Динамическая диспетчеризация

Объектно-Ориентированное Программирование

Иван Трепаков

NSU

Полиморфизм

Полиморфизм — возможность функции с одним именем иметь разные реализации.

Полиморфизм

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

Полиморфизм

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = ...;
var y = ...;
var z = x + y; // ???
```

Полиморфизм

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = 1;     // int
var y = ...;
var z = x + y; // ???
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = 1;     // int
var y = 2;     // int
var z = x + y; // ???
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = 1;     // int
var y = 2;     // int
var z = x + y; // 3 (int)
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = 1;     // int
var y = 2.0;   // double
var z = x + y; // ???
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
var x = 1;     // int
var y = 2.0;   // double
var z = x + y; // 3.0 (double)
```

Полиморфизм

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2)
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2) // ints: 3
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2) // ints: 3
add("1", 2)
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2) // ints: 3
add("1", 2) // mixed: 12
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2) // ints: 3
add("1", 2) // mixed: 12
add("1", "2")
```

Ad hoc полиморфизм

Ad hoc (букв. «к этому») — латинская фраза, означающая «для данного случая», «специально для этого». *Wikipedia*

- Выбор реализации делается в зависимости от количества и типов формальных параметров функции
  - Перегрузка функций в Java
  - Перегрузка операторов в C++

```
String add(int x, int y) {
  return "ints: " + (x + y);
}
String add(String x, int y) {
  return "mixed: " + (x + y);
}
String add(String x, String y) {
  return "strings: " + (x + y);
}
add(1, 2) // ints: 3
add("1", 2) // mixed: 12
add("1", "2") // strings: 12
```

Параметрический полиморфизм

Параметрический полиморфизм

- Реализация функции использует *обобщенный* параметр
  - Generics в Java
  - Templates в C++
  - Type classes в Haskell
- Можно задавать дополнительные ограничения на обобщенный тип
- *Подробнее на следующей лекции*

Полиморфизм

Параметрический полиморфизм

- Реализация функции использует *обобщенный* параметр
  - Generics в Java
  - Templates в C++
  - Type classes в Haskell
- Можно задавать дополнительные ограничения на обобщенный тип
- *Подробнее на следующей лекции*

```
static <T, S> T foo(T x, S y) {
  return x;
}

static <T extends I> boolean bar(I x, I y) {
  return x.test(y);
}

interface I {
  boolean test(I x);
}
```

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T
  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...;
x.foo();   // ???
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* S `<:` T

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <:` B если `class C extends` B
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new A()
x.foo();   // ???
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new A()
x.foo();   // A.foo
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new B()
x.foo();   // ???
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new B()
x.foo();   // B.foo
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T

  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new C()
x.foo();   // ???
```

[1] Даже для массивов, но это другая история

Полиморфизм подтипов

- Отношение *подтипа* `S <: T`

  Тип S совместим по присваиванию с типом T
  - `int <: long`
  - `C <: B` если `class C extends B`
- В общем случае отношение подтипа *независимо* от наследования
- Но для ссылочных типов в Java совпадает[1]
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - *Переопределение* методов

```
class A {
  void foo() { println(”A.foo”); }
}
class B extends A {
  void foo() { println(”B.foo”); }
}
class C extends A { }

A x = ...; // new C()
x.foo();   // A.foo
```

[1] Даже для массивов, но это другая история

Диспетчеризация

Статическая

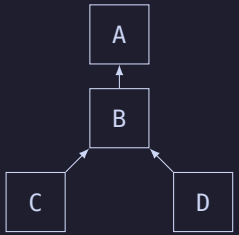
Динамическая

Одиночная

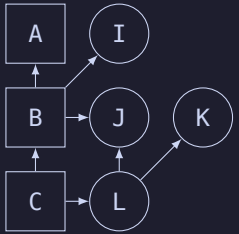
Множественная

Наследование

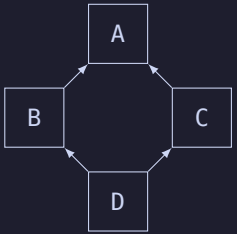
Одиночное



Интерфейсное



Множественное



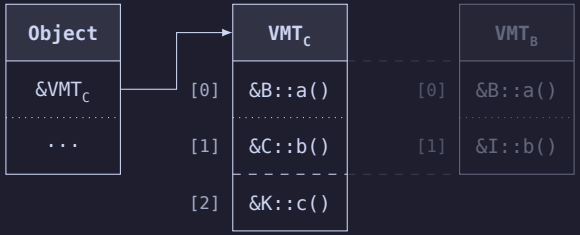
simula

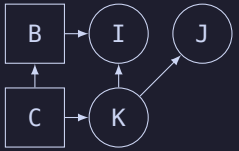
Java

OCaml

Eiffel

Диспетчеризация виртуальных методов

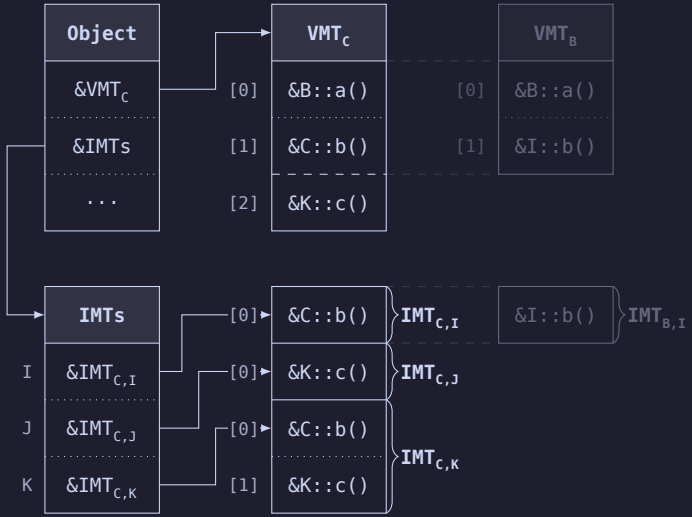


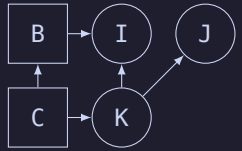


Виртуальный вызов `x.b()`

```
// Формальныйтип x: C
call x.vmt[vnum_C,b]
```

Диспетчеризация виртуальных методов





Виртуальный вызов x.b()

```
// Формальныйтип x: C
call x.vmt[vnum_C,b]
```

Интерфейсный вызов x.b()

```
// Формальныйтип x: I
imt_C,I := x.imts.find(&I)
call imt_C,I[vnum_I,b]
```

Диспетчеризация интерфейсных методов

Таблица интерфейсных методов

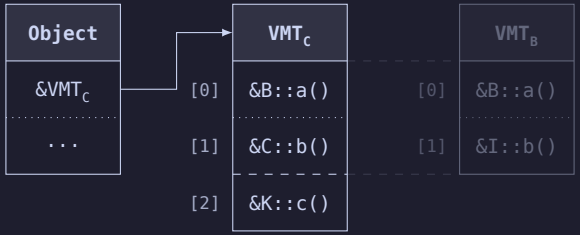
Диспетчеризация интерфейсных методов

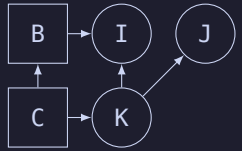
Полиморфный инлайн кэш

Заключение

Q&A

Таблица виртуальных методов

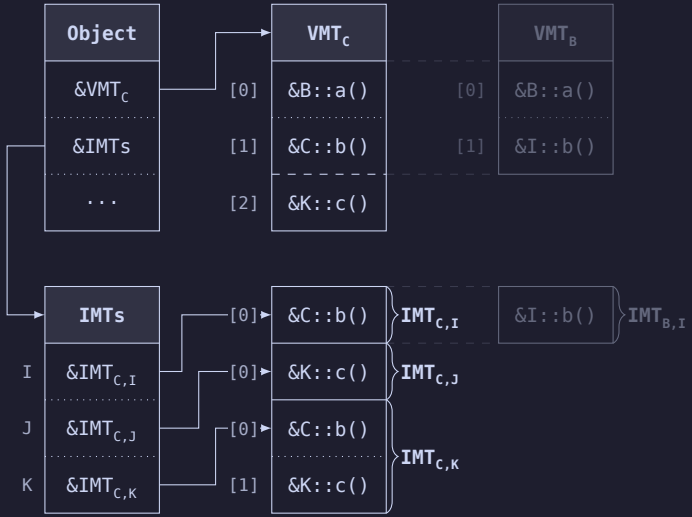


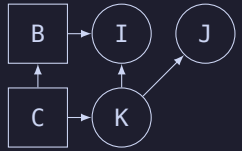


Виртуальный вызов x.b()

```
// Формальныйтип x: C
call x.vmt[vnum_C,b]
```

Таблица виртуальных методов





Виртуальный вызов x.b()

```
// Формальныйтип x: C
call x.vmt[vnum_C,b]
```

Интерфейсный вызов x.b()

```
// Формальныйтип x: I
imt_C,I := x.imts.find(&I)
call imt_C,I[vnum_I,b]
```