Отчет по лабораторной работе 3

Тема: Объектно-ориентированное программирование

Сведения о студенте

Дата: 2025-10-09
Семестр: 2 курс 1 полугодие - 3 семестр
Группа: ПИН-б-о-24-1 (2)
Дисциплина: Технологии программирования
Студент: Макаров Роман Дмитриевич

Цель работы

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

Теоретическая часть

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

Основные принципы ООП по Алану Кею:

- Все является объектом
- Вычисления осуществляются путем взаимодействия между объектами
- Каждый объект имеет независимую память
- Каждый объект — представитель класса, выражающего общие свойства
- Классы организованы в иерархию наследования

Ключевые механизмы ООП:

- **Инкапсуляция** — скрытие внутренней реализации объекта
- **Наследование** — создание новых классов на основе существующих
- **Полиморфизм** — возможность объектов с одинаковой спецификацией иметь различную реализацию
- **Абстракция** — выделение существенных характеристик объекта

Основные понятия:

- **Класс** — шаблон для создания объектов, определяющий их структуру и поведение
- **Объект** — экземпляр класса с конкретным состоянием
- **Атрибуты** — данные, характеризующие объект
- **Методы** — функции, определяющие поведение объекта

Реализация в R:

- **Пакет R6** — основной инструмент для создания классов
- **Генератор классов** — создается функцией `R6Class()`
- **Списки**:
  - `private` — для данных объекта
  - `public` — для методов
  - `active` — для свойств только для чтения
- **Метод initialize()** — конструктор для инициализации объектов
- **Дженерики** — функции, работающие с разными классами через механизм `UseMethod()`

Преимущества ООП:

- Повторное использование кода
- Упрощение сопровождения программ
- Четкое разделение ответственности
- Улучшение масштабируемости приложений

Выполненные задания

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.

Ход работы

Задание 1.

Программный код:

```
get_area <- function(x){
    UseMethod('get_area')
}


get_area.default <- function(x) {
    cat('Невозможно обработать данные.')
    return(NA)
}

create_rectangle <- function(length, width) {
    if (any(is.na(c(length, width))) || length <= 0 || width <= 0) {
        stop("Ошибка: длина и ширина должны быть положительными числами")
  }
  rectangle <- list(
  length = length,
  width = width,
  type = 'Прямоугольник'
  )
  class(rectangle) <- c('rectangle', 'view')
  return (rectangle)
}


create_parallelogram <- function(base, height) {
    if (any(is.na(c(base, height))) || base <= 0 || height <= 0) {
        stop("Ошибка: основание и высота должны быть положительными числами")
  }
  parallelogram <- list(
  base = base,
  height = height,
  type = 'Параллелограмм'
  )
  class(parallelogram) <- c('parallelogram', 'view')
  return (parallelogram)
}


create_circle <- function(radius) {
  if (is.na(radius) || radius <= 0) {
```

```
    stop("Ошибка: радиус должен быть положительным числом")
  }
  circle <- list(
  radius = radius,
  type = 'Круг'
  )
  class(circle) <- c('circle', 'view')
  return (circle)
}


get_area.rectangle <- function(x) {
    area <- x$length * x$width
    cat('Площадь прямоугольника: ', area, '\n')
    return(area)
}

get_area.parallelogram <- function(x) {
    area <- x$base * x$height
    cat('Площадь параллелограмма: ', area, '\n')
    return(area)
}

get_area.circle <- function(x) {
    area <- pi * (x$radius ^ 2)
    cat('Площадь круга: ', area, '\n')
    return(area)
}


# test

# создание фигур
rec <- create_rectangle(5, 2)
par <- create_parallelogram(2, 5)
cir <- create_circle(5)

# вычисление площадей фигур
area_rec <- get_area(rec)
area_par <- get_area(par)
area_cir <- get_area(cir)

# вызов  метода по умолчанию
cat('\n')
get_area()

# вывод информации о фигурах (классах фигур)
cat('\n')
print(rec)
cat('\n')
print(par)
cat('\n')
print(cir)
```

