Основы объектно-ориентированного программирования

Наумов Д.А., доц. каф. КТ

Основы программной инженерии, 2019

Содержание лекции

Объектно-ориентированное программирование (ООП)

самая распространенная в современной проектной практике парадигма программирования.

- поддерживается большинством современных языков программирования;
- реализации концепций и механизмов ООП могут отличаться в разных языках.

Широкое применение технологии = технология позволяет успешно решать актуальные проблемы проектирования.

Прогресс информационных технологий

- качество проектирования <- квалификация и способности проектировщиков;
- технология проектирования помогает построить качественный продукт;
- необходимо изучать не языки программирования сами по себе, а концепции, выражаемые этими языками.

Язык программирования

модель (виртуальная вычислительная машина), позволяющая наиболее эффективно использовать возможности вычислительных средств, существенные для конкретных областей применения.

- при написании программ ориентируются не на вычислительную машину как таковую, а на некоторую абстрактную модель вычислительного устройства;
- качество модели определяет качество и эффективность проектирования;
- сложность задач, которые возможно решить, непосредственно связана с уровнем абстракции как при постановке задачи, так и в ходе ее решения.

Развитие языков программирования

это, прежде всего, развитие абстрактных моделей, облегчающих и систематизирующих проектирование.

Методология структурного императивного программирования

воплощает подход, характеризующийся принципом последовательного изменения состояния вычислителя пошаговым образом с поддержкой концепции структурного программирования.

- ориантируется на класс архитектур фон Неймана;
- примеры языков: Fortran, Pascal, C, PL/1.

Развитие структурного программирования:

- функционально-иерархическая декомпозиция;
- структурная организация данных;
- повторное использование проектных решений;
- технология тестирования программного обеспечения.

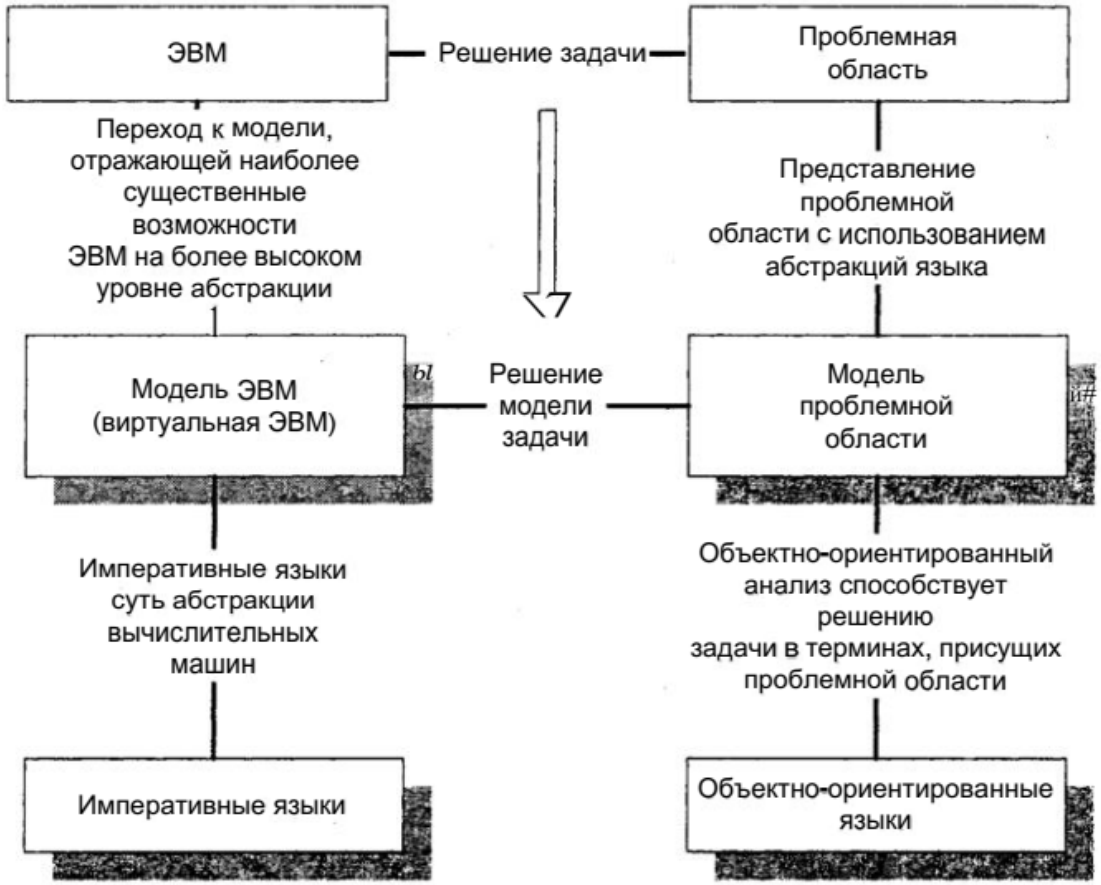
Проблема процедурных языков:

- уровень абстракции все еще требует от программиста мышления в большей мере в терминах вычислительной машины;
- качество проектирования определяется в итоге тем, насколько удачно программисту удалось установить соответствие между пространством понятий, характерных для решаемой задачи, и набором изобразительных средств языка;
- абстрагирование, достигаемое посредством использования процедур, хорошо подходит для описания абстрактных действий, но не предоставляет адекватных языковых средств для описания абстрактных объектов.

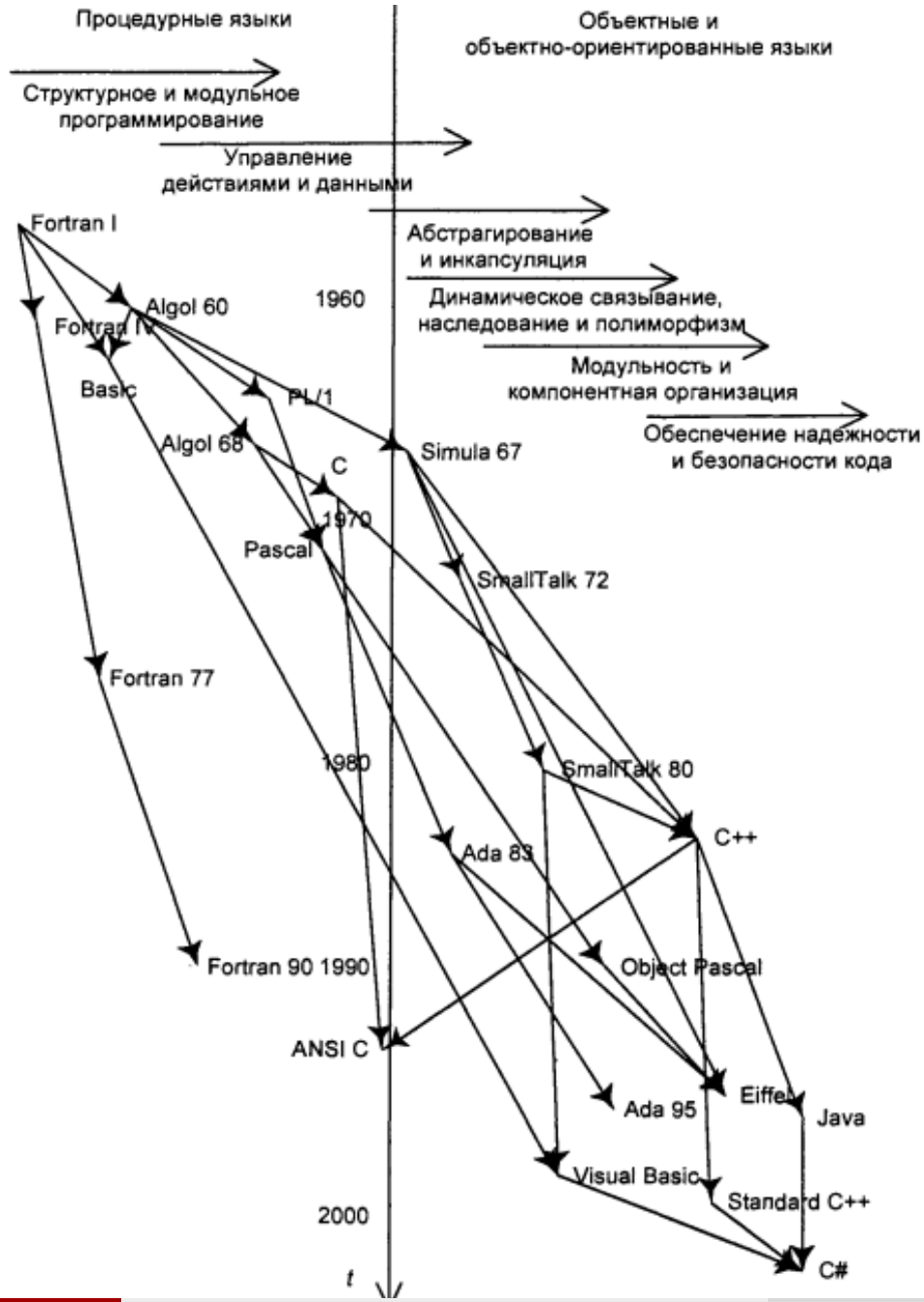
Альтернативные модели:

- функциональное программирование, язык LISP: все задачи, в конечном счете, могут быть сведены к работе со списками;
- логическое программирование, язык Prolog: все задачи могут быть сведены к цепочке логических рассуждений.

ООП предоставляет разработчику инструмент, который позволяет описать задачу и существенную часть реализации проекта в терминах, характеризующих предметную область, а не компьютерную модель.



ООП предоставляет разработчикам гибкий мощный универсальный инструмент, не связанный с каким-то определенным классом задач.


Процедурные языки
Объектные и
объектно-ориентированные языки
Структурное и модульное
программирование
Управление
действиями и данными
Абстрагирование
и инкапсуляция
Динамическое связывание,
наследование и полиморфизм
Модульность и
компонентная организация
Обеспечение надежности
и безопасности кода
Fortran I
Algol 60
Fortran II
Basic
PL/1
Algol 68
Simula 67
C
1960
1970
Pascal
SmallTalk 72
Fortran 77
1980
SmallTalk 80
C++
Ada 83
Object Pascal
Fortran 90 1990
ANSI C
Ada 95
Eiffel
Java
Visual Basic
Standard C++
2000
C#
t

- объектно-ориентированное программирование улучшает проектирование, фокусируясь на данных как более стабильном элементе вычислительной системы;
- объектно-ориентированный подход концентрируется на разработке кода, направленного на повторное использование;
- объектно-ориентированная модель обеспечивает лучшую масштабируемость проектов;
- большинство успешных современных технологий проектирования программных систем предполагает преимущественное или искчючительное использование объектно-ориентированной парадигмы.

Объектную модель составляют четыре главных элемента:

- абстрагирование (abstraction);
- инкапсуляция (encapsulation);
- модульность (modularity);
- иерархия (hierarchy).

Абстрагирование

позволяет выделить существенные характеристики некоторого объекта, отличающие его от всех других видов объектов. Абстракция четко определяет концептуальные границы объекта с точки зрения наблюдателя.

Инкапсуляция

это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция также служит для того, чтобы изолировать внешнее поведение объекта от его внутреннего устройства.

Объектную модель составляют четыре главных элемента:

- абстрагирование (abstraction);
- инкапсуляция (encapsulation);
- модульность (modularity);
- иерархия (hierarchy).

Иерархия

это упорядочение абстракций, средство классификации объектов, систематизация связей между объектами.

Модульность

это представление системы в виде совокупности обособленных сегментов, связь между которыми обеспечивается посредством связей между классами, определяемыми в этих сегментах.

Абстракция

Абстракция

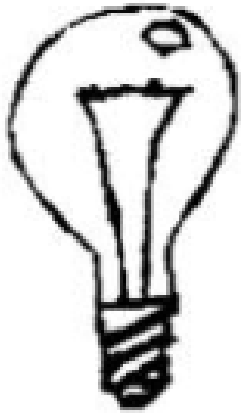
выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.

