МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторная работа №3

по дисциплине

Функциональное программирование

на тему

«Pattern matching и case-классы»

Выполнил: студент группы

БВТ2201

Шамсутдинов Рустам

Москва 2025

Введение

Цель работы:

Изучить механизм сопоставления с образцом (pattern matching). Научиться использовать case-классы и алгебраические типы данных (ADT). Реализовать простые структуры данных и их обработку.

Теоретические основы:

– Синтаксис pattern matching в Scala.

– Case-классы, их особенности и преимущества.

– Алгебраические типы данных (ADT).

– Полиморфизм и типизация.

Задания:

1. Создать case-класс с несколькими полями и использовать его в pattern matching.

2. Реализовать функцию, которая принимает значение произвольного типа и обрабатывает его с помощью pattern matching.

3. Создать ADT (например, тип для арифметических выражений) и написать функцию вычисления значения выражения.

4. Продемонстрировать работу полиморфной функции с использованием дженериков.

Ход работы

Задание 1. Создать case-класс с несколькими полями и использовать его в pattern matching.

```
run | debug
object PatternMatching {
  def main(args: Array[String]): Unit = {
    println("\ntask 1:\n")

    case class Person(name: String, age: Int, city: Option[String])

    val people = List(
      Person("Joe", 21, Some("Los Angeles")),
      Person("Alice", 25, None),
      Person("Bryan", 16, Some("Austin")),
      Person("David", 15, None)
    )

    def describe(p: Person): String = p match {
      case Person(name, age, Some(city)) if age >= 18 => s"$name, $age y.o. lives in $city, adult"
      case Person(name, age, Some(city)) if age < 18 => s"$name, $age y.o. lives in $city, kid"
      case Person(name, age, None) if age >= 18 => s"$name, $age y.o. homeless adult"
      case Person(name, age, None) if age < 18 => s"$name, $age y.o. homeless kid"
      case _ => "no data"
    }
    people.map(describe).foreach(println)
  }
}
```

```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

sypoo@pop-os:~/Mtuci/mtuci-scala/lab3/patternmatching$ sbt
[info] welcome to sbt 1.11.7 (Eclipse Adoptium Java 11.0.28)
[info] loading project definition from /home/sypoo/Mtuci/mtuci-scala/lab3/patternmatching/project
[info] loading settings for project root from build.sbt...
[info] set current project to PatternMatching (in build file:/home/sypoo/Mtuci/mtuci-scala/lab3/patternmatching/)
[info] sbt server started at local:///home/sypoo/.sbt/1.0/server/42c1944c6d71b255b29f/sock
[info] started sbt server
sbt:PatternMatching> ~run
[info] running PatternMatching

task 1:

Joe, 21 y.o. lives in Los Angeles, adult
Alice, 25 y.o. homeless adult
Bryan, 16 y.o. lives in Austin, kid
David, 15 y.o. homeless kid
[success] Total time: 1 s, completed Nov 13, 2025, 4:29:45 AM
[info] 1. Monitoring source files for root/run...
[info]    Press <enter> to interrupt or '?' for more options.
```

Рис. 1 – работа с case-классами и pattern matching.

Задание 2. Реализовать функцию, которая принимает значение произвольного типа и обрабатывает его с помощью pattern matching.

