Лабораторная работа №3

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Цель: познакомиться с особенностями объектно-ориентированного программирования. Научиться создавать собственные классы с использованием R6. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретические сведения

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Принципы ООП по Алану Кею

- все является объектом;
- вычисления осуществляются путем взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некоторое действие;
- объекты взаимодействуют, посылая и получая сообщения;
- сообщение – это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия;
- каждый объект имеет независимую память, которая состоит из других объектов;
- каждый объект является представителем (экземпляром) класса, который выражает общие свойства объектов.
- в классе задается поведение (функциональность) объекта.
- все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия;
- классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования
- память и поведение, связанное с экземплярами определенного класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.
- программа представляет собой набор объектов, имеющих состояние и поведение;
- устойчивость и управляемость системы обеспечивается за счет четкого разделения ответственности объектов (за каждое действие отвечает определенный объект), однозначного определения интерфейсов межобъектного взаимодействия и полной изолированности внутренней структуры объекта от внешней среды (инкапсуляции).

Механизмы ООП

Абстракция – придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы;

Инкапсуляция – можно скрыть ненужные внутренние подробности работы объекта от окружающего мира (алгоритмы работы хранятся вместе с данными);

Наследование – можно создавать специализированные классы на основе базовых (позволяет избегать написания повторного кода);

Полиморфизм – в разных объектах одна и та же операция может выполнять различные функции;

Композиция – объект может быть составным и включать другие объекты.

Некоторые понятия

Объект – абстракция данных;

Объект – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом;

Объект: тип, методы,

Данные – объекты и отношения между ними;

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт)

С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

Атрибут класса – содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса

Методы класса – функция, которая может выполнять какие-либо действия над данными (свойствами) класса.

Дженерик (обобщенная функция) – функция, способная принимать разные структуры данных (разные классы), и работающая по-разному с данными структурами.

Синтаксис создания дженериков (на примере языка R)

```
get_n_elements <- function(x, ...){
  UseMethod("get_n_elements")
}
```

Рисунок 1 – Создание дженерика (подсчет элементов)

```
get_n_elements.data.frame <- function(x, ...){
  return(nrow(x) * ncol(x))
}
```

Рисунок 2 – Описание работы с классом data.frame для дженерика (подсчет элементов)

```
get_n_elements.default <- function(x, ...){
  return(length(unlist(x)))
}
```

Рисунок 3 – Описание работы по умолчанию для дженерика (подсчет элементов)

```
vec_numbers <- rnorm(10, mean = 0, sd = 1)
class(vec_numbers) <- "norm_distrib"
class(vec_numbers)
```

Рисунок 4 – Определение собственного класса

Генератор классов – шаблон для создания объектов.

Для создания классов в виде сложных структур в языке программирования R рекомендуется использовать пакет R6. Для работы с пакетом его сперва необходимо установить, а затем подключить, используя стандартные функции install.packages() и library(). Следующим этапом является написание генератора класса с использованием функции R6Class() пакета R6. При написании генератора используется стиль, похожий на создание функций, за исключением того, что вместо function используется ключевое слово R6Class, в примере на рисунке 5 представлен генератор класса Thing.

```
ThingFactory <- R6Class(
  "Thing",
  private = list(
    a_field = "a value",
    another_field = 123
  )
)
```

Рисунок 5 – Создание генератора класса Thing

Генератор класса в качестве первого аргумента получает имя класса, далее в генератор передается до трех списков: private, active и public. Список active целесообразно рассматривать с точки зрения ограничения доступа, и использования данных исключительно на чтение (в рамках данного занятия не рассматривается). Имя класса рекомендуется писать в стиле UpperCamelCase.

Для создания нового объекта класса необходимо вызвать генератор с добавлением $new().

```
thing_factory_object <- ThingFactory$new()
```

Рисунок 6 – Создание объекта класса Thing

В ходе дальнейшего рассмотрения мы остановимся на двух списках: private и public, логика конструктора предполагает, что содержимое списка private является данными, а содержимое списка public – методами. На рисунке 7 показан пример создания метода в списке public который выводит на печать содержимое переменных, находящихся в списке private.

```
ThingFactory <- R6Class(
  "Thing",
  private = list(
    a_field = "a value",
    another_field = 123
  ),
  public = list(
    do_something = function(x, y, z) {
      # Доступ к приватным полям
      paste(
        private$a_field,
        private$another_field
      )
    }
  )
)
```

Рисунок 7 – Создание генератора класса Thing с двумя списками

При создании объектов бывает необходимо создавать их с некоторыми значениями (передавать параметры в список private). Для этого необходимо использовать специальный метод initialize() в списке public. Данный метод автоматически вызывается при вызове конструктора, но может быть переопределен пользователем (рисунок 8).

```
ThingFactory <- R6Class(
  "Thing",
  private = list(
    a_field = "a value",
    another_field = 123
  ),
  public = list(
    initialize = function(a_field, another_field) {
      if(!missing(a_field)) {
        private$a_field <- a_field
      }
      if(!missing(another_field)) {
        private$another_field <- another_field
      }
    },
    print = function() {
      print(paste0(private$a_field, " = ", private$another_field))
    }
  )
)
```

Рисунок 8 – Создание генератора класса Thing с определением метода initialize()

Для создания объекта с другими значениями необходимо передать эти значения в генератор как представлено на рисунке 9.

```
a_thing <- ThingFactory$new(
  a_field = "a different value",
  another_field = 456
)
```

Рисунок 9 – Создание объекта класса Thing с новыми значениями

Практическая часть

Задание 1. Создайте дженерик, принимающий вектор, содержащий параметры фигуры и вычисляющий ее площадь. Для разных фигур создайте разные классы. В качестве метода по умолчанию дженерик должен выводить сообщение о невозможности обработки данных.

Задание 2. Создайте генератор класса Микроволновая печь. В качестве данных класс должен содержать сведения о мощности печи (Вт) и о состоянии дверцы (открыта или закрыта). Данный класс должен обладать методами открыть и закрыть дверь микроволновки, а также методом, отвечающим за приготовление пищи. Метод, отвечающий за приготовление пищи, должен вводить систему в бездействие (используется Sys.sleep) на определенное количество времени (которое зависит от мощности печи) и после выводить сообщение о готовности пищи.

Выполните создание двух объектов этого класса со значением по умолчанию и с передаваемыми значениями. Продемонстрируйте работу этих объектов по приготовлению пищи.

Задание 3. Создайте класс копилка. Описание структуры классы выполните из своего понимания копилки.

Вопросы для контроля из материалов лабораторного занятия

1. Принципы ООП по Алану Кею
2. Механизмы ООП
3. Основные понятия ООП
4. Создание и назначение дженериков
5. Создание класса в R6
6. Структура класса в R6

Вопросы для поиска и письменного ответа

1. История появления ООП. Основные этапы
2. Связь ООП с другими парадигмами программирования
3. Чистые языки, реализующие концепцию ООП. История появления
4. Мультипарадигмальные языки, реализующие концепцию ООП. История появления