Пример работы программы:

```
Площадь прямоугольника:  10
Площадь параллелограмма:  10
Площадь круга:  78.53982

Невозможно обработать данные.[1] NA

$length
[1] 5

$width
[1] 2

$type
[1] "Прямоугольник"

attr(,"class")
[1] "rectangle" "view"

$base
[1] 2

$height
[1] 5

$type
[1] "Параллелограмм"

attr(,"class")
[1] "parallelogram" "view"

$radius
[1] 5

$type
[1] "Круг"

attr(,"class")
[1] "circle" "view"
}
```

Задание 2.

Программный код:

```
library(R6)

MicrowaveOven <- R6Class(
  "MicrowaveOven",
  private = list(
```

```
  power = NULL,      # мощность в Вт
  door_open = NULL   # состояние дверцы (TRUE - открыта, FALSE - закрыта)
),

public = list(
  initialize = function(power = 800, door_open = FALSE) {
    private$power <- power
    private$door_open <- door_open
    cat("Создана микроволновая печь. Мощность:", private$power, "Вт, Дверца:",
        ifelse(private$door_open, "открыта", "закрыта"), "\n")
  },

  # Метод для открытия дверцы
  open_door = function() {
    if (!private$door_open) {
      private$door_open <- TRUE
      cat("Дверца микроволновки открыта\n")
    } else {
      cat("Дверца уже открыта\n")
    }
    invisible(self)
  },

  # Метод для закрытия дверцы
  close_door = function() {
    if (private$door_open) {
      private$door_open <- FALSE
      cat("Дверца микроволновки закрыта\n")
    } else {
      cat("Дверца уже закрыта\n")
    }
    invisible(self)
  },

  # Метод для получения мощности
  get_power = function() {
    return(private$power)
  },

  # Метод для установки мощности
  set_power = function(new_power) {
    if (new_power > 0) {
      private$power <- new_power
      cat("Мощность изменена на:", private$power, "Вт\n")
    } else {
      cat("Ошибка: Мощность должна быть положительным числом\n")
    }
    invisible(self)
  },

  # Метод для получения состояния дверцы
  is_door_open = function() {
    return(private$door_open)
```

```
},

# Метод для приготовления пищи
cook_food = function(food_name = "еда") {
  # Проверка состояния дверцы
  if (private$door_open) {
    cat("ОШИБКА: Нельзя готовить с открытой дверцей! Закройте дверцу сначала.\n")
    return(FALSE)
  }

  # Расчет времени приготовления в зависимости от мощности
  # Формула: время обратно пропорционально мощности
  # Для мощности 800 Вт - стандартное время 60 секунд
  base_time <- 60    # базовое время в секундах
  base_power <- 800  # базовая мощность в Вт

  cooking_time <- base_time * (base_power / private$power)

  cat("\n=== Начинаем приготовление ===\n")
  cat("Блюдо:", food_name, "\n")
  cat("Мощность печи:", private$power, "Вт\n")
  cat("Расчетное время приготовления:", round(cooking_time, 1), "секунд\n")

  # Имитация процесса приготовления (с ограничением для демонстрации)
  demo_time <- min(cooking_time, 5)  # ограничиваем 5 сек для демонстрации
  cat("Приготовление... (ожидание", demo_time, "секунд)\n")

  # Прогресс-бар
  for(i in 1:10) {
    Sys.sleep(demo_time / 10)
    cat(".")
  }
  cat("\n")

  cat("=== Приготовление завершено ===\n")
  cat("Ваше блюдо '", food_name, "' готово! Приятного аппетита!\n\n")
  return(TRUE)
},

# Метод для отображения статуса
show_status = function() {
  status <- ifelse(private$door_open, "ОТКРЫТА", "ЗАКРЫТА")
  cat("Статус микроволновки:\n")
  cat("  Мощность:", private$power, "Вт\n")
  cat("  Дверца:", status, "\n")
},

# Метод для получения полной информации о состоянии
get_info = function() {
  return(list(
    power = private$power,
    door_open = private$door_open
  ))
```

```
    }
  )
)
```