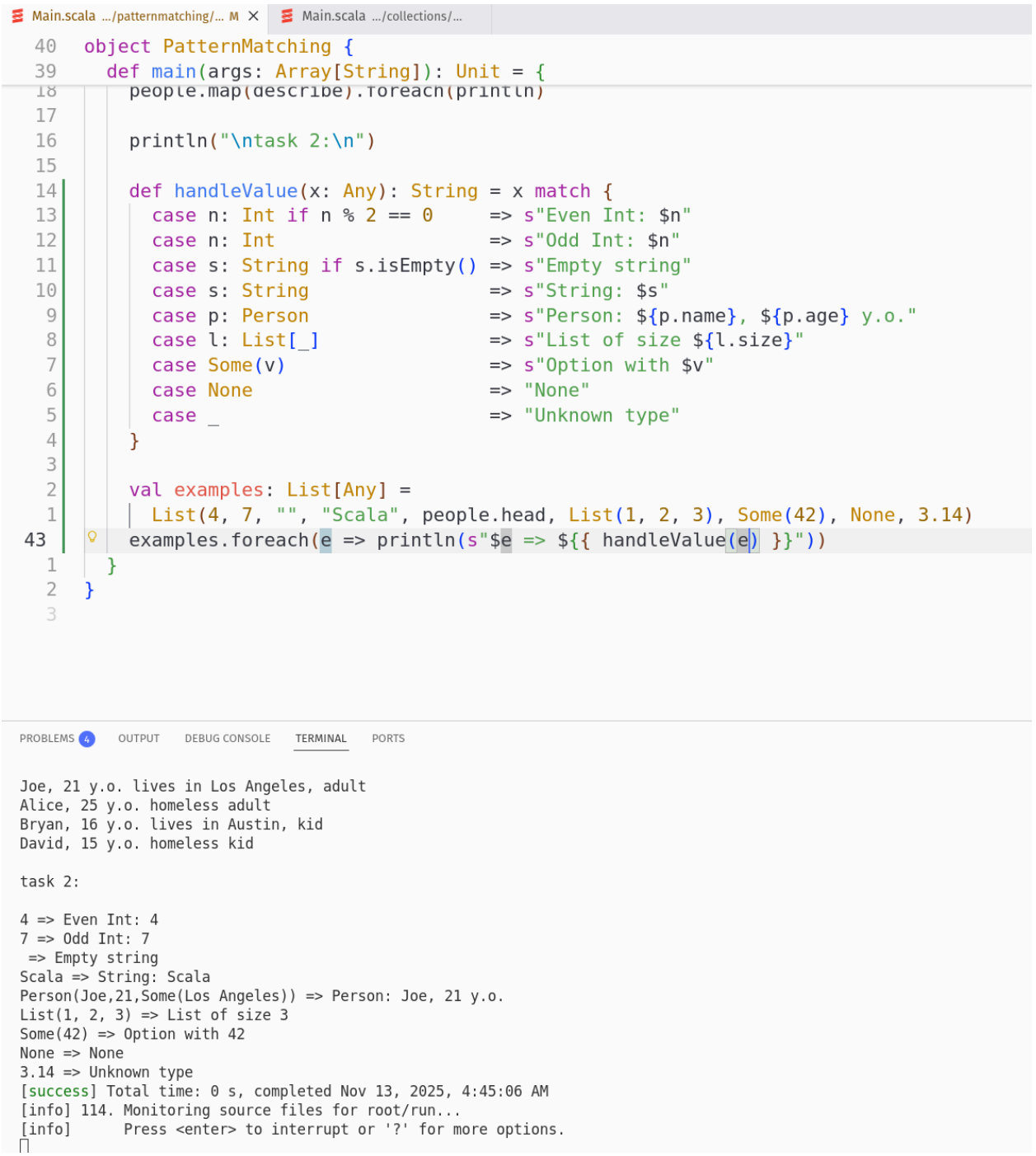

Рис. 2 – работа с pattern matching для произвольных типов.

Задание 3. Создать ADT (например, тип для арифметических выражений) и написать функцию вычисления значения выражения.

