Интерфейсы и типы данных (часть 1)

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

9 апреля 2015 г.

Типы данных в программировании

Интерпретация строки в C++ как разных типов данных:

Тип	**Представление**
std::string	"Datatype"
signed char[8]	68 97 116 97 116 121 112 101
char[8]	D a t a t y p e
short[4]	24900 24948 31092 25968
int[2]	1635017028 1701869940
long	7309475735980499268
float[2]	2.81751e+20 7.09755e+22
double	4.27256e+180

Каков смысл заданной последовательности байт? Какие операции к ней применимы?

Система типов

Определение

Тип данных — множество представимых в машинной памяти величин, которые характеризуются своими свойствами и производимыми над ними операциями.

Определение

Система типов языка программирования — совокупность правил, определяющих свойство типа для конструкций языка (переменных, выражений, функций, модулей, …).

Цели системы типов:

- **основная:** определение интерфейсов для взаимодействия с частями программы и обеспечение их корректного использования с целью устранения ошибок;
- обеспечение функциональности языка (напр., динамическая диспетчеризация в ООП);
- повышение доступности программы для понимания.

Типы данных и интерфейсы

Роль ТД в определении интерфейсов:

- **Функциональные ЯП:**
  спецификация допустимых аргументов и возвращаемых результатов для определяемых функций.
- **Процедурные ЯП:**
  то же + спецификация глобального состояния программы.
- **Объектно-ориентированные ЯП:**
  спецификация классов / объектов в виде набора допустимых методов для работы с инкапсулированными данными.

Интерфейсы классов

Области видимости интерфейса:

- публичные (англ. *public*) операции — доступны всем клиентам;
- защищенные (англ. *protected*) операции — доступны классу и подклассам;
- частные (англ. *private*) операции — доступны только классу;
- внутренние (англ. *package-wide*) операции — доступны классам, находящимся в одном модуле с данным (методы без спецификатора видимости в Java, **internal** в C#).

Способ определения:

- методы;
- свойства;
- точки расширения (напр., события).

Стандарт ISO/IEC 11404

Стандарт ISO/IEC 11404 «Типы данных общего назначения» (англ. *General purpose datatypes*) — содержит описание типов данных, независимое от языка программирования.

Содержание стандарта:

- спецификация для формального описания типов данных, независимого от реализации (ЯП и среды выполнения);
- определение базовых типов данных (содержимое + допустимые операции);
- способы генерации новых типов данных из базовых.

Классификация типов данных

Типы данных по способу определения:

- примитивные — задаваемые через аксиомы, т. е. независимые от других определений типов данных;
- генерируемые (англ. *generated*) — определяемые через другие типы данных.

По структуре:

- атомарные — ТД, значения которых семантически неделимы (напр., целые числа);
- агрегационные — ТД со сложной внутренней структурой (напр., массивы). Подвиды:
  - однородные / неоднородные — компоненты структуры имеют один / разные типы;
  - фиксированный / переменный размер;
  - уникальность и упорядоченность компонент;
  - метод доступа (произвольный / последовательный).

Общие свойства типов

Операция сравнения ($=$):

- универсальная: $\forall a, b \in T \quad (a = b) \vee (a \neq b)$;
- непротиворечивая: $\nexists a, b \in T \quad (a = b) \wedge (a \neq b)$;
- тривиальная: $\forall a \in T \quad a = a$;
- коммутативная: $\forall a, b \in T \quad (a = b) \Leftrightarrow (b = a)$;
- транзитивная: $\forall a, b, c \in T \quad (a = b) \wedge (b = c) \Rightarrow (a = c)$.

Примечание. Во многих ЯП сущесвуют две операции сравнения — по содержимому и по ссылке (`.equals()` и `==` в Java, `==` и `is` в Python).

Необязательная операция упорядочивания ($\leqslant$):

- универсальная: $\forall a, b \in T \quad (a \leqslant b) \vee (a \leqslant b)$;
- связанная с равенством: $\forall a, b \in T \quad (a \leqslant b) \wedge (a \leqslant b) \Rightarrow (a = b)$;
- транзитивная: $\forall a, b, c \in T \quad (a \leqslant b) \wedge (b \leqslant c) \Rightarrow (a \leqslant c)$.

Общие свойства типов

(Не)ограниченность:

- Тип данных T называется ограниченным сверху, если в нем задана операция упорядочивания и существует значение $U \in T$: $\forall a \in T \quad a \leqslant U$.
- Ограниченность снизу: $\exists L \in T : \forall a \in T \quad L \leqslant a$.

Аппроксимация:

- точные типы;
- типы, в которых вычислительная модель налагает ограничения на точность операций (напр., числа с плавающей запятой согласно стандарту IEEE 754).

Мощность:

- конечное множество значений;
- счетное множество значений;
- несчетное множество значений (требуют аппроксимации, т. к. представление в ЭВМ несчетного множества значений невозможно).

Примитивные типы данных

- **Булев тип.**

  `bool` в C#; **`boolean`** в Java; **`bool`** в Python.

- **Состояние** (англ. *state*) — конечное множество различимых неупорядоченных значений.

- **Перечисление** — конечное множество упорядоченных значений.

  `enum` в C / C++, C#, Java.

- **Символ.**

  `char` в C / C++, C#, Java.

- **Ординальный тип** (англ. *ordinal*): `"first"`, `"second"`, `"third"`, …

- **Дата и время.**

  `DATETIME` в SQL.

- **Целые числа.**

  ∼BigInteger в Java, **`long`** в Python.

Примитивные типы данных (продолжение)

- **Рациональные числа.**

  `fractions.Fraction` в Python.

- **Масштабированный тип** (англ. *scaled*) — рациональные числа с фиксированным знаменателем, напр., валютный тип (знаменатель равен 100).

  ∼BigDecimal в Java, `decimals.Decimal` в Python.

- **Действительные числа.**

  `float` и **`double`** в C++, C#, Java, **`float`** в Python.

- **Комплексные числа.**

  `complex` в Python.

- **Пустой тип** (англ. *void*) — объект, необходимый с точки зрения синтаксиса или семантики, не содержащий информации.

  `void` в C++, C#, Java.

Подтипы

Подтипы типов данных:

- диапазон значений;
- перечисление значений из базового типа;
- исключение значений из базового типа;
- ограничение размера базового типа-коллекции;
- расширение базового типа.

Примеры: целочисленные типы в ЯП, напр., **`short`**, **`int`** и **`long`** в C++, C#, Java; **`int`** в Python.

Генерируемые типы данных

Генерируемые типы данных:

- **Выбор** (англ. *choice*) — значение типа принадлежит одному из конечного набора базовых типов, в зависимости от состояния.

  `union` в C / C++.

- **Указатель** на другой тип.

  Указатели и ссылки в C / C++.

- **Процедурный тип**, задаваемый типами и направлением аргументов (входные / выходные) и типом возвращаемого значения.

  Функции в функциональных ЯП; в Python, JavaScript (без выделения подтипов).

Агрегационные типы данных

- **Запись** (англ. *record*) — компоненты (значения различных типов) определяются с помощью именованных полей.
- **Класс** (англ. *class*) — компоненты (значения различных типов, а также методы) определяются с помощью именованных полей; определены наследование и перегрузка компонентов.
- **Множество** (англ. *set*) — имплементация математического понятия подмножества пространства базового типа.
- **Мешок** (англ. *bag*) — неупорядоченный набор с повторениями значений базового типа.

Агрегационные типы данных (продолжение)

- **Последовательность** (англ. *sequence*) — упорядоченный набор значений базового типа с повторениями (упорядочивание задано извне).
  - **Строка** (англ. *string*) — последовательность символов.
- **Массив** (англ. *array*) — отображение между конечным пространством индексов (декартово произведение одного или более конечных типов) и значениями типа элементов.
- **Таблица** (англ. *table*) — конечное множество ассоциаций, заданное на декартовом произведении базовых типов ($\sim$ реляционная таблица SQL).

Примеры определений ТД

- Целочисленный тип со знаком, занимающий 4 байта памяти:

```
type Int = new integer range (-2147483648 .. 2147483647)
```

- Строка символов Unicode:

```
type String = new sequence of character ({ iso standard 10646 })
```

