Lembar Kerja Praktikum KOM120C Pemrograman
13: Functional Programming V (OOP di Scala)

PETUNJUK PRAKTIKUM

Review

- Konsep Functional Programming, menggunakan bahasa Scala
- Function as first class citizen, higher order function (HOF)
- HOF untuk pengurutan dalam list: sorted, sortBy, dan sortWith.
- HOF untuk tipe data string: split, length, reverse, dll.
- HOF groupBy untuk mengelompokkan list berdasarkan ketentuan tertentu, menghasilkan kembalian berupa map.
- Struktur data pada Scala (Array, List, Set, Map, Tuple)
- Compiler Scala Online: https://onecompiler.com/scala

Kenapa OOP menggunakan Scala?

- Scala dibangun di atas Java Virtual Machine (JVM), di mana Java sendiri adalah bahasa object-oriented. Menggunakan OOP di Scala memudahkan integrasi dengan library-library di Java.
- Penggunaan kedua paradigma (FP dan OOP) dalam satu program memungkinkan pengembangan kode yang lebih modular, selama menerapkan *best practice* seperti separation of concern.
- Konsep OOP yang akan dibahas di pertemuan ini: Class & Constructor, Inheritance, Polymorphism, dan Singleton Class

Class dan Constructor

- Class dapat dideklarasikan sebagai berikut. Jika pada bahasa lain seperti Java constructornya ditulis terpisah seperti method, pada Scala, constructornya secara default menyatu dengan deklarasi dari class.

```
class Person(val name: String, val age: Int) {
  def greet(): String = {
    s"Hello, my name is $name and I am $age years old."
  }
}

// (val name: String, val age: Int) termasuk contructor

val adi = new Person("Adi", 30)
println(adi.greet())

// Output: Hello, my name is Adi and I am 30 years old.
```

- Jika kita ingin menetapkan default value dari masing-masing atribut, kita bisa melakukannya seperti berikut.

```
class Person(val name: String = "Fulan", val age: Int = 0) {
  def greet(): String = {
    s"Hello, my name is $name and I am $age years old."
  }
}
```

- Lebih lanjut tentang constructor: constructor didefinisikan sebagai method spesial yang tujuannya menginisialisasi class dan tidak mempunyai return value. Pernyataan pada constructor dijalankan setiap kali class diinstansiasi menjadi objek.
- Ada dua jenis constructor di Scala: primary (default) dan auxiliary (tambahan). Constructor primary letaknya adalah pada class itu sendiri. Misalnya:

```
class Person(val name: String = "Fulan", val age: Int = 0) {
  println(s"Constructor terpanggil. Data objek: nama $name usia $age")
  def greet(): String = {
    s"Hello, my name is $name and I am $age years old."
  }
}

object Main {
       def main(args: Array[String]): Unit = {
    val adi = new Person("Adi", 12)
       }
}

// output: Constructor terpanggil. Data objek: nama Adi usia 12
```

- Jika diperlukan, kita juga bisa menambahkan auxiliary constructor. Constructor jenis ini didefinisikan seperti sebuah method dengan keyword “this”, mirip dengan constructor pada bahasa pemrograman lain. Sebuah konstruktor auxiliary harus memanggil constructor lain (default/auxiliary). Pada contoh di bawah, auxiliary constructor menetapkan default value=0 untuk age.

```
class Person(val name: String, val age: Int) {
  // primary
  println(s"Constructor terpanggil. Data objek: nama $name usia $age")

  // auxiliary
  def this(name: String) = this(name, 0)

  def greet(): String = {
    s"Hello, my name is $name and I am $age years old."
  }
}

object Main {
       def main(args: Array[String]): Unit = {
    val adi = new Person("Adi")
       }
}

// output: Constructor terpanggil. Data objek: nama Adi usia 0
```

- Sebagai catatan, access modifier di Scala by default adalah public. Bisa diubah dengan menambahkan keyword “private” atau “protected” di depan definisi atribut/method.

Inheritance

- Seperti pada Java, inheritance pada Scala dapat dilakukan menggunakan keyword extends. Method overriding dapat dilakukan dengan keyword override.

```
// Superclass
class Animal(val name: String) {
  def makeSound(): String = "suara binatang"
}

// Subclass
class Dog(name: String) extends Animal(name) {
  override def makeSound(): String = "guk guk"
}

object HelloWorld {
  def main(args: Array[String]): Unit = {
    val dog = new Dog("hachiko")
    val dogname = dog.name
    val dogsound = dog.makeSound()
    println(s"$dogname bilang $dogsound" )
  }
}

// output: hachiko bilang guk guk
```

- Di Scala juga terdapat abstract class, yaitu class yang tidak bisa diinstansiasi. Untuk menerapkan class ini, harus dilakukan inheritance dan overriding terhadap methodnya.

```
// Superclass
abstract class Animal(val name: String) {
  def makeSound(): String // abstract method
}

// Subclass
class Dog(name: String) extends Animal(name) {
  override def makeSound(): String = "gukguk"
}

object Main {
  def main(args: Array[String]): Unit = {
    val animal = new Animal("otis") // error
    val dog1 = new Dog("hachiko")
    println(dog1.makeSound()) // output: gukguk
  }
}
```

Polymorphism

- Scala mendukung konsep polymorphism sebagaimana bahasa lainnya yang mendukung OOP. Pada saat compile (compile-time), beberapa object dari subclass

yang berbeda akan dianggap sama (sebagai superclassnya). Namun pada saat runtime, object-object tersebut akan dianggap berasal dari class aslinya (sebagai subclass).

