Обобщенное программирование

материал взят из книги Александра Степанов "Начала программирования"

11 марта 2026 г.

Сущности

Абстрактная сущность — это предмет, который является вечным и неизменным, тогда как **конкретная сущность** — это предмет, который может обрести свое существование в пространстве и времени или перестать существовать.

Сущности

Атрибут определяет соответствие между конкретной сущностью и абстрактной сущностью — он описывает некоторое свойство, измерение или качество конкретной сущности.

Идентичность

Идентичность, одно из примитивных понятий нашего восприятия действительности, определяет одинаковость предмета, изменяющегося в течение времени.
Атрибуты конкретной сущности могут изменяться, не затрагивая его идентичность.

Сущности

Синий цвет и число 13 — это примеры абстрактных сущностей. Сократ и Соединные Штаты Америки – примеры конкретных сущностей.

Сущности

Цвет глаз Сократа и число американских штатов — это примеры атрибутов.

Вид

Абстрактный вид описывает общие свойства эквивалентных абстрактных сущностей. Примеры абстрактных видов — натуральное число и цвет.

Вид

Конкретные виды описывают множество атрибутов почти полностью эквивалентных конкретных сущностей. Примеры конкретных видов — человек и штат США.

Функция

Функция — это правило, которое связывает одну или несколько абстрактных сущностей, называемых аргументами, из соответствующего вида с абстрактной сущностью, называемой результатом, из другого вида.

Функция

Примерами функций являются функция определения преемника, которая связывает каждое натуральное число с непосредственно следующим за ним, и функция, которая связывает с двумя цветами результат их смешивания.

Род

Абстрактный род описывает различные абстрактные виды, которые являются подобными в некотором отношении. Примеры абстрактных родов — число и бинарный оператор.

Род

Конкретный род описывает различные конкретные виды, подобные в некотором отношении. Примеры конкретных родов — млекопитающее и двуногое.

Род

Сущность принадлежит к одному и только одному виду, который регламентирует правила для ее построения или существования. Сущность может принадлежать к нескольким родам, каждый из которых описывает определенные свойства.

Значения

Если мы не знаем интерпретацию, то единственными предметами, которые мы видим в компьютере, являются нули и единицы. **Элемент данных** представляет собой конечную последовательность нулей и единиц.

Значения

Тип значения определяет соответствие между видом (абстрактным или конкретным) и множеством элементов данных.

Значения

Элемент данных, соответствующий определенной сущности, называется **представлением сущности**; сущность называется **интерпретацией элемента данных**.

Значения

Мы называем элемент данных вместе с его интерпретацией значением.

Значения

Примерами значений являются целые числа, представленные в формате 32-битовых двоичных чисел в дополнительном коде с обратным порядком байтов.

Значения

Примером значения являются рациональные числа, представленные как конкатенация двух 32-битовых последовательностей, интерпретируемых как целочисленный числитель и знаменатель, которые представлены как значения двоичных чисел в дополнительном коде с обратным порядком байтов.

Значения

Тип значений является **собственно частичным**, если его значения представляют собственное подмножество абстрактной сущности в соответствующем виде; в противном случае он является **полным**. Например, тип int — собственно частичный, тогда как тип bool — полный.

Значения

Тип значений является **уникально представленным**, если и только если каждой абстрактной сущности соответствует не больше чем одно значение.

Значения

Например, тип, представляющий истинностное значение как байт, который интерпретирует нулевое значение как ложь и отличное от нуля значение как истину, не является уникально представленным.

Значения

Тип, представляющий целое число как двоичное число в дополнительном коде, является уникально представленным.

Значения

Тип значений является неоднозначным, если и только если значение типа имеет больше чем одну интерпретацию.

Значения

Два значения из типа значений являются равными, если и только если они представляют одну и ту же абстрактную сущность.

Значения

Они являются **представительно равными**, если и только если их элементы данных представляют собой идентичные последовательности нулей и единиц.

Лемма 1.1

Если тип значений является уникально представленным, то из равенства следует представительное равенство.

Лемма 1.2

Если тип значений не является неоднозначным, то из представительного равенства следует равенство.

Равенство

Если тип значений уникально представлен, мы осуществляем проверку на равенство, проверяя, не являются ли обе последовательности нулей и единиц одинаковыми.
В противном случае мы должны осуществлять проверку на равенство таким способом, который сохраняет его согласованность с интерпретациями его аргументов.

Равенство

Неуникальные представления выбираются, если проверка на равенство осуществляется менее часто, чем операции, вырабатывающие новые значения, и если есть возможность ускорить выработку новых значений за счет замедления проверки на равенство.

Равенство

Например, два рациональных числа, представленные как пары целых чисел, равны, если они приводятся к одному к тому же несократимому виду.

Равенство

Два конечных множества, представленные как неотсортированные последовательности, равны, если после сортировки и устранения дубликатов равны их соответствующие элементы.

Функции

Компьютеры реализуют функции на абстрактных сущностях как функции на значениях.

Функции

Функция, определенная применительно к данному типу значений, является **регулярной**, если и только если она соблюдает проверку на равенство: подстановка равного значения для аргумента дает равный результат.

Функции

Примером числовой функции, не являющейся регулярной, служит функция, которая возвращает числитель рационального числа, представленного как пара целых чисел, поскольку

$$\frac{1}{2} = \frac{2}{4},$$

но

$$\mathit{numerator}\left(\frac{1}{2}\right) \neq \mathit{numerator}\left(\frac{2}{4}\right)$$

Функции

Регулярные функции позволяют провести рассуждение, основанное на равенствах: подстановку равных вместо равных.

Объект

Память — это множество слов, каждое из которых имеет адрес и содержание.

Объект

Адрес — это значение фиксированного размера, называемого длиной адреса.

Объект

Содержание — это значение другого фиксированного размера, называемого длиной слова.

Объект

Получение содержания по адресу осуществляется в операции загрузки.

Объект

Изменение связи содержания с адресом осуществляется в операции сохранения.

Объект

Примерами реализации памяти являются байты в оперативной памяти и блоки на диске.

Объект

Объект — это представление конкретной сущности как значения в памяти.

Объект

Объект имеет **состояние**, которое представляет собой значение, относящееся к некоторому типу значений.

Объект

Тип объектов — это шаблон для хранения и изменения значений в памяти. Каждому типу объектов соответствует тип значений, описывающий состояния объектов этого типа.

Объект

Примером типа объектов могут служить целые числа, представленные в формате 32-битового двоичного числа в дополнительном коде с обратным порядком байтов (little-endian), выровненные к 4-байтовой границе адреса.

Объект

Значения являются неизменными и независимыми от любой конкретной реализации в компьютере.
Объекты являются изменяемыми и имеют реализации, зависимые от компьютера.

Объект

Состояние объекта в любой точке во времени может быть описано значением, которое можно в принципе записать на бумаге (создание моментального снимка) или сериализовать (преобразовать в последовательную форму) и отправить по линии связи.

Объект

Функциональное программирование имеет дело со значениями, а императивное — с объектами.

Объект

Мы используем значения для представления сущностей. Так как значения являются неизменными, они могут представить абстрактную сущность. Последовательности значений могут также представлять последовательности моментальных снимков конкретных сущностей.

Объект

Так как объекты являются изменяемыми, они могут представлять конкретные сущности, принимая новое значение для представления изменения в сущности.

Объект

Объекты могут также представить абстрактную сущность, оставаясь постоянными или принимая различные приближения к абстрактному.

Причины использования объектов

- Объекты моделируют изменяемые конкретные сущности, такие как карточки служащих в приложении учета заработной платы.
- Объекты обеспечивают удобный способ реализации функции на значениях, таких как процедура, извлекающая квадратный корень числа с плавающей запятой с использованием итеративного алгоритма.
- Компьютеры с памятью составляют единственную доступную реализацию универсального вычислительного устройства.

Объект

Некоторые свойства типов значений переносятся на типы объектов.

Объект

Объект является **полностью сформированным**, если и только если полностью сформировано его состояние.

Объект

Тип объектов является **собственно частичным**, если и только если его тип значений собственно частичен; в противном случае он является **полным**.

Объект

Тип объектов является **уникально представленным**, если и только если уникально представлен его тип значений.

Объект

Так как конкретные сущности имеют идентичности, представляющие их объекты нуждаются в соответствующем понятии идентичности.

Объект

Маркер идентичности(identity token) — это уникальное значение, выражающее идентичность объекта, которое вычисляется из значения объекта и адреса его ресурсов.

Объект

Примерами маркеров идентичности могут служит адрес объекта, индекс в массиве, в котором хранится объект, и номер служащего в картотеке персонала.

Объект

Проверка маркеров идентичности на равенство соответствует проверке на идентичность.

Объект

В течение времени существования приложения каждый конкретный объект может использовать различные маркеры идентичности в ходе переме щения либо в пределах структуры данных, либо из одной структуры данных в другую.

Объект

Два объекта одного и того же типа равны, если и только если равны их состояния. Если два объекта равны, мы говорим, что каждый из них является копией другого. Внесение изменения в объект не затрагивает ни одной его копии

Процедура

Процедура — это последовательность команд, которая изменяет состояние некоторых объектов; процедура может также создавать или уничтожать объекты.

Объект с которыми взаимодействует процедура

- Объекты **входные/выходные** состоят из объектов, передаваемых в процедуру или из процедуры прямо или косвенно с применением ее аргументов или возвращаемого результата.
- Объекты **локального состояния** состоят из объектов, создаваемых, уничтожаемых и, как правило, изменяемых в течение одного вызова процедуры.
- Объекты **глобального состояния** состоят из объектов, доступных для этой и других процедур на протяжении многочисленных вызовов.
- Объекты **собственного состояния** состоят из объектов, доступных только для данной процедуры (и присоединенных к ней процедур), но предоставляемых для общего доступа на протяжении многочисленных вызовов.

Процедура

Объект передан **прямо**, если он передан как аргумент или возвращен как результат, и передан **косвенно**, если он передан через указатель или объект, подобный указателю.

Процедура

Объект является **входным** объектом процедуры, если происходит его чтение, но не изменение процедурой.

Процедура

Объект является **выходным** объектом процедуры, если происходит его запись, создание или уничтожение процедурой, но не чтение его начального состояния процедурой.

Процедура

Объект является **входным/выходным** объектом процедуры, если происходит не только его изменение, но и чтение процедурой.

Процедура

Вычислительный базис для типа — это конечное множество процедур, которые обеспечивают построение любой другой необходимой процедуры применительно к данному типу.

Процедура

Базис является **эффективным**, если и только если любая процедура, реализованная с его использованием, оказывается такой же эффективной, как и эквивалентная процедура, написанная на основе альтернативного базиса.

Процедура

Например, базис для k-битовых целых чисел без знака, который предоставляет только значение нуль, процедуру проверки на равенство и функцию определения преемника, не эффективен, поскольку сложность реализации операции сложения на основе определения последующего элемента является экспоненциальной в k.

Процедура

Базис является выразительным, если и только если он позволяет создавать компактные и удобные определения процедур применительно к данному типу.

Процедура

Например, операция вычитания может быть реализована с использованием операций перемены знака и сложения, но должна быть включена в выразительный базис. Аналогично перемена знака может быть реализована с использованием операции вычитания и значения нуль, но должна быть включена в выразительный базис.

Регулярные типы

Может быть предложено множество процедур, включение которых в вычислительный базис типа позволит нам помещать объекты в структуры данных и использовать алгоритмы для копирования объектов из одной структуры данных в другую. Мы называем типы, имеющие такой базис, **регулярными**, так как их использование гарантирует регулярность поведения и благодаря.

Регулярный тип

Тип является регулярным, если и только если его базис включает процедуру проверки на равенство, процедуру присваивания, деструктор, стандартный конструктор, конструктор копии, процедуру полного упорядочения и основополагающий тип.

Регулярные типы

Равенство — это процедура, которая принимает два объекта одного и того же типа и возвращает значение true, если и только если состояния объектов равны. **Неравенство** определено тогда же, когда и равенство, и возвращает значение, противоположное равенству.

```
a == b
a != b
```

Регулярные типы

Присваивание — это процедура, которая принимает два объекта одного и того же типа и делает первый объект равным второму, не изменяя второй. Смысл присваивания не зависит от начального значения первого объекта.

$$a = b$$

Деструктор

Деструктор — это процедура, вызывающая прекращение существования объекта. После того как по отношению к объекту вызван деструктор, к этому объекту больше не может быть применена ни одна процедура, а его прежние местоположения в памяти и ресурсы могут быть повторно использованы в других целях.

Деструктор

Глобальные объекты уничтожаются после завершения работы приложения, локальные объекты – после выхода из блока, в котором они объявлены, а элементы структуры данных – после уничтожения структуры данных.

Конструктор

Конструктор — это процедура, преобразовывающая местоположения в памяти в объект. Возможные варианты поведения находятся в пределах от невыполнения никаких действий до установления состояния сложного объекта.

Конструктор

Объект находится в **частично сформированном** состоянии, если к нему можно применить процедуру присваивания или уничтожения. Для объекта, который частично, но не полностью сформирован, результат любой процедуры, кроме присваивания (только на левой стороне) и уничтожения, не определен.

Конструктор

Полностью сформированный объект является частично сформированным.

Конструктор

Стандартный конструктор не принимает аргументы и оставляет объект в частично сформированном состоянии.

```
T a;
T()
```

Конструктор копии

Конструктор копии принимает дополнительный аргумент того же типа и создает новый, равный ему объект.

```
T a = b;
```