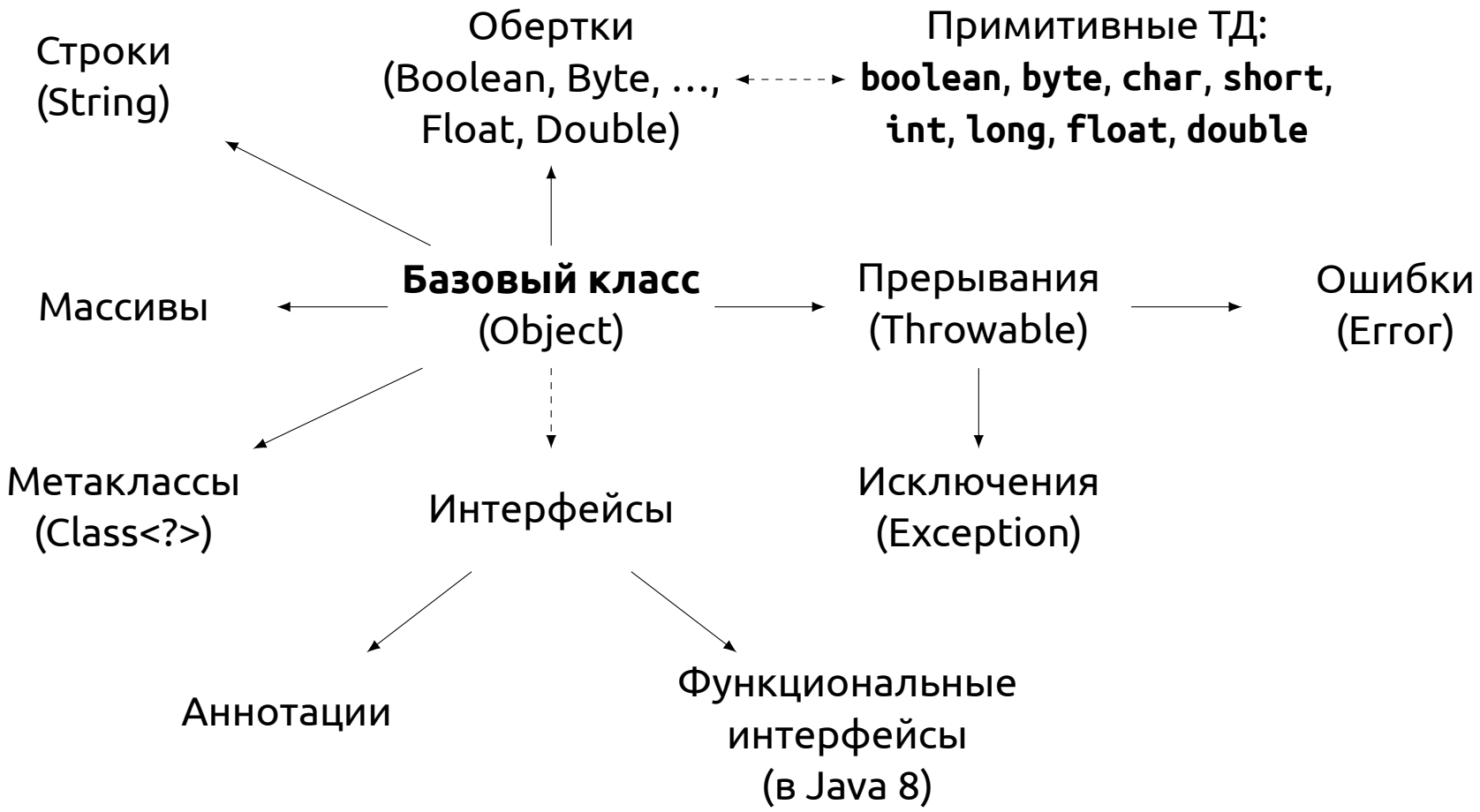
- Дерево:

```
type Tree = record (
  label: String,
  branches: set of (Tree)
)
```

- Интерфейс процедуры для вычисления чисел Фибоначчи:

```
type Fib = procedure ( in n: Int selecting (0 .. *) ) returns ( integer )
```

Иерархия типов данных (Java)



В C / C++, Java, PHP примитивные типы объектов изолированы (не являются потомками других типов).