Демонстрация работы программы:

```
# Демонстрация работы программы
cat("========================================\n")
cat("ДЕМОНСТРАЦИЯ РАБОТЫ МИКРОВОЛНОВЫХ ПЕЧЕЙ\n")
cat("========================================\n\n")

# Создание первого объекта со значениями по умолчанию
cat("1. СОЗДАНИЕ ПЕРВОЙ МИКРОВОЛНОВКИ (со значениями по умолчанию):\n")
microwave1 <- MicrowaveOven$new()  # мощность 800 Вт, дверца закрыта
cat("\n")

# Создание второго объекта с передаваемыми значениями
cat("2. СОЗДАНИЕ ВТОРОЙ МИКРОВОЛНОВКИ (с передаваемыми значениями):\n")
microwave2 <- MicrowaveOven$new(power = 1200, door_open = TRUE)
cat("\n")

# Демонстрация работы первой микроволновки
cat("3. РАБОТА С ПЕРВОЙ МИКРОВОЛНОВКОЙ (800 Вт):\n")
microwave1$show_status()

cat("\n3.1. Приготовление супа:\n")
microwave1$cook_food("суп")

cat("\n3.2. Открываем дверцу:\n")
microwave1$open_door()

cat("\n3.3. Попытка приготовить с открытой дверцей:\n")
microwave1$cook_food("суп")

cat("\n3.4. Закрываем дверцу и готовим картофель:\n")
microwave1$close_door()
microwave1$cook_food("картофель")

# Демонстрация работы второй микроволновки
cat("\n4. РАБОТА СО ВТОРОЙ МИКРОВОЛНОВКОЙ (1200 Вт):\n")
microwave2$show_status()

cat("\n4.1. Закрываем дверцу (изначально была открыта):\n")
microwave2$close_door()

cat("\n4.2. Готовим пиццу:\n")
microwave2$cook_food("пицца")

# Демонстрация геттеров и сеттеров
cat("\n5. РАБОТА С ГЕТТЕРАМИ И СЕТТЕРАМИ:\n")
cat("Текущая мощность microwave1:", microwave1$get_power(), "Вт\n")
cat("Дверца microwave1 открыта:", microwave1$is_door_open(), "\n")
```

```
cat("\nИзменяем мощность microwave1 на 1500 Вт:\n")
microwave1$set_power(1500)
microwave1$show_status()

cat("\n5.1. Готовим курицу с новой мощностью:\n")
microwave1$cook_food("курица")

# Сравнение времени приготовления для разных мощностей
cat("\n6. СРАВНЕНИЕ ВРЕМЕНИ ПРИГОТОВЛЕНИЯ:\n")

cat("\n6.1. Microwave1 (1500 Вт) - готовим рыбу:\n")
microwave1$cook_food("рыба")

cat("\n6.2. Microwave2 (1200 Вт) - готовим рыбу:\n")
microwave2$cook_food("рыба")

# Попытка установить некорректную мощность
cat("\n7. ПРОВЕРКА ВАЛИДАЦИИ ДАННЫХ:\n")
cat("Попытка установить мощность -100 Вт:\n")
microwave1$set_power(-100)

# Финальный статус всех микроволновок
cat("\n8. ФИНАЛЬНЫЙ СТАТУС ВСЕХ МИКРОВОЛНОВОК:\n")
cat("Микроволновка 1:\n")
microwave1$show_status()
cat("\nИнформация через get_info():")
print(microwave1$get_info())

cat("\nМикроволновка 2:\n")
microwave2$show_status()
cat("\nИнформация через get_info():")
print(microwave2$get_info())

cat("\n========================================\n")
cat("ДЕМОНСТРАЦИЯ ЗАВЕРШЕНА\n")
cat("========================================\n")
```

Результат демонстрации программы:

```
========================================
ДЕМОНСТРАЦИЯ РАБОТЫ МИКРОВОЛНОВЫХ ПЕЧЕЙ
========================================

1. СОЗДАНИЕ ПЕРВОЙ МИКРОВОЛНОВКИ (со значениями по умолчанию):
Создана микроволновая печь. Мощность: 800 Вт, Дверца: закрыта

2. СОЗДАНИЕ ВТОРОЙ МИКРОВОЛНОВКИ (с передаваемыми значениями):
Создана микроволновая печь. Мощность: 1200 Вт, Дверца: открыта

3. РАБОТА С ПЕРВОЙ МИКРОВОЛНОВКОЙ (800 Вт):
Статус микроволновки:
  Мощность: 800 Вт
  Дверца: ЗАКРЫТА
```

```
3.1. Приготовление супа:

=== Начинаем приготовление ===
Блюдо: суп
Мощность печи: 800 Вт
Расчетное время приготовления: 60 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' суп ' готово! Приятного аппетита!


3.2. Открываем дверцу:
Дверца микроволновки открыта

3.3. Попытка приготовить с открытой дверцей:
ОШИБКА: Нельзя готовить с открытой дверцей! Закройте дверцу сначала.

3.4. Закрываем дверцу и готовим картофель:
Дверца микроволновки закрыта

=== Начинаем приготовление ===
Блюдо: картофель
Мощность печи: 800 Вт
Расчетное время приготовления: 60 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' картофель ' готово! Приятного аппетита!


4. РАБОТА СО ВТОРОЙ МИКРОВОЛНОВКОЙ (1200 Вт):
Статус микроволновки:
  Мощность: 1200 Вт
  Дверца: ОТКРЫТА

4.1. Закрываем дверцу (изначально была открыта):
Дверца микроволновки закрыта

4.2. Готовим пиццу:

=== Начинаем приготовление ===
Блюдо: пицца
Мощность печи: 1200 Вт
Расчетное время приготовления: 40 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' пицца ' готово! Приятного аппетита!


5. РАБОТА С ГЕТТЕРАМИ И СЕТТЕРАМИ:
```

```
Текущая мощность microwave1: 800 Вт
Дверца microwave1 открыта: FALSE

Изменяем мощность microwave1 на 1500 Вт:
Мощность изменена на: 1500 Вт
Статус микроволновки:
  Мощность: 1500 Вт
  Дверца: ЗАКРЫТА

5.1. Готовим курицу с новой мощностью:

=== Начинаем приготовление ===
Блюдо: курица
Мощность печи: 1500 Вт
Расчетное время приготовления: 32 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' курица ' готово! Приятного аппетита!


6. СРАВНЕНИЕ ВРЕМЕНИ ПРИГОТОВЛЕНИЯ:

6.1. Microwave1 (1500 Вт) - готовим рыбу:

=== Начинаем приготовление ===
Блюдо: рыба
Мощность печи: 1500 Вт
Расчетное время приготовления: 32 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' рыба ' готово! Приятного аппетита!


6.2. Microwave2 (1200 Вт) - готовим рыбу:

=== Начинаем приготовление ===
Блюдо: рыба
Мощность печи: 1200 Вт
Расчетное время приготовления: 40 секунд
Приготовление... (ожидание 5 секунд)
..........
=== Приготовление завершено ===
Ваше блюдо ' рыба ' готово! Приятного аппетита!


7. ПРОВЕРКА ВАЛИДАЦИИ ДАННЫХ:
Попытка установить мощность -100 Вт:
Ошибка: Мощность должна быть положительным числом

8. ФИНАЛЬНЫЙ СТАТУС ВСЕХ МИКРОВОЛНОВОК:
Микроволновка 1:
```

```
Статус микроволновки:
  Мощность: 1500 Вт
  Дверца: ЗАКРЫТА

Информация через get_info():
$power
[1] 1500

$door_open
[1] FALSE


Микроволновка 2:
Статус микроволновки:
  Мощность: 1200 Вт
  Дверца: ЗАКРЫТА

Информация через get_info():
$power
[1] 1200

$door_open
[1] FALSE


========================================
ДЕМОНСТРАЦИЯ ЗАВЕРШЕНА
========================================
```

Задание 3.

