Класс как основной механизм абстракции

Наумов Д.А., доц. каф. КТ

Основы программной инженерии, 2019

Содержание лекции

1. Структура и организация определения класса
   - Элементы-данные и элементы-действия
   - Управление доступом к членам класса
   - Спецификация и реализация класса
2. Конструкторы и деструкторы
   - Перегрузка конструкторов
   - Права доступа в связи с конструкторами
   - Конструкторы как преобразования типов
   - Деструкторы
3. Статические данные

Задача классов - предоставить программисту инструмент для создания новых типов данных с тем, чтобы их можно было без ограничений использовать в программе наряду со встроенными типами.

Тип данных является вопложением некоторой концепции (пример: целые числа), которое определяется рядом характеристик:

- областью применения типа;
- способом представления типа в памяти;
- множеством операций, разрешенных для этого типа;
- множеством совместимых типов.

Новые типы создаются для воплощения концепций, которые *не выражаются непосредственно* (или адекватно) встроенными типами.

- **Класс** - элемент абстракции.
- Определяя класс, мы создаем программную сущность, позволяющую выделить существенные характеристики некоторого объекта, отличающие его от других видов объектов.
- Мы отделяем друг от друга элементы объекта, определяющие его устройство (**элементы-данные**), от элементов, определяющих его поведение (**элементы-действия**).
- Данные, объявленные внутри определения класса, могут, в частности, быть п**еременными-членами класса** и **константами-членами класса**.
- Функции, объявленные внутри определения класса, называются **функциями-членами класса**, или **методами** класса.

Пример. Очередь

```
1 type
2   QueueInt = class(TObject)
3     public
        //массив для хранения данных
4       body: array[0..MAXSIZE] of integer;
5       size: integer;   //Предельный размер
6       length: integer; //Текущая длина очереди
7       last: integer;   //Индекс последнего
                         //занесенного элемента
8       next: integer;   //Индекс извлекаемого
                         //Элемента очереди
9       procedure Init(CustomSize: integer); //Инициализация
10      procedure Enqueue(Elem: integer); //Записать элемент
11      function Dequeue(): integer; //Извлечь элемент
12      procedure Print; //Печать содержимого
13 end;
```

Разграничение доступа к членам класса преследует две основные цели:

- обеспечение связывания функций-членов класса с переменными-членами класса таким образом, чтобы только эти функции непосредственно зависели от представления класса и имели непосредственный доступ к объектам класса;
- обеспечение инкапсуляции, т. е. отделения внешнего интерфейса класса от его реализации.