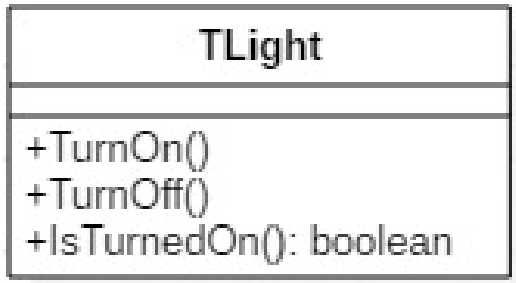
Два вида абстракций в ООП:

- **тип данных объектной природы (класс)**— определяемое программистом расширение исходных типов языка;
- **экземпляр класса (объект)** — переменная класса. Объект обладает состоянием, поведением и идентичностью.

Состояние объекта:

- характеризуется набором его свойств (атрибутов) и текущими значениями каждого из этих свойств;
- результат его поведения






```
type

  TLight = class
    public
      procedure TurnOn();
      procedure TurnOff();
      function IsTurnedOn: boolean;
  end;
```

Описание и создание экземпляра класса:

```
var
  light1: TLight;

begin
  light1 := TLight.Create;

  light1.TurnOn();
  light1.TurnOff();
end.
```

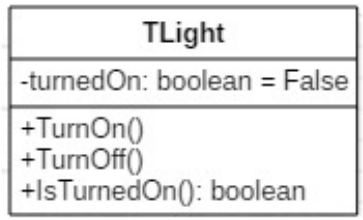
- объектная переменная хранит информацию, характеризующую объект;
- объект размещается в динамической памяти (Create).

Абстрагирование направлено на наблюдаемое поведение объекта (а не на внутреннюю реализацию).

Инкапсуляция

Инкапсуляция

изолирует интерфейс от реализации и связана с управлением доступом к данным класса.




```
TLight = class
  private
    isTurned: boolean;
  public
    procedure TurnOn();
    procedure TurnOff();
    function IsTurnedOn: boolean;
end;
```

Инкапсуляция

Инкапсуляция предполагает возможность ограничения доступа к данным класса из других классов.

- это позволяет упростить интерфейс класса, показав наиболее существенные для внешнего пользователя данные и методы;
- скрытие реализации обеспечивает возможность внесения изменений в реализацию класса без изменения других классов.

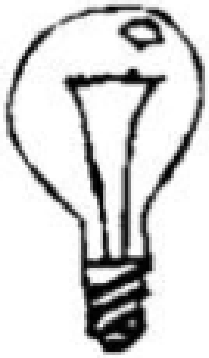
```
procedure TLight.TurnOn();
begin
  isTurned := true;
end;

procedure TLight.TurnOff();
begin
  isTurned := false;
end;

function TLight.IsTurnedOn: boolean;
begin
  IsTurnedOn := true;
end;
```

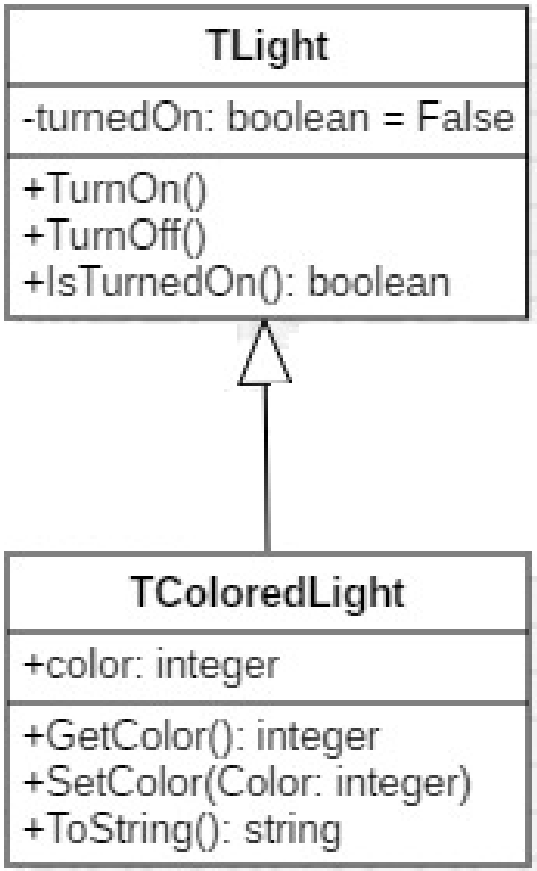
Доступ к закрытым данным возможен из всех классов, расположенных в том же модуле, что и анализируемый класс.