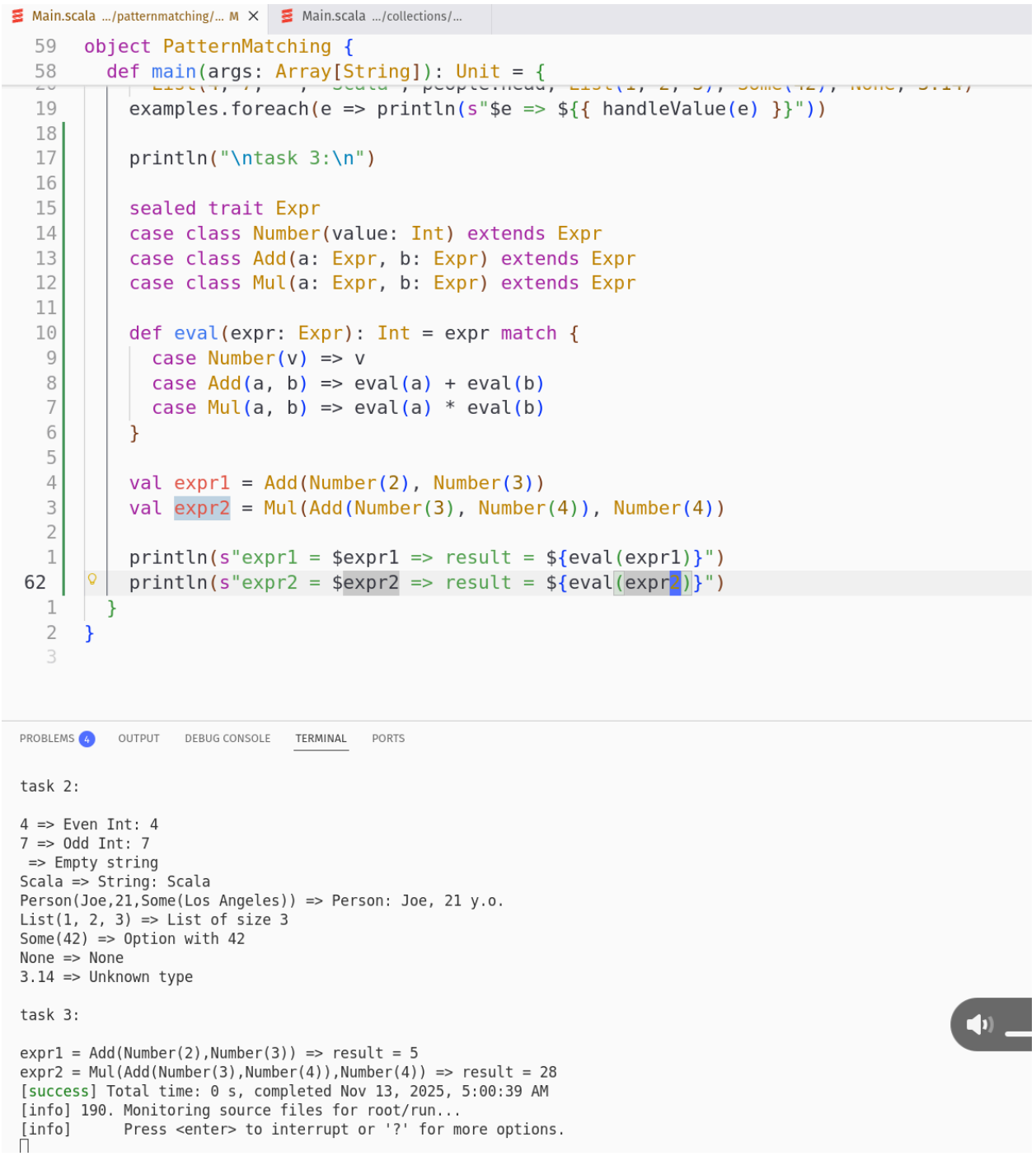

Рис. 3 – работа с ADT для арифметических выражений.

Задание 4. Продемонстрировать работу полиморфной функции с использованием дженериков.

Рис. 4 – работа с полиморфными функциями с дженериками.

Заключение

Вывод: проделав работу, мы изучили механизм сопоставления с образцом (pattern matching), научились использовать case-классы и алгебраические типы данных (ADT), а также реализовали простые структуры данных и их обработку.

Контрольные вопросы

1. Назовите ключевые принципы функционального программирования. Чем они отличаются от императивного подхода?

Ключевые принципы ФП:

– Чистые функции (без побочных эффектов)

– Иммутабельность данных

– Функции высшего порядка

– Рекурсия вместо циклов

– Выражения вместо инструкций

Отличия от императивного подхода:

– ФП: что вычислять (декларативно)

– Императивное: как вычислять (последовательность команд)

– ФП: данные иммутабельны, создаются новые

– Императивное: данные мутабельны, изменяются состояния

– ФП: акцент на трансформации данных

– Императивное: акцент на изменении состояния

2. Что такое «чистые функции»? Приведите пример в Scala.