Программный код:

```
library(R6)

PiggyBank <- R6Class(
  "PiggyBank",
  private = list(
    balance = 0,           # текущий баланс
    currency = "рубли",    # валюта
    capacity = 1000        # максимальная вместимость
  ),

  public = list(
    initialize = function(balance = 0, currency = "рубли", capacity = 1000) {
      # Проверка корректности входных данных
      if (balance < 0) {
        stop("Баланс не может быть отрицательным")
      }
      if (capacity <= 0) {
        stop("Вместимость должна быть положительной")
      }
```

```
  if (balance > capacity) {
    stop("Начальный баланс не может превышать вместимость")
  }

  private$balance <- balance
  private$currency <- currency
  private$capacity <- capacity

  cat("Создана копилка:\n")
  cat("  Баланс:", private$balance, private$currency, "\n")
  cat("  Валюта:", private$currency, "\n")
  cat("  Вместимость:", private$capacity, private$currency, "\n\n")
},

# Метод для добавления денег в копилку
add_money = function(amount) {
  if (amount <= 0) {
    cat("ОШИБКА: Сумма должна быть положительной!\n")
    return(FALSE)
  }

  if ((private$balance + amount) > private$capacity) {
    cat("ОШИБКА: Превышена вместимость копилки!\n")
    cat("         Можно добавить максимум:", private$capacity - private$balance, private$
    return(FALSE)
  }

  private$balance <- private$balance + amount
  cat("Успешно добавлено:", amount, private$currency, "\n")
  cat("Новый баланс:", private$balance, "/", private$capacity, private$currency, "\n\n")
  return(TRUE)
},

# Метод для извлечения денег из копилки
withdraw_money = function(amount) {
  if (amount <= 0) {
    cat("ОШИБКА: Сумма должна быть положительной!\n")
    return(0)
  }

  if (amount > private$balance) {
    cat("ОШИБКА: Недостаточно средств в копилке!\n")
    cat("         Доступно:", private$balance, private$currency, "\n")
    return(0)
  }

  private$balance <- private$balance - amount
  cat("Успешно извлечено:", amount, private$currency, "\n")
  cat("Остаток в копилке:", private$balance, "/", private$capacity, private$currency, "\n
  return(amount)
},

# Метод для проверки баланса
```

```
  check_balance = function() {
    cat("Текущий баланс:", private$balance, private$currency, "\n")
    cat("Вместимость:", private$capacity, private$currency, "\n")
    cat("Свободное место:", private$capacity - private$balance, private$currency, "\n")
    cat("Заполнено:", round((private$balance / private$capacity) * 100, 1), "%\n\n")
    return(private$balance)
  },

  # Метод для получения текущего баланса
  get_balance = function() {
    return(private$balance)
  },

  # Метод для получения валюты
  get_currency = function() {
    return(private$currency)
  },

  # Метод для получения вместимости
  get_capacity = function() {
    return(private$capacity)
  },

  # Метод для установки новой вместимости
  set_capacity = function(new_capacity) {
    if (new_capacity <= 0) {
      cat("ОШИБКА: Вместимость должна быть положительной!\n")
      return(FALSE)
    }

    if (private$balance > new_capacity) {
      cat("ОШИБКА: Новая вместимость меньше текущего баланса!\n")
      cat("         Текущий баланс:", private$balance, private$currency, "\n")
      return(FALSE)
    }

    private$capacity <- new_capacity
    cat("Вместимость изменена на:", private$capacity, private$currency, "\n")
    cat("Свободное место:", private$capacity - private$balance, private$currency, "\n\n")
    return(TRUE)
  },

  # Метод для отображения полной информации
  get_info = function() {
    info <- list(
      balance = private$balance,
      currency = private$currency,
      capacity = private$capacity,
      free_space = private$capacity - private$balance,
      fill_percentage = round((private$balance / private$capacity) * 100, 1)
    )

    cat("Информация о копилке:\n")
```

```
      cat("  Баланс:", info$balance, info$currency, "\n")
      cat("  Валюта:", info$currency, "\n")
      cat("  Вместимость:", info$capacity, info$currency, "\n")
      cat("  Свободное место:", info$free_space, info$currency, "\n")
      cat("  Заполнено:", info$fill_percentage, "%\n\n")

      return(info)
    }
  )
)
```

