Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

*Кафедра «Математическая кибернетика и информационные технологии»*

Отчет по лабораторной работе №1

Выполнила: студентка группы БВТ1903

Лаптева Анастасия

Руководитель:

Мосева Марина Сергеевна

1. Написать функции, выполняющие расчеты по некоторым математическим формулам.

object Functions {  
  
 /\* a) Напишите функцию, которая рассчитывает площадь окружности  
 \* r^2 \* Math.PI  
 \*/  
 def A(r: Double): Double = r \* r \* Math.*PI* // примените вашу функцию из пункта (a) здесь, не изменяя сигнатуру  
 def testCircle(r: Double): Double = *A*(r)  
  
  
 /\* b) Напишите карированную функцию которая рассчитывает площадь прямоугольника a \* b.  
 \*/  
 def B(a: Double)(b: Double): Double = a \* b  
  
  
 // примените вашу функцию из пукта (b) здесь, не изменяя сигнатуру  
 def testRectangleCurried(a: Double, b: Double): Double = *B*(a)(b)  
  
  
 // c) Напишите не карированную функцию для расчета площади прямоугольника.  
  
 def C(a: Double, b: Double): Double = a \* b  
  
 // примените вашу функцию из пункта (c) здесь, не изменяя сигнатуру  
 def testRectangleUc(a: Double, b: Double): Double = *C*(a, b)

Результат работы программы:

Изображение выглядит как текст, снимок экрана, монитор, компьютер

Автоматически созданное описание

1. Написать функции с использованием конструкции case.

def A (n:Int): String = n match {  
 case 1 => "it is one"  
 case 2 => "it is two"  
 case 3 => "it is three"  
 case \_ => "what's that"  
 }  
  
 // примените вашу функцию из пункта (a) здесь, не изменяя сигнатуру  
 def testIntToString(value: Int): String = *A*(value)  
  
 /\* b) Напишите функцию которая возвращает true если переменная `value` принимает значение:  
 \* "max" или "Max  
 \* "moritz" или "Moritz"  
 \*/  
def B (s: String):Boolean = s match {  
 case "max"|"Max"|"moritz"|"Moritz"=>true  
 case \_=> false  
}  
  
  
 // примените функции из пункта (b) здесь, не изменяя сигнатуру  
 def testIsMaxAndMoritz(value: String): Boolean = *B*(value)  
  
 // c) Напишите функцию проверки является ли `value` четным   
def C (n:Int):Boolean = n match {  
 case n => n%2==0  
 case \_=> false  
}  
  
  
 // примените функции из пункта (c) здесь, не изменяя сигнатуру  
 def testIsEven(value: Int): Boolean = *C*(value)  
  
  
   
 /\* d) Напишите функцию, моделирующую игру в Камень ножницы бумага   
 \* 1. камень побеждает ножницы  
 \* 2. ножницы побеждают бумагу  
 \* 3. бумага побеждает камень  
 \* Выиграет ли игрок `a`?  
 \*///Scissor, Paper, Rock  
def D (a: Hand, b: Hand): Result = (a,b) match {  
 case (Rock,Scissor)|(Scissor,Paper)|(Paper,Rock)=>Win  
 case (Rock,Rock)|(Scissor,Scissor)|(Paper,Paper)=>Draw  
 case \_=>Lose  
}  
  
 // примените вашу функцию из пункта (d) здесь, не изменяя сигнатуру  
 def testWinsA(a: Hand, b: Hand): Result = *D*(a,b)  
  
  
  
 // Примечание: используйте определение Animals  
  
 // e) Верните вес (weight: Int) объекта Mammal, иначе верните -1.  
def e (animal: Animal):Int = animal match {  
 case animal: Mammal => animal.weight  
 case \_=> -1  
}  
  
  
 // примените функцию из пункта (e) здесь, не изменяйте сигнатуру  
 def testExtractMammalWeight(animal: Animal): Int = *e*(animal)  
  
  
 // f) Измените поле еда объектов классов Fish и Birds на Plants, класс Mammal оставьте неизмененным.  
def f ( animal: Animal):Animal = animal match {  
 case animal: Mammal=>animal  
 case \_=> animal.food = Plants  
 animal  
}  
  
  
 // примените функцию из пункта (f) здесь, не изменяйте сигнатуру  
 def testUpdateFood(animal: Animal): Animal = *f*(animal)

Результат работы программы:

Изображение выглядит как текст, снимок экрана, монитор, компьютер

Автоматически созданное описание

1. Написать высокоуровневые функции.

