

# Динамическая диспетчеризация

## Объектно-Ориентированное Программирование

Иван Трепаков

NSU

*Полиморфизм — возможность функции с одним именем иметь разные реализации*

## Ad hoc полиморфизм

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

[Wikipedia](#)

## 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; // ???
```

## 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)
```

## 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*

## 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  $S <: T$   
*Любое значение типа  $S$  является значением  
типа  $T$*

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - Переопределение методов



## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - Переопределение методов

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

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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();    // ???
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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();   // ???
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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();   // ???
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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();   // ???
```

## Полиморфизм подтипов

- Отношение подтипа  $S <: T$   
*Любое значение типа  $S$  является значением типа  $T$*
- В объектно-ориентированных языках отношение подтипа обычно совпадает с отношением наследования
- Выбор реализации делается в зависимости от реального типа объекта *receiver* в момент исполнения вызова
  - Виртуальные методы
  - 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
```



*Полиморфизм — возможность функции с одним именем иметь разные реализации*

*Полиморфизм — возможность функции с одним именем иметь разные реализации*

*Диспетчеризация — процесс выбора реализации полиморфной функции*



## Статическая диспетчеризация

- Реализация известна до исполнения
    - Прямой вызов (например статический)
    - Перегрузка
    - *Девиртуализация* в компиляторе
  - Эффективно без поддержки среды исполнения
-

## Статическая диспетчеризация

- Реализация известна до исполнения
  - Прямой вызов (например статический)
  - Перегрузка
  - *Девиртуализация* в компиляторе
- Эффективно без поддержки среды исполнения

---

`C.foo()`       $\longrightarrow$       `call &C::foo()`

## Статическая диспетчеризация

- Реализация известна до исполнения
  - Прямой вызов (например статический)
  - Перегрузка
  - *Девиртуализация* в компиляторе
- Эффективно без поддержки среды исполнения

## Динамическая диспетчеризация

- Реализация известна только во время исполнения конкретного вызова
  - Виртуальный вызов
  - Интерфейсный вызов
- Требуется поддержка среды исполнения

---

`C.foo()`       $\longrightarrow$       `call &C::foo()`

## Статическая диспетчеризация

- Реализация известна до исполнения
  - Прямой вызов (например статический)
  - Перегрузка
  - *Девиртуализация* в компиляторе
- Эффективно без поддержки среды исполнения

---

`C.foo()`       $\longrightarrow$       `call &C::foo()`

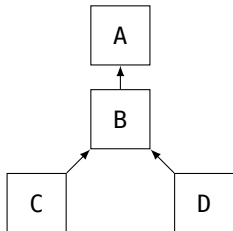
## Динамическая диспетчеризация

- Реализация известна только во время исполнения конкретного вызова
  - Виртуальный вызов
  - Интерфейсный вызов
- Требуется поддержка среды исполнения

---

`x.foo()`       $\longrightarrow$       ???

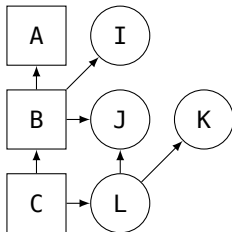
Одиночное



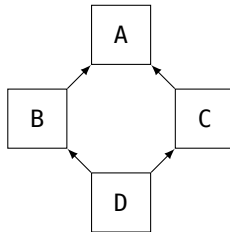
simula



Интерфейсное



Множественное



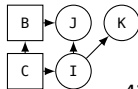
 **Eiffel**





## Таблица виртуальных методов

*Virtual Method Table, VMT, vtable*



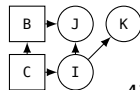
# Диспетчеризация виртуальных методов

## Таблица виртуальных методов

*Virtual Method Table, VMT, vtable*

- Массив с адресами реализаций виртуальных методов класса

$VMT_c$
$\&B::a()$
$\&C::b()$
$\&K::c()$

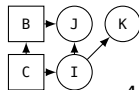
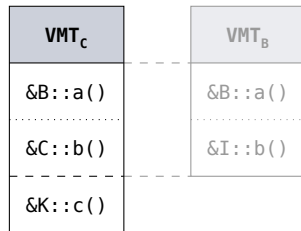


# Диспетчеризация виртуальных методов

## Таблица виртуальных методов

*Virtual Method Table, VMT, vtable*

- Массив с адресами реализаций виртуальных методов класса
- Таблицы подклассов расширяют таблицу суперкласса

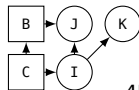
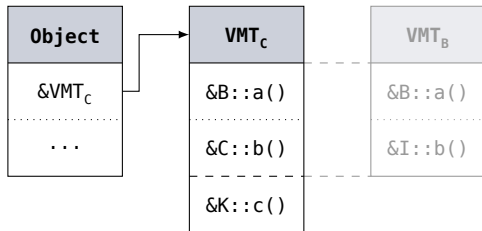


# Диспетчеризация виртуальных методов

## Таблица виртуальных методов

*Virtual Method Table, VMT, vtable*

- Массив с адресами реализаций виртуальных методов класса
- Таблицы подклассов расширяют таблицу суперкласса
- Таблица доступна напрямую из каждого объекта класса

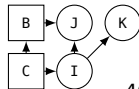
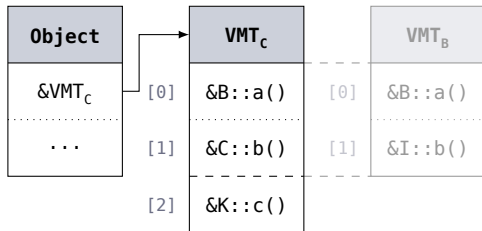


# Диспетчеризация виртуальных методов

## Таблица виртуальных методов

*Virtual Method Table, VMT, vtable*

- Массив с адресами реализаций виртуальных методов класса
- Таблицы подклассов расширяют таблицу суперкласса
- Таблица доступна напрямую из каждого объекта класса
- Для каждого виртуального метода можно определить *виртуальный номер*



# Диспетчеризация виртуальных методов

## Таблица виртуальных методов

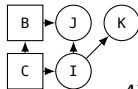
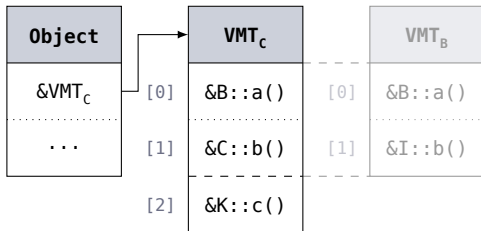
*Virtual Method Table, VMT, vtable*

- Массив с адресами реализаций виртуальных методов класса
- Таблицы подклассов расширяют таблицу суперкласса
- Таблица доступна напрямую из каждого объекта класса
- Для каждого виртуального метода можно определить *виртуальный номер*

Виртуальный вызов  $x.b()$

// Формальный тип  $x$  - класс  $B$

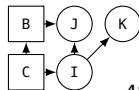
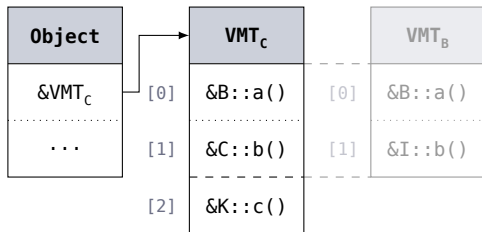
call  $x.vmt[vnum_{B,b}]$



# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*



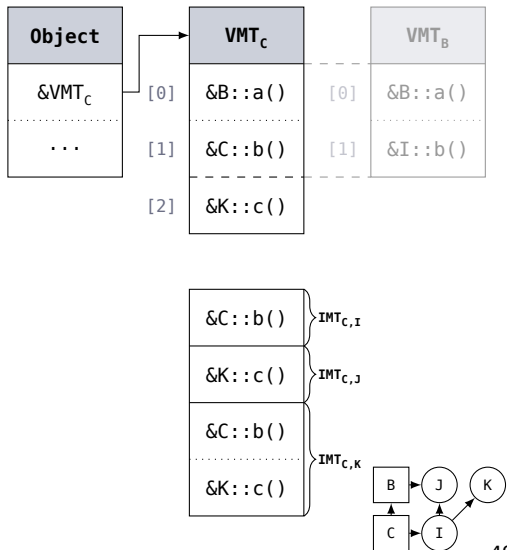


# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

- Массив с адресами реализаций методов

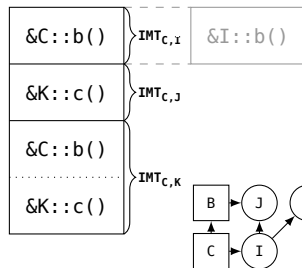
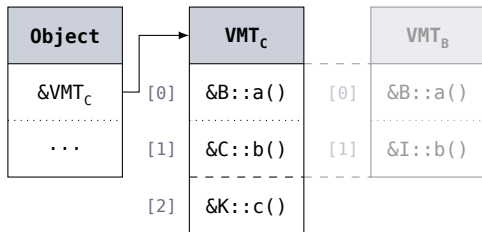


# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

- Массив с адресами реализаций методов
- Раскладка фиксирована во всех реализующих классах

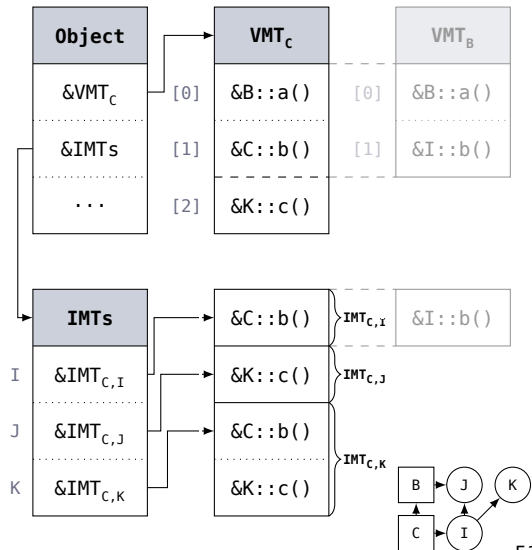


# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

- Массив с адресами реализаций методов
- Раскладка фиксирована во всех реализующих классах
- Таблицы доступны из каждого объекта

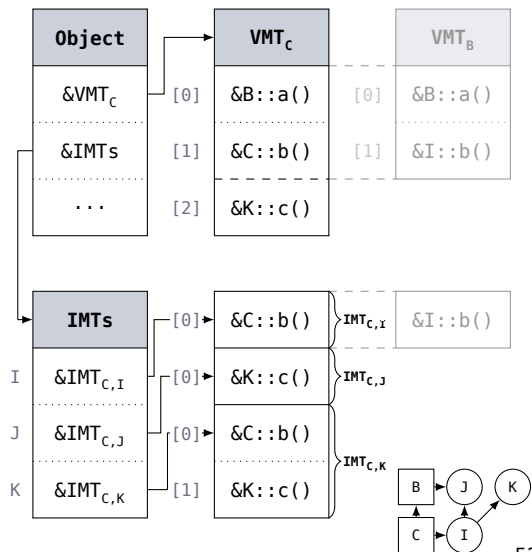


# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

- Массив с адресами реализаций методов
- Раскладка фиксирована во всех реализующих классах
- Таблицы доступны из каждого объекта
- Для каждого интерфейсного метода можно определить *виртуальный номер*

