Санкт-Петербургский государственный университет
Кафедра системного программирования

Rust и ООП

Дмитриевцев Алексей Сергеевич

Санкт-Петербург
2024

Парадигмы

Какие парадигмы вы знаете?

- Императивная
- Функциональное
- ***Объектно-ориентированная***

ООП абстрактно

- Сущности — объекты
- Взаимодействие объектов
- Объект содержит
  - Данные/Свойства
  - Методы

История ООП

Simula (1960-е годы прошлого века)

Smalltalk-72, Алан Кей: *«Основная идея при создании программ на Smalltalk заключается в том, чтобы определить классы, которые обрабатывают связи между объектами в созданной нами среде»*

ООП язык

Определение #1

Язык, похожий на C++ или Java

ООП язык

Определение #2

Язык, позволяющий создавать программы, основанные на использовании системы объектов.

ООП язык

Определение #3

Язык, позволяющий реализовать наследование, инкапсуляцию и полиморфизм объектов.

C vs Objective-C Инкапсуляция

```
static void f1 () {
  puts("f1");
}

void f2 () {
  puts("f2");
}
```

main.c

```
int main() {
    f1();
    f2();
    return 0;
}
```

Распределим по файлам, структурам и получим инкапсуляцию.

C vs Objective-C Инкапсуляция

```
//class.h

@interface MyClass : NSObject {}
- (void)publicMethod;
@end

//main.m

@interface MyClass()
- (void)privateMethod;
@end

@implementation MyClass
- (void)publicMethod {
    puts("public");
}

- (void)privateMethod {
    puts("private");
}
@end
```

```
//main.m

int main ()
{
    MyClass * exClass = [[MyClass
alloc] init];
    [exClass publicMethod];
    [exClass privateMethod];
    return 0;
}
```

C vs Objective-C Наследование

Объявим структуру внутри структуры:

```
typedef struct struct_1 {
  int arg1, arg2;
}

typedef struct struct_2 {
    struct struct_1;
    int arg3;
} struct_2;
```

Есть нюансы относительно обращения к полям внутренний структуры напрямую, но мы их опустим

C vs Objective-C Наследование

```
@interface ClassParent : NSObject {
   NSString arg1;
   NSString arg2;
}

@end

@interface ClassChildren : ClassParent {
   NSString arg3;
}

@end
```

C vs Objective-C Полиморфизм

Начиная с C11 добавили _Generic:

```
//structs.h
#define typename(x) _Generic((x), \
  struct_1   : "struct_1", \
  struct_2   : "struct2", \
)

//main.c
int main() {
  struct_1 *example = newStruct_1();
  puts(typename(example));
  return 0;
}
```

C vs Objective-C Полиморфизм

Начиная с C11 добавили _Generic:

```
//structs.h
#define typename(x) _Generic((x), \
  struct_1   : "struct_1", \
  struct_2   : "struct2", \
)

//main.c
int main() {
  struct_1 *example = newStruct_1();
  puts(typename(example));
  return 0;
}
```

Итоги сравнения?

Выходит, чистый C — полноценный ООП язык?

Давайте посмотрим определения далее

ООП язык

Определение #4

Язык, построенный на принципах объектно-ориентированного программирования.

В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

ООП язык

Определение #5

Object-oriented language. A programming language that allows the user to express a program in terms of objects and messages between those objects. Examples include Smalltalk and LOGO.

IEEE Std 610.12-1990

Итоги сравнения?

Чистый C — ООП язык?

Ответ зависит от того, какой смысл автор вопроса вкладывает в слова «ООП язык»

Rust и ООП

Rust — ООП язык?

Ответ зависит от того, какой смысл автор вопроса вкладывает в слова «ООП язык». Теперь давайте разберемся, как с расте обстоят дела с ООП.

Rust и ООП

Базовый Класс выглядит так:

```
struct Analyser {
    dict: Vec<DictRecord>,
    chunk_ids: Vec<usize>,
    chunks: Vec<Vec<u8>>,
}
impl Analyser {
        fn fbc_dedup(&mut self) → i32 {
                //реализация метода
        }
}
```

Инкапсуляция

Вспомним работу с модулями. Здесь все также

```
pub struct Analyser {
    dict: Vec<DictRecord>,
    chunk_ids: Vec<usize>,
    chunks: Vec<Vec<u8>>,
}
impl Analyser {
        pub fn fbc_dedup(&mut self) → i32 {
                //реализация метода
        }
}
```

Наследование

- Наследование реализации
- Наследование интерфейса

По умолчанию только наследование интерфейса.

Что такое интерфейс?

А если очень хочется наследовать реализацию?

Наследование

```
trait Animal {
    fn name(&self) ->
String;
}

trait Pet: Animal {
    fn breed(&self) ->
String;
}

struct MyCat {
    nickname: String,
    color: String,
}
```

```
impl Pet for MyCat {
    fn breed() -> String {
        “Serval“
    }
}

impl Animal for MyCat {
    fn name(&self) -> String
{
        self.nickname;
    }
}
```

Множественное наследование и супертрейты

```
trait Animal {
    fn name(&self) -> String;
}

trait Pet: Animal {
    fn breed(&self) -> String;
}

trait Mammal: Animal {
    fn get_age(&self) -> u32;
}

trait HouseCat: Mammal + Pet {
    fn get_owner(&self) -> String
}
```

Default Implementations

```
pub trait Summary {
    fn summarize_angles(&self) -> String;
    fn summarize(&self) -> String {
     format!("(Sum of angles is: {})",
self.summarize_angles())
    }
}
```

Мы должны определить summarize_angles, чтобы использовать summarize.

Что дальше?

- Полиморфизм
- Основы Объектно-Ориентированного Проектирования
- Отдельные особенности ООП в расте

Ваши отзывы