EXP. No.: 6

Execute pattern matching, exception handling, method creation, functional programming(closures, currying, expressions, anonymous functions).

Aim:

To write a program to execute pattern matching, exception handling, method creation and functional programming.

Pattern Matching:

Pattern Matching is a powerful feature in Scala that allows checking a value against a pattern. It's much more versatile than a traditional switch statement.

Syntax :

```
value match{

        case pattern 1 => // code for pattern 1
        case pattern 2 => // code for pattern 2
        case _ => // default case
}
```

Example :

```
val x = 10
x match {
    case 1 => println("One")
    case 2 => println("Two")
     case 10 => println("Ten")
    case _ => println("Something else")
}
```

Output :

```
Ten
 val x: Int = 10
```

Exception Handling:

Scala uses try, catch and finally blocks to handle exceptions, similar to Java. However, unlike Java, in Scala are expression which means they can return values.

Syntax:

```
try {

   // code that might throw an exception

} catch {
```

```
    // case ex: ExceptionType => // handle exception
  } finally {
    //optional block to clean up resources
  }
```

Example:

```
def divide(x: Int, y: Int): Int = {
    try {
        x / y
    } catch {
        case e: ArithmeticException =>
         println("Cannot divide by zero!")
        0 // Return a default value or handle accordingly
     } finally {
        println("Execution completed.")
    }
 }
val result = divide(10, 0)
println(s"Result: $result")
```

Output :

```
Cannot divide by zero!
Execution completed.
Result: 0
def divide(x: Int, y: Int): Int
val result: Int = 0
```

Method Creation :

Methods in Scala are similar to functions in other programming languages.You can define methods inside objects, classes or traits.

Syntax :

```
def methodName (parameter1: Type , parameter2: Type) : ReturnType= {
    //method body
    returnvalue
}
```

Example :

```
val sum = add(5, 3)
println(s"Sum: $sum") // Output: Sum: 8
```

Output :

Sum: 8
val sum: Int = 8

Functional Programming:

Scala is a functional programming language that supports concepts like closure, currying, expression and anonymous functions.

1.Closures:

A closure is a function that captures the bindings of its free variables. In Scala, closures are functions whose return value depends on variables declared outside the function.

Example:

```
var number = 10
val addnumber = (x: Int) => x + number
println(addnumber(5))
number = 20
println(addnumber(5))
```

Output:

15
25

2.Currying:

Currying is the process of transforming a function that takes multiple arguments into a Sequence of function each with one argument.

Syntax:

```
def add(a: Int)(b: Int): Int = a + b
```

Example:

```
def multiply(x: Int)(y: Int): Int = x * y
val multiplyBy2 = multiply(2) _
println(multiplyBy2(5))
```

Output:

10

3.Expression:

In Scala, almost everything is an expression meaning it returns a value.
For example, if-else: match and even methods returns values.

Example:

```
val result = if (5 > 3) "Greater" else "Lesser"
println(result)
```

Output:

Greater

4.Anonymous Function:

They are functions without a name. In Scala, you can define anonymous functions using the => symbol.

Syntax:

```
(parameter1:type,parameter2:type)=> expression
```

Example:

```
val add = (x: Int, y: Int) => x + y
println(add(3, 4))
```

Output:

7

Result:

Hence, the implementation of pattern matching, exception handling, method creation, functional programming was executed successfully.