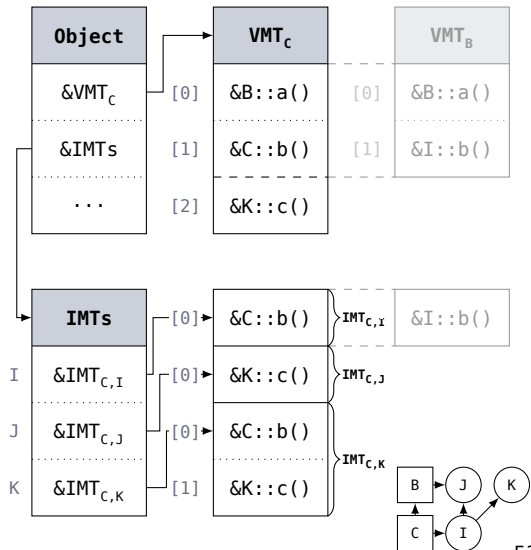


# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

- Массив с адресами реализаций методов
- Раскладка фиксирована во всех реализующих классах
- Таблицы доступны из каждого объекта
- Для каждого интерфейсного метода можно определить *виртуальный номер*
- Необходим поиск нужной таблицы



# Диспетчеризация интерфейсных методов

## Таблица интерфейсных методов

*Interface Method Table, IMT, itable*

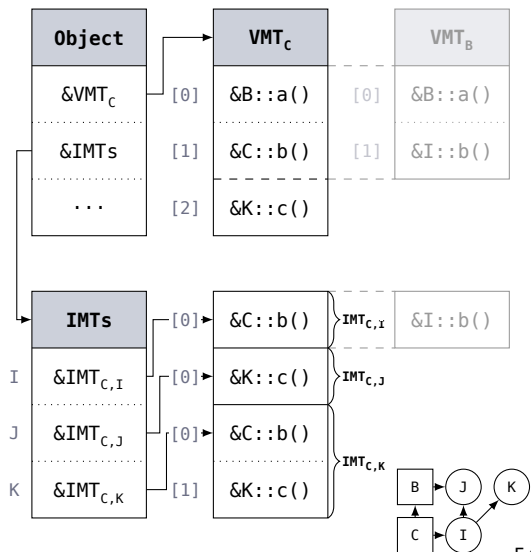
- Массив с адресами реализаций методов
- Раскладка фиксирована во всех реализующих классах
- Таблицы доступны из каждого объекта
- Для каждого интерфейсного метода можно определить *виртуальный номер*
- Необходим поиск нужной таблицы

## Интерфейсный вызов $x.b()$

// Формальный тип  $x$  - интерфейс  $I$

$imt_{C,I} = x.imts.find(\&I)$

call  $imt_{C,I}[vnum_{I,b}]$



## Что узнали?

- Виды полиморфизма
  - Ad hoc полиморфизм
  - Параметрический полиморфизм
  - Полиморфизм подтипов
- Виды диспетчеризации
  - Статическая
  - Динамическая
- Реализация динамической диспетчеризации
  - Таблица виртуальных методов для одиночного наследования
  - Таблица интерфейсных методов для интерфейсного наследования

## Что узнали?

- Виды полиморфизма
  - Ad hoc полиморфизм
  - Параметрический полиморфизм
  - Полиморфизм подтипов
- Виды диспетчеризации
  - Статическая
  - Динамическая
- Реализация динамической диспетчеризации
  - Таблица виртуальных методов для одиночного наследования
  - Таблица интерфейсных методов для интерфейсного наследования

## Что осталось за кадром?

- Параметрический полиморфизм на следующей лекции
- Диспетчеризации в полноценном множественном наследовании (например, в C++)
- Оптимизации компилятора
  - Открытая подстановка
  - Девиртуализация
  - Условная девиртуализация
  - Profile-Guided Optimization (PGO)
- Оптимизации времени исполнения
  - Polymorphic inline cache (PIC)
  - Профилировка
  - Деоптимизация



Q&A