Демонстрация программы:

```
cat("========================================\n")
cat("ДЕМОНСТРАЦИЯ РАБОТЫ КОПИЛКИ\n")
cat("========================================\n\n")

# Создание первого объекта со значениями по умолчанию
cat("1. СОЗДАНИЕ ПЕРВОЙ КОПИЛКИ (со значениями по умолчанию):\n")
bank1 <- PiggyBank$new()
bank1$get_info()

# Создание второго объекта с передаваемыми значениями
cat("2. СОЗДАНИЕ ВТОРОЙ КОПИЛКИ (с передаваемыми значениями):\n")
bank2 <- PiggyBank$new(balance = 200, currency = "доллары", capacity = 5000)
bank2$get_info()

# Демонстрация работы с первой копилкой
cat("3. РАБОТА С ПЕРВОЙ КОПИЛКОЙ:\n")

cat("3.1. Добавляем 300 рублей:\n")
bank1$add_money(300)

cat("3.2. Добавляем 500 рублей:\n")
bank1$add_money(500)

cat("3.3. Пытаемся добавить 300 рублей (превышение лимита):\n")
bank1$add_money(300)

cat("3.4. Извлекаем 200 рублей:\n")
bank1$withdraw_money(200)

cat("3.5. Проверяем баланс:\n")
bank1$check_balance()

# Демонстрация работы со второй копилкой
cat("\n4. РАБОТА СО ВТОРОЙ КОПИЛКОЙ:\n")

cat("4.1. Добавляем 1000 долларов:\n")
bank2$add_money(1000)

cat("4.2. Добавляем 2000 долларов:\n")
bank2$add_money(2000)
```

```
cat("4.3. Извлекаем 500 долларов:\n")
bank2$withdraw_money(500)

cat("4.4. Пытаемся извлечь 5000 долларов (недостаточно средств):\n")
bank2$withdraw_money(5000)

cat("4.5. Изменяем вместимость на 10000 долларов:\n")
bank2$set_capacity(10000)

cat("4.6. Пытаемся установить вместимость 1000 долларов (меньше текущего баланса):\n")
bank2$set_capacity(1000)

# Работа с геттерами
cat("\n5. РАБОТА С ГЕТТЕРАМИ:\n")
cat("Баланс bank1:", bank1$get_balance(), bank1$get_currency(), "\n")
cat("Баланс bank2:", bank2$get_balance(), bank2$get_currency(), "\n")
cat("Вместимость bank2:", bank2$get_capacity(), bank2$get_currency(), "\n")

# Создание копилки с начальным балансом
cat("\n6. СОЗДАНИЕ КОПИЛКИ С НАЧАЛЬНЫМ БАЛАНСОМ:\n")
bank3 <- PiggyBank$new(balance = 150, currency = "евро", capacity = 1000)
bank3$get_info()

cat("6.1. Добавляем 100 евро:\n")
bank3$add_money(100)

cat("6.2. Извлекаем 50 евро:\n")
bank3$withdraw_money(50)

cat("6.3. Финальная информация о bank3:\n")
bank3$get_info()

cat("\n========================================\n")
cat("ДЕМОНСТРАЦИЯ ЗАВЕРШЕНА\n")
cat("========================================\n")
```