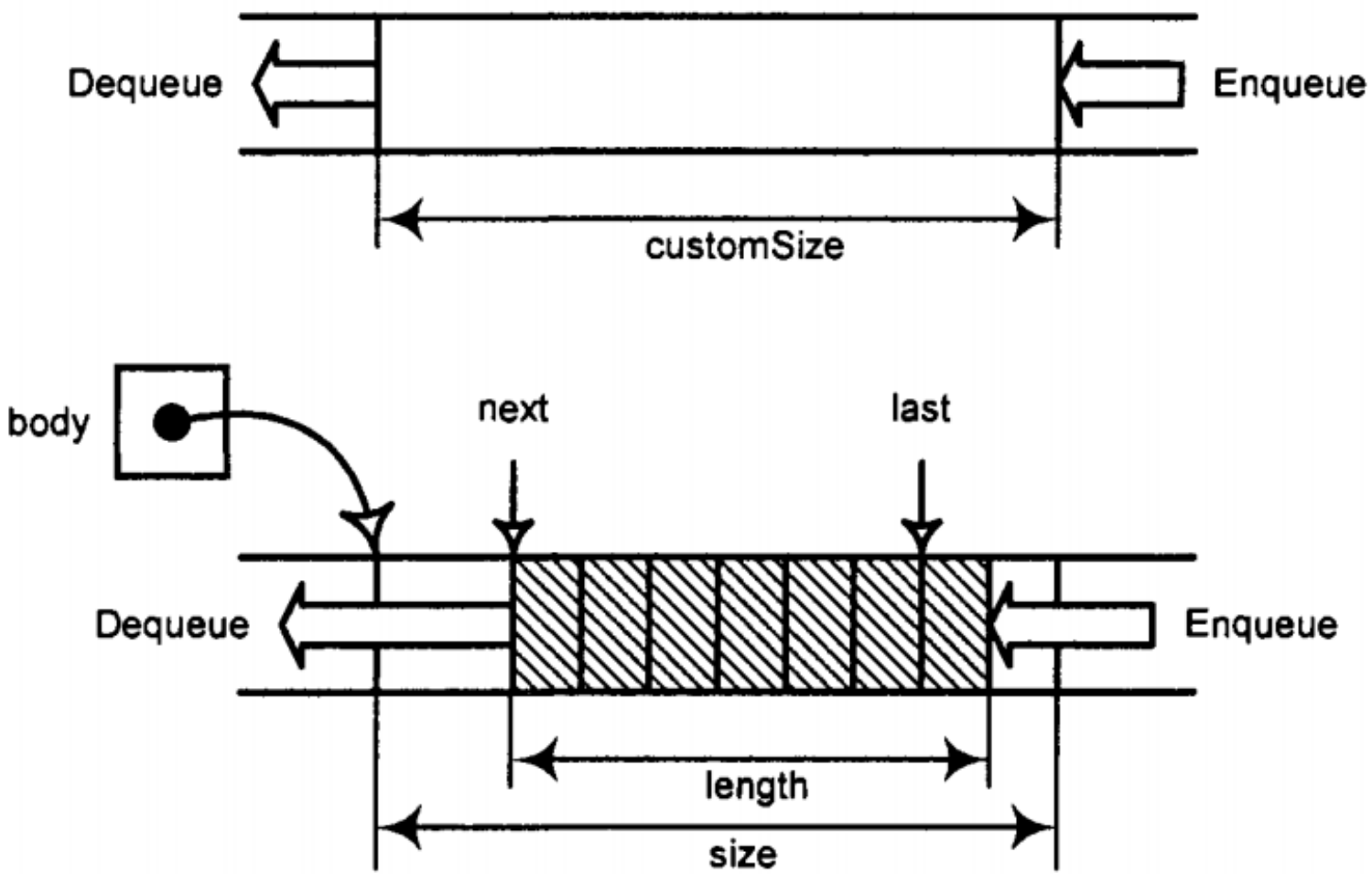
Спецификаторы доступа:

- спецификатор **private** - закрытые члены класса, могут использоваться только методами класса;
- спецификатор **public** - открытый интерфейс класса, разрешен доступ не только функциям-членам класса, но и обращение из-за пределов объявления класса;
- **protected** - будет рассмотрен позже;
- **published** - будет рассмотрен позже.

