**ПАКЕТЫ И ИМПОРТ**

**Пакеты** в Kotlin **представляют логический блок**, который объединяет функционал, например, классы и функции, используемые для решения близких по характеру задач. Так, классы и функции, которые предназначены для решения одной задачи, можно поместить в один пакет, классы и функции для других задач можно поместить в другие пакеты.

Для определения пакета применяется ключевое слово **package**, после которого идет имя пакета: package email

package email

class Message(val text: String)

fun send(message: Message, address: String){

println("Message `${message.text}` has been sent to

$address")

}

Допустим, мы хотим использовать функционал этого пакета в другом файле. Для подключения сущностей из пакета необходимо применить директиву import.

**import email.\***

После названия пакета ставится точка и звездочка, тем самым импортируются все типы из этого пакета.

import email.\*

fun main() {

val myMessage = Message("Hello Kotlin")

send(myMessage, "tom@gmail.com")

}

Поскольку в начале файла импортированы все типы из пакета email, то мы можем использовать класс Message и функцию send в функции main.

Также можно импортировать типы, определенные в пакете, по отдельности:

import email.send

import email.Message

**ПСЕВДОНИМЫ**

С помощью оператора as можно определять псевдоним для подключаемого типа и затем обращаться к этому типу через его псевдоним:

import email.send as sendEmail

import email.Message as EmailMessage

fun main() {

val myMessage = EmailMessage("Hello Kotlin")

sendEmail(myMessage, "tom@gmail.com")

}

Здесь для функции send() определен псевдоним sendEmail. И

далее для обращения к этой функции надо использовать ее псевдоним:

sendEmail(myMessage, "tom@gmail.com")

Также для класса Message определен псевдоним EmailMessage.

Соответственно при использовании класса необходимо

**применять его псевдоним**, а не оригинальное имя:

val myMessage = EmailMessage("Hello Kotlin")

**Псевдонимы могут нам особенно пригодится, если у нас импортируются из разных пакетов типы с одним и тем же именем.**

**НАСЛЕДОВАНИЕ**

Наследование позволяет создавать классы, которые расширяют функциональность или изменяют поведение уже существующих классов.

В отношении наследования выделяются **два ключевых компонента**. Прежде всего это **базовый класс** (класс-родитель, родительский класс, суперкласс), который определяет базовую функциональность. И **производный класс** (класс- наследник, подкласс), который наследует функциональность базового класса и может расширять или модифицировать ее.

Чтобы функциональность класса можно было унаследовать, необходимо определить для этого класса аннотацию **open**. По умолчанию без этой аннотации класс не может быть унаследован.