Результат демонстрации работы программы:

```
========================================
ДЕМОНСТРАЦИЯ РАБОТЫ КОПИЛКИ
========================================

1. СОЗДАНИЕ ПЕРВОЙ КОПИЛКИ (со значениями по умолчанию):
Создана копилка:
  Баланс: 0 рубли
  Валюта: рубли
  Вместимость: 1000 рубли

Информация о копилке:
  Баланс: 0 рубли
  Валюта: рубли
  Вместимость: 1000 рубли
```

```
  Свободное место: 1000 рубли
  Заполнено: 0 %

2. СОЗДАНИЕ ВТОРОЙ КОПИЛКИ (с передаваемыми значениями):
Создана копилка:
  Баланс: 200 доллары
  Валюта: доллары
  Вместимость: 5000 доллары

Информация о копилке:
  Баланс: 200 доллары
  Валюта: доллары
  Вместимость: 5000 доллары
  Свободное место: 4800 доллары
  Заполнено: 4 %

3. РАБОТА С ПЕРВОЙ КОПИЛКОЙ:
3.1. Добавляем 300 рублей:
Успешно добавлено: 300 рубли
Новый баланс: 300 / 1000 рубли

3.2. Добавляем 500 рублей:
Успешно добавлено: 500 рубли
Новый баланс: 800 / 1000 рубли

3.3. Пытаемся добавить 300 рублей (превышение лимита):
ОШИБКА: Превышена вместимость копилки!
         Можно добавить максимум: 200 рубли
3.4. Извлекаем 200 рублей:
Успешно извлечено: 200 рубли
Остаток в копилке: 600 / 1000 рубли

3.5. Проверяем баланс:
Текущий баланс: 600 рубли
Вместимость: 1000 рубли
Свободное место: 400 рубли
Заполнено: 60 %


4. РАБОТА СО ВТОРОЙ КОПИЛКОЙ:
4.1. Добавляем 1000 долларов:
Успешно добавлено: 1000 доллары
Новый баланс: 1200 / 5000 доллары

4.2. Добавляем 2000 долларов:
Успешно добавлено: 2000 доллары
Новый баланс: 3200 / 5000 доллары

4.3. Извлекаем 500 долларов:
Успешно извлечено: 500 доллары
Остаток в копилке: 2700 / 5000 доллары

4.4. Пытаемся извлечь 5000 долларов (недостаточно средств):
```

```
ОШИБКА: Недостаточно средств в копилке!
         Доступно: 2700 доллары
4.5. Изменяем вместимость на 10000 долларов:
Вместимость изменена на: 10000 доллары
Свободное место: 7300 доллары

4.6. Пытаемся установить вместимость 1000 долларов (меньше текущего баланса):
ОШИБКА: Новая вместимость меньше текущего баланса!
         Текущий баланс: 2700 доллары

5. РАБОТА С ГЕТТЕРАМИ:
Баланс bank1: 600 рубли
Баланс bank2: 2700 доллары
Вместимость bank2: 10000 доллары

6. СОЗДАНИЕ КОПИЛКИ С НАЧАЛЬНЫМ БАЛАНСОМ:
Создана копилка:
  Баланс: 150 евро
  Валюта: евро
  Вместимость: 1000 евро

Информация о копилке:
  Баланс: 150 евро
  Валюта: евро
  Вместимость: 1000 евро
  Свободное место: 850 евро
  Заполнено: 15 %

6.1. Добавляем 100 евро:
Успешно добавлено: 100 евро
Новый баланс: 250 / 1000 евро

6.2. Извлекаем 50 евро:
Успешно извлечено: 50 евро
Остаток в копилке: 200 / 1000 евро

6.3. Финальная информация о bank3:
Информация о копилке:
  Баланс: 200 евро
  Валюта: евро
  Вместимость: 1000 евро
  Свободное место: 800 евро
  Заполнено: 20 %


========================================
ДЕМОНСТРАЦИЯ ЗАВЕРШЕНА
========================================
```