object HigherOrder {  
  
 val *plus*: (Int, Int) => Int = \_ + \_  
 val *multiply*: (Int, Int) => Int = \_ \* \_  
  
 /\* a) Напишите функцию, которая принимает `f: (Int, Int) => Int`, параменты `a` и `b`  
 \* и коэффициент умножения `n` и возвращает n \* f(a, b). Назовите `nTimes`.  
 \*/  
def nTimes (f: (Int, Int) => Int,a:Int, b:Int, n:Int):Int=n\*f(a,b)  
  
  
 // примените вашу функцию (a) здесь, не изменяйте сигнатуру  
 def testNTimes(f: (Int, Int) => Int, a: Int, b: Int, n: Int): Int = *nTimes*(f,a,b,n)  
  
 /\* b) Напишите анонимную функцию, функцию без идентификатора ((a, b) => ???) для `nTimes` которая  
 \* выполняет следующее:  
 \* if (a > b) a else b  
 \*/  
 def testAnonymousNTimes(a: Int, b: Int, n: Int): Int = *nTimes*((a:Int,b:Int)=>if (a > b) a else b,a, b, n)

Результат работы программы:

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

1. Написать класс, переопределить его, как Трейт. Унаследовать от этого трейта классы – образцы. Создать класс-компаньон для трейта и определить там несколько методов. После чего создать трейт Еда.

*/\*\*  
 a) Создать класс Animal, который имеет следующие поля:  
 \** - *name: String (название)  
 \** - *species: String (вид)  
 \** - *food: String  
 \*  
 \* Синтаксис: class MyClass(val publicField: Int, privateField: String) {  
 \* // остальные поля и методы  
 \* }  
  
 class Animal (val name: String,val species: String,val food: String) {  
 /\* c) Добавьте следующие метод в Animals:  
 \* def eats(food: String): Boolean  
 \*  
 \* который проверяет ест ли животное определенную пищу  
 \*/  
 def Eats(food:String):Boolean = food==this.food  
 }  
 \*\*/*/\* d) Переопределите ваш класс Animal как трейт и создайте объекты класса-образца для Mammals, Birds и Fishs.  
\* Вам все еще нужно поле `species`?  
\*/  
trait Animal{  
val name : String  
val food : Food  
def eats(food: Food): Boolean = food == this.food  
}  
case class Mammal (name: String, food: Food, weight: Int)extends Animal  
case class Bird (name: String, food: Food)extends Animal  
case class Fish (name: String, food: Food)extends Animal  
/\* b) Создайте объект-компаньон для класса Animal и добавьте следующие сущности как поля:  
\* - cat, mammal, meat  
\* - parrot, bird, vegetables  
\* - goldfish, fish, plants  
\*  
\* Синтаксис: object MyClass {  
\* // статические поля и методы  
\* }  
\*/  
object Animal {  
 */\*\*  
 val c = Animal("cat", "mammal", "meat")  
 val p = Animal("parrot", "bird", "vegetables")  
 val d = Animal("goldfish", "fish", "plants")\*\*/* /\* e) Добавьте следующие функции в объект-компаньон Animal:  
 \* def knownAnimal(name: String): Boolean // true если это имя одного из трех животных из (b)  
 \* def apply(name: String): Option[Animal] // возвращает одно из трех животных в соответствии с именем (Some) или ничего (None), см. ниже  
 \*/  
 def knownAnimal(name: String): Boolean = name=="cat"|name=="parrot"|name=="goldfish"  
 def apply(name: String): Option[Animal] = name match {  
 case "cat"=>*Some*(*Mammal*("cat",Meat,5))  
 case "parrot"=>*Some*(*Bird*("parrot",Vegetables))  
 case "goldfish"=>*Some*(*Fish*("goldfish",Plants))  
 case \_ => None  
 }  
}  
  
  
  
/\* f) Создайте трейт Food со следующими классами-образцами:  
\* - Meat  
\* - Vegetables  
\* - Plants  
\* и добавьте это в определение Animal. Так же добавьте объект-компаньон с методом apply():  
\* def apply(food: String): Option[Food]  
\*/  
trait Food  
case object Meat extends Food  
case object Vegetables extends Food  
case object Plants extends Food  
object Food {  
 def apply(food: String): Option[Food] = food match {  
 case "Meat" => *Some*(Meat)  
 case "Vegetables" => *Some*(Vegetables)  
 case "Plants" => *Some*(Plants)  
 case \_ => None  
 }  
}

Результат работы программы:

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание