a trait) that follows the superclass constructor invocation and the mixin-evaluation of the template's base classes is passed to a special hook, which is inaccessible from user code. Normally, that hook simply executes the code that is passed to it. But templates inheriting the scala.DelayedInit trait can override the hook by re-implementing the delayedInit method, which is defined as follows:

def delayedInit(body: => Unit)

m^e

Annotations:

- Annotations associate meta-information with definitions. A simple annotation has the form @*c* or @*c*(*a*1,…,*an*). Here, *c* is a constructor of a class *C*, which must conform to the class scala.Annotation.
- Annotations may apply to definitions or declarations, types, or expressions. An annotation of a definition or declaration appears in front of that definition.
- More than one annotation clause may apply to an entity. The order in which these annotations are given does not matter.

Basic Example of a scala language:

```
object pyramid_inv
{
    def main(args: Array[String])
    {
        var i=0
        var j=0
        var k=0
        var n=readInt()
        for( i <- 1 to n)
        {
            //for spaces
            for( k <- 1 to (i-1))
            {
                print(" ")
            }
            //for stars
            for( j <- i to n)
            {
                print("* ")
            }
            println()
        }
    }
}
```

Expressions:

- Expressions are composed of operators and operands. Expression forms are discussed subsequently in decreasing order of precedence.

Packagings:

A package is a special object which defines a set of member classes, objects and packages. Unlike other objects, packages are not introduced by a definition. Instead, the set of members of a package is determined by packagings**.**

Scala is a functional language in the sense that every function is a value. Nesting of function definitions and higher-order functions are naturally supported. Scala also supports a general notion of pattern matching which can model the algebraic types used in many functional languages. Scala has been designed to interoperate seamlessly with Java. Scala classes can call Java methods, create Java objects, inherit from Java classes and implement Java interfaces. None of this requires interface definitions or glue code.