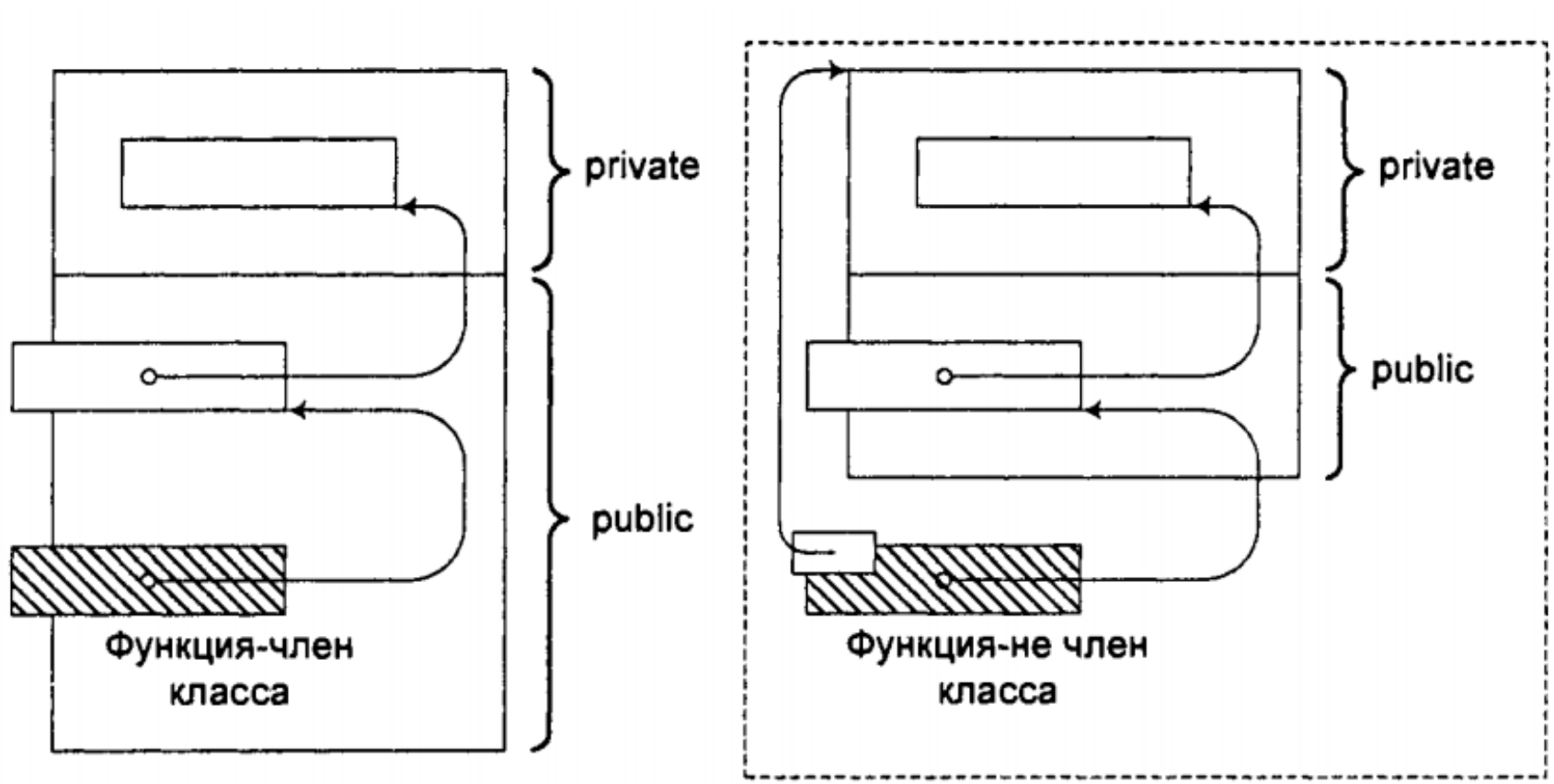
Пример. Инкапсуляция

```
1 type
2   QueueInt = class(TObject)
3     private
        //массив для хранения данных
4       body: array[0..MAXSIZE] of integer;
5       size: integer;   //Предельный размер
6       length: integer; //Текущая длина очереди
7       last: integer;   //Индекесы первого и
8       next: integer;   //последнего элементов
      public
9       procedure Init(CustomSize: integer); //Инициализация
10      procedure Enqueue(Elem: integer); //Записать элемент
11      function Dequeue(): integer; //Извлечь элемент
12      procedure Print; //Печать содержимого
13 end;
```

Различия абстрактных моделей пользователя и разработчика



Функции - не члены класса, улучшающие инкапсуляцию



Инициализация объектов

Инициализация с использованием специальной функции:

```
//Инициализация очереди
procedure QueueInt.Init(CustomSize: integer);
begin
  if (CustomSize < 1) or (CustomSize > MAXSIZE) then
  begin
    writeln('Incorrect size. Using default value');
    size := MAXSIZE;
  end
  else
    size := CustomSize;
end;
```

- нигде в объявлении типа не указано, что объект должен быть проинициализирован;
- некоторые объекты могут не нуждаться в инициализации, а для некоторых инициализация должна быть обязательной.

Конструкторы

Конструктор

метод класса, основное назначение которого состоит в инициализации объекта

```
//Инициализация очереди при помощи конструктора
constructor QueueInt.Create(CustomSize: integer);
begin
  if (CustomSize < 1) or (CustomSize > MAXSIZE) then
  begin
    writeln('Incorrect size. Using default value');
    size := MAXSIZE;
  end
  else
    size := CustomSize;
end;
```

- рекомендованное имя конструктора - Create;
- конструктор вызывается всегда при создании объекта.

Конструкторы

```
procedure TestQueue;
var
  q: QueueInt;     //объект, размещенный в стеке
begin
  // Автоматическое размещение объекта в стеке
  // => Вызов конструктора при попадании объектной
  // переменной в область видимости

  // Создание объектаа
  q := QueueInt.Create();

  // Занесение элемента в очередь
  q.Enqueue(50);

  // Вывод очереди
  q.Print;
end;
```

Спецификация класса

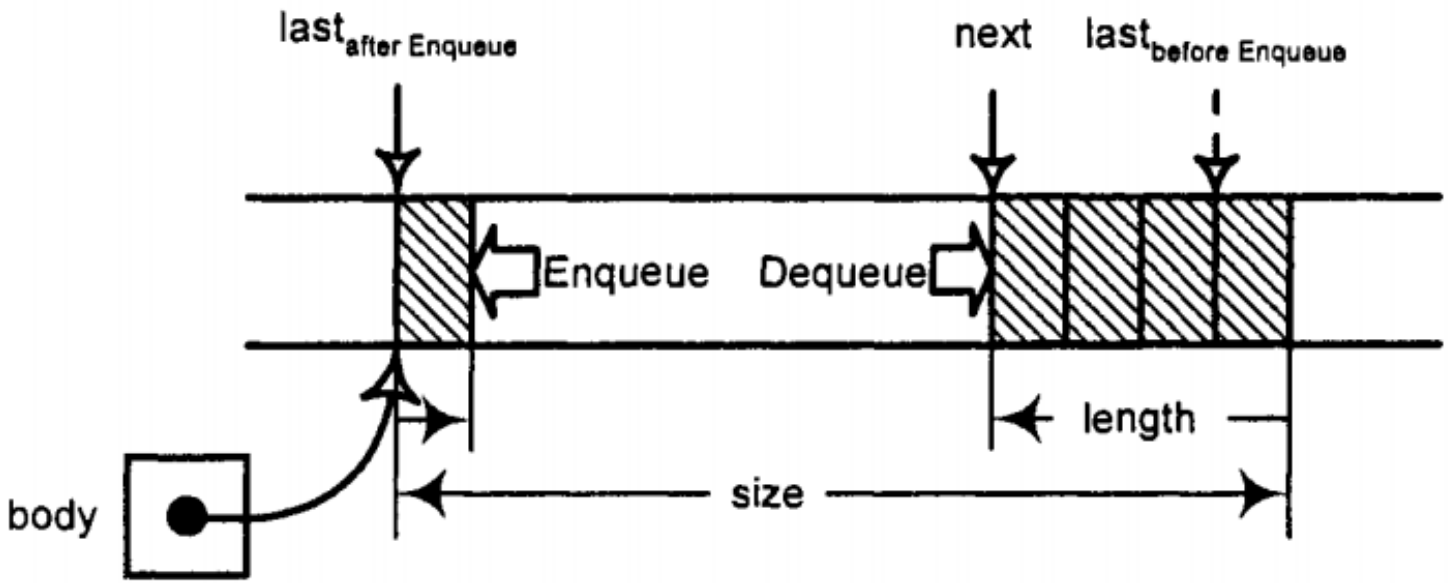
метод класса, основное назначение которого состоит в инициализации объекта

Спецификация:

- содержит информацию, достаточную для корректного использования класса.
- для разработчика класса должна содержать объявления всех членов класса.
- для пользователя должна содержать, по крайней мере, объявления открытых и защищенных членов класса.

```
// class QueueInt
//
// Обеспечивает функциональность для хранения целых чисел
// в стандартной очереди, обрабатываемой по принципу:
// "первым пришел - первым обслужен".
//
// Определен в модуле queue.pas
//
// Конструктор (является конструктором по умолчанию):
//   QueueInt( int customSize = 100 );
//     Инициализирует очередь указанного размера.
//     Если аргумент имеет некорректное значение,
//     печатается диагностическое сообщение
//     и устанавливается значение по умолчанию.
//
//     Аргументы конструктора:
//       customSize - задаваемый предельный размер очереди,
//       Значение по умолчанию: 100
//       Предельное разрешенное значение: 100
//
// Открытые функции-члены класса:
//   procedure Enqueue(item: integer);
//     Обеспечивает занесение элемента в очередь.
//     Если очередь полна, печатает сообщение об ошибке.
//   Аргументы:
//     item - элемент, заносимый в очередь
//     Возвращаемое значение:
//       Нет
//
//   function Dequeue(): integer;
//     Обеспечивает извлечение элемента из очереди.
//     Если очередь пуста, печатает сообщение об ошибке.
//   Аргументы:
//     Нет
//   Возвращаемое значение:
//     Значение извлекаемого элемента
```

Обработка индексов при работе с массивом QueueInt



Недостатки приведенной реализации:

1. В ходе работы методов класса QueueInt происходит печать на консоль. Это ограничивает область применения класса только консольными приложениями.
2. Методы класса реализованы таким образом, что в случае возникновения ошибок производится печать информации об ошибке на консоль, что может не вписываться в организацию клиентского кода.
3. Встроенная обработка ошибок работы с очередью не позволяет пользователю класса реализовать свой обработчик ошибок с учетом инфраструктуры клиентского приложения.
4. В случае, если очередь пуста, функция Dequeue() возвращает фиктивное значение, которое вполне может совпадать с одним из элементов очереди. Это может быть источником ошибок при использовании очереди.
5. Спроектированный класс позволяет хранить в очереди только целые данные.

Пример. Рациональные числа

Класс может иметь несколько конструкторов с различными списками параметров. Это позволяет иметь несколько вариантов инициализации объекта.

```
type
  //класс для работы с рациональными числами
  Ratio = object
    private
      //n - числитель числа
      //d - знаменатель числа
      n, d: integer;

    public
      //конструктор по-умолчанию
      constructor Init;
      //конструктор с двумя параметрами
      constructor Init(InN, InD: integer);
      //конструктор копии
      constructor Init(other: Ratio);

      //ввод числа
      procedure RatioOut(s:string);
      //вывод числа
      procedure RatioIn(s:string);

      //сокращение дроби
      procedure Reduce;
  end;
```

Права доступа в связи с конструкторами

Конструктор может быть объявлен с разным спецификатором доступа.

- Помещение конструктора в защищенную секцию делает возможным обращение к нему только в инициализаторе производного класса.
- Объявление закрытого конструктора запрещает инициализацию объекта определенным образом.
- Если класс не имеет ни одного открытого конструктора, объекты такого класса создавать нельзя. Это означает, что данный класс можно использовать только в качестве базового для других классов.

Конструкторы как преобразования типов

Конструктор, имеющий один аргумент, задает преобразование типа, то есть определяет правило преобразования типа аргумента конструктора в тип объекта.

```
//Преобразования между Double и Ratio
function DtoR(v:double):TRatio;
begin
  Result := TRatio.Create;

  Result.n:=round(v*9e7);

  Result.d:=90000000;

  Result.Reduce;
end;
```

```
//конструктор с одним параметром
constructor TRatio.Create(other: double);
begin
  n := DtoR(other).n;
  d := DtoR(other).d;
end;
```

Конструкторы как преобразования типов

```
var
  x, y, z: TRatio;

begin
  x := TRatio.Create; //вызываем конструктор по-умолчанию

  y := TRatio.Create(2, 10);   //вызываем конструктор с двумя
                               //паарametрами (дробь 2/10)
  x.RatioOut('x=');
  y.RatioOut('y=');

  z := TRatio.Create(1.2);     //вызываем конструктор с одним параметром
                               //для преобразования типа double в TRatio
  z := z + x + y;

  z.RatioOut('z=');

  writeln('value of z is ', RtoD(z));
```

Деструктор

специальный метод класса, обеспечивающий корректное разрушение среды, в которой функционирует объект.

- Иногда создание объекта предполагает помимо прочего захват каких-либо ресурсов (например, резервирование динамической памяти).
- Для таких классов необходимо наличие метода, который будет гарантированно вызываться по окончании использования объекта.

В подавляющем большинстве случаев деструкторы вызываются неявно.

```
destructor TRatio.Destroy;
begin
  DecCounter();

  writeln('ratio has been destroyed');
end;
```

Статическими члены класса

переменных-члены класса, которые, являясь элементами класса, не являются элементами объекта, или, иными словами, являются общими для всех объектов данного класса.

```
public
  //статическое поле класса для подсчета числа экземпляров
  counter: integer; static;

  //конструктор класса
  class constructor Create;
  //деструктор класса
  class destructor Destroy;

  class function GetCounter():integer; static;
  class procedure IncCounter(); static;
  class procedure DecCounter(); static;
```

Статические члены класса позволяют, с одной стороны, обеспечить требуемую в подобных случаях независимость от состояния и поведения отдельного объекта, с другой стороны — обеспечить их существование в области видимости класса.

- К статическим членам класса можно обращаться без указания имени объекта.
- В качестве квалификатора используется имя класса, который и образует область видимости статических данных.
- Статические функции не могут обращаться к нестатическим членам класса.

```
class function TRatio.GetCounter():integer; static;
begin
  Result := counter;
end;
```

```
class procedure TRatio.IncCounter(); static;
begin
  Inc(counter);
end;
```

```
class procedure TRatio.DecCounter(); static;
begin
  Dec(counter);
end;
```

Статические конструктор и деструктор

```
class function GetCounter():integer; static;
class procedure IncCounter(); static;
class procedure DecCounter(); static;
```

- может существовать только один конструктор класса (статический конструктор) без параметров.
- может существовать только один деструктор класса (статический деструктор) без параметров.
- статические конструктор и деструктор вызываются только неявно.
- порядок вызова конструкторов и деструкторов не определен.