Чистая функция - функция, которая:

– Всегда возвращает одинаковый результат для одинаковых входных данных

– Не имеет побочных эффектов (не изменяет внешнее состояние)

```
// Чистая функция

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

```
// НЕ чистая функция
var counter = 0
def increment(): Int = {
  counter += 1  // побочный эффект
  counter
}
```

3. Объясните термин «иммутабельность». Почему она важна в ФП?

Иммутабельность - свойство объектов, которые нельзя изменить после создания.

Важность в ФП:

– Потокобезопасность (нет гонки данных)

– Предсказуемость поведения

– Упрощение отладки

– Возможность кеширования результатов

– Облегчение композиции функций

4. В чем разница между val и var? Когда что использовать?

val - иммутабельная ссылка (нельзя переназначить)

var - мутабельная ссылка (можно переназначить)

```
val name = "Alice"  // Нельзя изменить
var age = 25        // Можно изменить: age = 26
```

Когда что использовать:

– Используйте val по умолчанию

– Используйте var только когда действительно нужно изменять значение

5. Что такое case-классы? Какие преимущества они дают?

Case-классы - специальный тип классов с автоматической реализацией полезных методов.

Преимущества:

– Автоматические equals/hashCode

– Автоматический toString

– Копирование с изменением (copy)

– Возможность использования в pattern matching

– Компаньон-объект с apply

```
case class Person(name: String, age: Int)

val person = Person("Alice", 25)  // Не нужен 'new'
```

6. Как работает pattern matching в Scala? Приведите пример.

Pattern matching - мощный механизм сопоставления с образцом, похожий на switch, но более выразительный.

```
def describe(x: Any): String = x match {
  case 1 => "один"
  case "hello" => "приветствие"
  case List(1, 2, 3) => "список 1,2,3"
  case Person(name, age) => s"Человек: $name, $age лет"
  case _ => "что-то другое"
}
```

7. Что такое функция высшего порядка? Приведите пример из стандартной библиотеки Scala.

Функция высшего порядка - функция, которая принимает другие функции как параметры или возвращает функцию как результат.

```
// exists - проверяет, существует ли элемент, удовлетворяющий условию
val hasEven = List(1, 3, 5, 7).exists(_ % 2 == 0)  // false
val hasNegative = List(1, -2, 3).exists(_ < 0)     // true
// takeWhile - берет элементы из коллекции, пока условие истинно
val numbers = List(2, 4, 6, 7, 8, 10)
val evenPrefix = numbers.takeWhile(_ % 2 == 0)  // List(2, 4, 6)
// sortWith - сортирует коллекцию с использованием функции сравнения
val names = List("Alice", "Bob", "Charlie")
val sortedByLength = names.sortWith(_.length < _.length)  // List("Bob", "Alice", "Charlie")
// collect - комбинирует map и filter, применяя частичную функцию
val mixed = List(1, "hello", 2, "world", 3)
val onlyInts = mixed.collect { case x: Int => x * 2 }  // List(2, 4, 6)
```

8. Объясните, как работают map, filter и reduce для коллекций.

map - преобразует каждый элемент коллекции

filter - оставляет только элементы, удовлетворяющие условию

reduce - агрегирует все элементы в один результат

```
List(1, 2, 3).map(_ * 2)  // List(2, 4, 6)
List(1, 2, 3, 4).filter(_ > 2)  // List(3, 4)
List(1, 2, 3, 4).reduce(_ + _)  // 10
```

9. Что такое каррирование? Как оно реализуется в Scala?

Каррирование - преобразование функции от многих аргументов в последовательность функций от одного аргумента.

```
// Обычная функция
def add(x: Int, y: Int): Int = x + y
// Каррированная версия
def curriedAdd(x: Int)(y: Int): Int = x + y
// Частичное применение
val addFive = curriedAdd(5)
val result = addFive(3)  // 8
```

10. Что такое алгебраические типы данных (ADT)? Приведите пример.

ADT - композитные типы, создаваемые комбинацией других типов через:

– Произведение (product types) - например, case-классы

– Сумму (sum types) - например, sealed traits

```
// ADT пример: представление арифметических выражений
sealed trait Expr
case class Number(n: Int) extends Expr
case class Add(left: Expr, right: Expr) extends Expr
case class Multiply(left: Expr, right: Expr) extends Expr
// Использование
val expression = Add(Number(2), Multiply(Number(3), Number(4)))
```