- Berikut adalah contoh polymorphism di mana beberapa object dari subclass berbeda dimasukkan ke dalam satu List yang menerima object "Animal".

```
// Superclass
class Animal(val name: String) {
  def makeSound(): String = "suara binatang"
}

// Subclasses
class Cow(name: String) extends Animal(name) {
  override def makeSound(): String = "moo"
}

class Dog(name: String) extends Animal(name) {
  override def makeSound(): String = "gukguk"
}

object Main {
  def main(args: Array[String]): Unit = {
    // Creating instances of different animals
    val cow1 = new Cow("otis")
    val dog1 = new Dog("hachiko")
    val cow2 = new Cow("jajang")

    // Storing instances in a list of Animals
    val animals: List[Animal] = List(cow1, dog1, cow2)

    // Accessing elements of the list and invoking methods polymorphically
    animals.foreach(animal => println(s"${animal.name} bilang
${animal.makeSound()}"))
  }
}

/* output:
otis bilang moo
hachiko bilang gukguk
jajang bilang moo
*/
```

Singleton Class

- Pada Scala, ada jenis object khusus yang disebut sebagai singleton object.
- Singleton object mirip dengan atribut/method yang bersifat static, di mana hanya ada satu instance dari object ini dalam seluruh program.
- Biasanya, singleton object digunakan untuk menyimpan dan mengumpulkan konstanta serta metode-metode yang digunakan untuk tujuan tertentu, seperti pada contoh berikut yang menghitung luas lingkaran.

```
object MathUtil {
  val PI = 3.141592
  def kuadrat(x: Double) : Double = x*x
}

val r = 20
val luas = MathUtil.PI * MathUtil.kuadrat(r)
println(luas)
```

```
// output: 1256.6368
```

- Singleton object juga kita temukan setiap kali kita ingin menjalankan kode Scala, yakni tempat di mana fungsi main berada. Di sini, nama objectnya adalah HelloWorld.

```
object HelloWorld {
    def main(args: Array[String]): Unit = {
        println("Hello, World!")
    }
}
```

CONTOH SOAL

Latihan: Buatlah sebuah program yang membaca N buah baris, dimana setiap baris adalah data sebuah bangun 2D: lingkaran atau pun persegi, yang ditunjukkan dengan huruf penanda: 'L' (lingkaran) atau 'P' (persegi) diikuti dengan panjang jari-jari (untuk lingkaran) atau panjang dan lebar (untuk persegi). Program kemudian menghitung luas dan keliling dari masing-masing bangun

Contoh Input:
3
L 5
P 3 5
P 2 4

Gunakan fitur-fitur OOP pada Scala untuk mengimplementasikan program tersebut (class, object, inheritance, singleton object, polymorphisme)

Kode

```
// konstanta & fungsi dasar
object MathUtil {
  val PI = 3.141592
  def kuadrat(x: Double) : Double = x*x
}

// superclass
abstract class Shape() {
  def luas(): Double
  def keliling(): Double
}

// subclasses
class Lingkaran(val radius: Double) extends Shape {
  override def luas(): Double = MathUtil.PI * MathUtil.kuadrat(radius)
  override def keliling(): Double = 2 * MathUtil.PI * radius
}

class Persegi(val panjang: Double, val lebar: Double) extends Shape {
  override def luas(): Double = panjang * lebar
  override def keliling(): Double = 2*panjang + 2*lebar
}
```

```
// main code
object Main {
  def main(args: Array[String]): Unit = {
    val n = scala.io.StdIn.readInt()

    // baca data
    for (_ <- 1 to n) {
      val line = scala.io.StdIn.readLine().split(" ")
      val tipe = line(0)

      // jika lingkaran
      val result = if (tipe == "L") {
        val radius = line(1).toDouble
        val lingkaran1 = new Lingkaran(radius)
        s"${lingkaran1.luas()} ${lingkaran1.keliling()}"
      } else if (tipe == "P") {
        val panjang = line(1).toDouble
        val lebar = line(2).toDouble
        val persegi1 = new Persegi(panjang, lebar)
        s"${persegi1.luas()} ${persegi1.keliling()}"
      } else {
        "input tdk valid"
      }

      println(result)
    }
  }
}
```