Результаты

Выводы

1. В ходе выполнения лабораторной работы были успешно освоены принципы объектно-ориентированного программирования через практическое создание классов с использованием пакета R6. Основное внимание уделялось реализации ключевых механизмов ООП: инкапсуляции, наследования и полиморфизма, что подтвердило эффективность данного подхода для моделирования сложных систем.

2. Создание дженериков для работы с различными классами фигур продемонстрировало преимущества полиморфного поведения, позволяющего единообразно обрабатывать разнотипные объекты. Разработка классов "Микроволновая печь" и "Копилка" показала практическую ценность инкапсуляции данных и методов в рамках единой сущности.

3. Полученный опыт доказал, что объектно-ориентированное программирование обеспечивает четкое разделение ответственности между компонентами системы, повышает переиспользуемость кода и упрощает процесс модификации программного обеспечения. Приобретенные навыки работы с классами в R заложили основу для разработки более сложных и масштабируемых приложений.

Ответы на контрольные вопросы

1. **Принципы ООП по Алану Кею**

Ответ:

- 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).

2. **Механизмы ООП**

Ответ:
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.

3. **Основные понятия ООП**

Ответ:
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.

4. **Создание и назначение дженериков**

Ответ: Синтаксис создания дженериков (на примере языка R):

Создание дженерика (подсчет элементов):

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

Описание работы с классом data.frame для дженерика (подсчет элементов):

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

Описание работы по умолчанию для дженерика (подсчет элементов):

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

Определение собственного класса:

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

5. **Создание класса в R6**

Ответ:

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

6. **Структура класса в R6**

Ответ: 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.

Создание генератора класса Thing с определением метода initialize():

```
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))
    }
  )
)
```

Ответы на вопросы для поиска и письменного ответа

1. **История появления ООП. Основные этапы**

Ответ: Объектно-ориентированное программирование зародилось в 1960-х годах с появлением языка Simula, который ввел базовые концепции классов и объектов. Значительный вклад внес Алан Кей с языком Smalltalk в 1970-х, сформулировавший основные принципы ООП. В 1980-х годах ООП получило распространение с языками C++ и Objective-C, а в 1990-х - с Java и C#, что сделало его доминирующей парадигмой в промышленной разработке.

2. **Связь ООП с другими парадигмами программирования**

Ответ: ООП тесно связано со структурным программированием, наследуя от него базовые управляющие конструкции. Оно также пересекается с функциональным программированием через использование методов и замыканий. Аспектно-ориентированное программирование дополняет ООП, решая проблемы сквозной функциональности. В мультипарадигмальных языках ООП успешно сочетается с другими подходами, создавая гибридные модели.

3. **Чистые языки, реализующие концепцию ООП. История появления**

Ответ: К чистым объектно-ориентированным языкам относятся Smalltalk (1972), где все элементы являются объектами, включая числа и классы; Ruby (1995), сохранивший принцип "всё - объект"; Eiffel (1986) с строгой реализацией контрактного программирования; и Self (1986) как прототипный язык без классов. Эти языки последовательно реализуют принципы ООП без компромиссов.

4. **Мультипарадигмальные языки, реализующие концепцию ООП. История появления**

Ответ: C++ (1983) стал первым массовым мультипарадигмальным языком, сочетающим ООП с системным программированием. Python (1991) и JavaScript (1995) успешно интегрируют ООП с функциональным и императивным подходами. Java (1995) и C# (2000) развили модель ООП, добавив элементы компонентного и аспектного программирования. Современные языки как Kotlin (2011) и Swift (2014) демонстрируют эволюцию мультипарадигмального подхода.