Боксинг типов данных

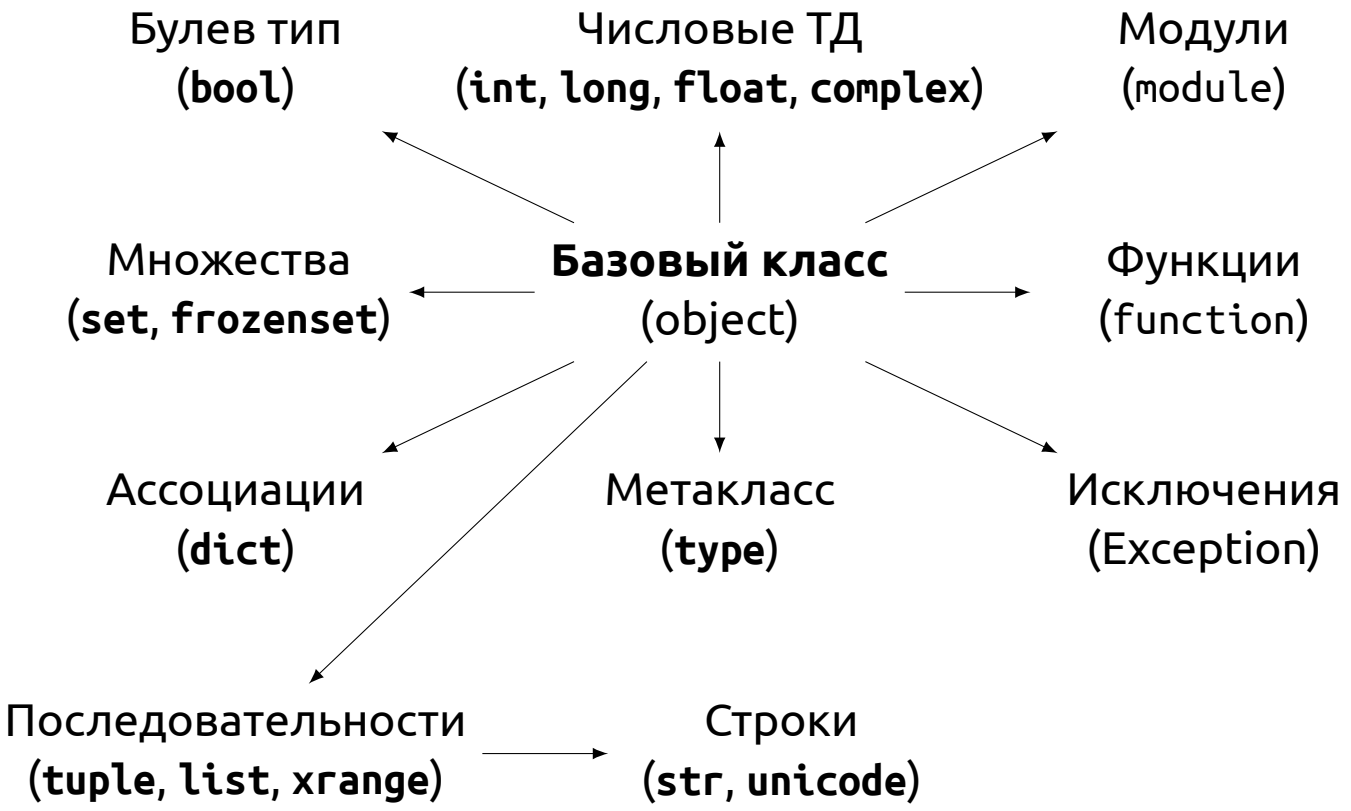
Определение

Боксинг (англ. *boxing*) — преобразование примитивного ТД к соответствующему ссылочному типу данных и обратно.

Пример:

```
1  // В Java примитивному типу int соответствует «сложный» тип Integer,
2  // наследуемый от Object
3  Set<Integer> set = new HashSet<Integer>();
4  // Set<int> было бы некорректным определением типа (в отличие от C#)
5
6  // значения типа int автоматически преобразуются к Integer
7  set.put(5);
8  int x = -15; set.put(x);
9
10 // Обратное преобразование Integer->int
11 // (может привести к ошибке, если в коллекции содержится элемент null)
12 for (int i : set) { /* ...*/ }
```

Иерархия типов данных (Python)



Python — ЯП с унифицированной системой типов: все ТД наследуются от базового.

Иерархия типов данных (Python)

```python
import math
class Foo(object): pass

# В Python все переменные — потомки object:
if (isinstance(5, object) # целые числа
    and isinstance(2.71828, object) # вещественные числа
    and isinstance(True, object) # булевы переменные
    and isinstance(None, object) # нулевой указатель
    and isinstance('str', object) # строки
    and isinstance([2, '3'], object) # последовательности
    and isinstance({'foo':'bar'}, object) # ассоциативные таблицы
    and isinstance(map, object) # функции
    and isinstance(Foo(), object) # объекты
    and isinstance(Foo, object) # классы
    and isinstance(math, object) # модули
):
    print 'OK' # OK
```

Интерфейс базового типа данных

Методы класса Object (Java):

```java
public class Object {
    public Object(); // конструктор
    protected void finalize(); // деструктор
    protected Object clone(); // операция клонирования
    public boolean equals(Object obj); // сравнение по значению
    public int hashCode(); // хэш-код для имплементаций таблиц и множеств
    public String toString(); // приведение к строковому ТД
    public Class<?> getClass(); // класс объекта (напр., для рефлексии)

    // примитивы синхронизации
    public void notify();
    public void notifyAll();
    public void wait();
    public void wait(long timeout);
    public void wait(long timeout, int nanos);
}
```

Интерфейс типов данных в ООП

Спецификация интерфейса объектов:

- Методы.
- Свойства (синтаксический сахар для методов).
- События (свойства процедурного типа).
- Перегрузка операций помимо сравнения (напр., арифметических):
  - **operator** в C++;
  - методы `__lt__`, `__le__` и т. д. в Python.
- Приведение к другим ТД помимо строкового:
  - **operator** `<Type>()` в C++;
  - методы `__int__`, `__float__` и т. д. в Python.
- Манипуляции с содержимым объекта, напр., методы для изменения логики доступа к полям:
  - методы `__getattr__`, `__setattr__`, `__delattr__` в Python;
  - методы `__get` и `__set` в PHP.

Выводы

1. Система типов данных — основной инструмент спецификации интерфейсов в языках программирования. Использование типов данных позволяет повысить понимание программы и служит средством верификации программы.
2. Стандарт ISO/IEC 11404 определяет классификацию типов данных и базовые типы, независимые от языка программирования. В этом стандарте также даны способы построения новых типов данных на основе базовых.
3. ЯП имплементируют систему типов данных в различном объеме и с различными средствами для ее расширения. В объектно-ориентированных ЯП система типов данных представляет собой иерархию с одним или несколькими корневыми классами.

Материалы

- Лавріщева К. М.

  Програмна інженерія (підручник).

  K., 2008. — 319 с.

- ISO / IEC

  General Purpose Datatypes.

  http://standards.iso.org/ittf/PubliclyAvailableStandards/c039479_ISO_IEC_11404_2007(E).zip

- Документация Python

  Data Model.

  https://docs.python.org/2/reference/datamodel.html

Спасибо за внимание!

Приложение. Ссылочные типы данных

Определение

Ссылочный тип (англ. *reference type*) — тип данных, ссылающийся на область памяти (∼указатель с автоматическим разыменованием).

Типы, содержащие значение (англ. *value type*)	**Ссылочные типы**
присвоение — копирование данных	присвоение — копирование ссылки
сравнение по значению	сравнение по ссылке или значению
размещаются в стеке	размещаются в куче

ЯП	**Типы, содержащие значение**	**Ссылочные типы**
Java	примитивные типы	все остальные
C#	примитивные, структуры (**`struct`**)	классы (**`class`**)
Python	числовые, булев (размещаются в куче)	все остальные
Функциональные	все	–

Приложение. Агрегационные типы данных в ЯП

Тип	Java	Python
Массив	`T[]` (одномерный), `T[]…[]` (многомерный)	Есть в модулях `array` и `numpy`
Строка	`String`	`str`, `unicode`
Последовательность	`List<T>`	`list` (изменяемая), `tuple` (неизменяемая)
Множество	`Set<T>`	`set` (изменяемое), `frozenset` (неизменяемое)
Ассоциативная таблица	`Map<K, V>`	`dict`

Обозначения: `T` — тип элементов; `K` — тип ключей, `V` — тип значений.

- **Статическая типизация:** тип элементов агрегационного типа указывается с помощью шаблонов (**`template`** в C++; generics в Java, C#).
- **Динамическая типизация:** агрегационные типы могут содержать произвольные элементы.

Приложение. Функции как тип данных

Определение

Наличие **функций первого класса** в ЯП — работа с функциями как с другими типами данных, в частности:

- поддержка передачи функций как параметров и возврата их как результатов функций;
- присваивание функций переменным;
- (опционально) наличие анонимных функций, вложенных функций и замыканий.

ЯП с функциями первого класса: Python, JavaScript, функциональные ЯП (Scheme, Haskell, ML, Scala).

Частичная поддержка функций как типа данных:

- C# (делегаты и анонимные функции);
- Java 8 (функциональные интерфейсы и анонимные классы для замыканий);
- C / C++ (указатели на функции).

Приложение. Работа с функциями в Python

```
1  # именованная функция
2  def square(x): return x*x
3  # анонимная функция (лямбда-выражение)
4  square = lambda x: x*x
5  print square(2) # 4
6  # функция композиции (аргументы и результат — функции)
7  compose = lambda fA, fB: lambda x: fB(fA(x))
8  # возвести в квадрат, затем привести к текстовому виду
9  f = compose(square, str)
10 numbers = [1, 2, 3]
11 # map отображает каждый элемент коллекции с помощью заданной функции f
12 print map(f, numbers) # ['1', '4', '9']
```