Отношения классов

- **Отношение обобщения** (generalization) описывает отношение между общей сущностью и ее конкретным воплощением (один класс является специализацией другого класса);
- **Отношение ассоциации** (association) описывает структурное отношение, показывающее, что объекты одного типа некоторым образом связаны с объектами другого типа (находятся в отношении типа "часть/целое").
- **Отношение зависимости** (dependency) описывает отношение использования, при котором изменение в спецификации одного класса может повлиять на класс, его использующий (объекты некоторого класса передаются в качестве аргументов функциям-членам другого класса).
- **Отношение реализации** (realization) - семантическое отношение, при котором класс гарантирует выполнение контракта, определяемого некоторым интерфейсом.

Отношение обощения





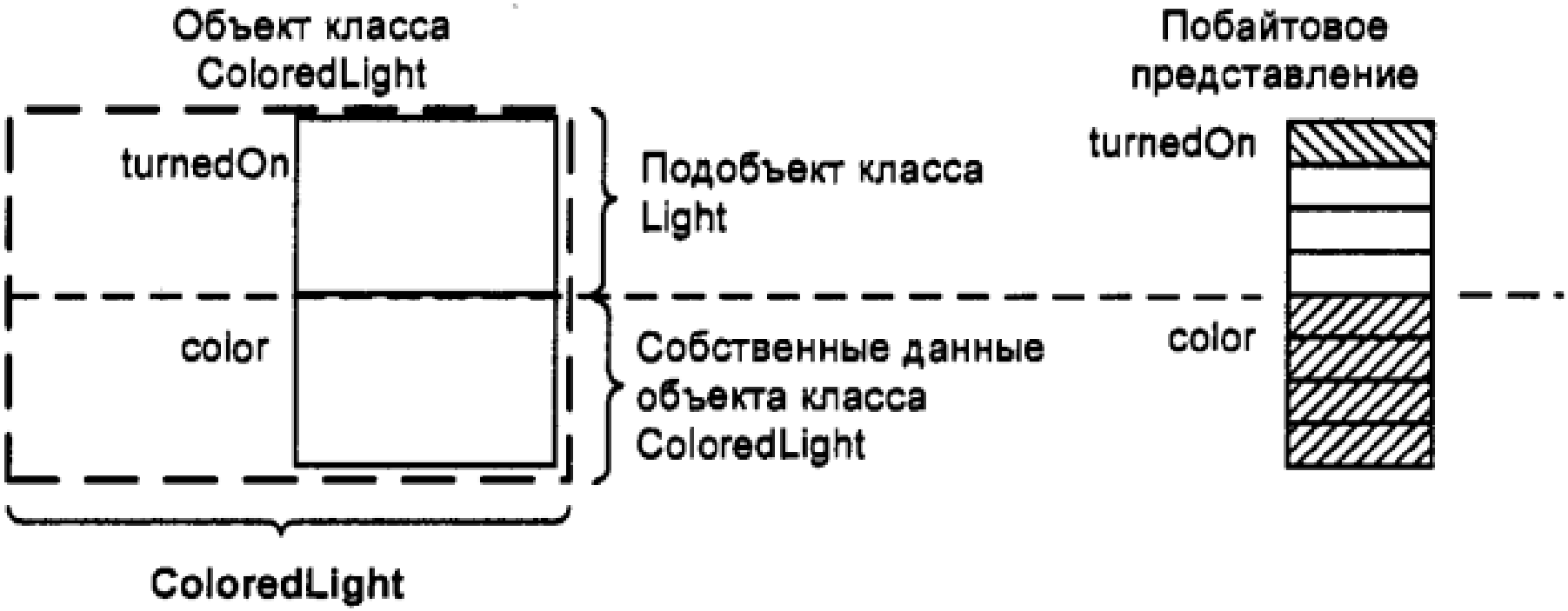
Отношение обощения

```
TColoredLight = class(TLight)
  private
    color: integer;
  public
    procedure SetColor(AColor:integer);
    function GetColor: integer;
    function ToString(): string;
end;
```

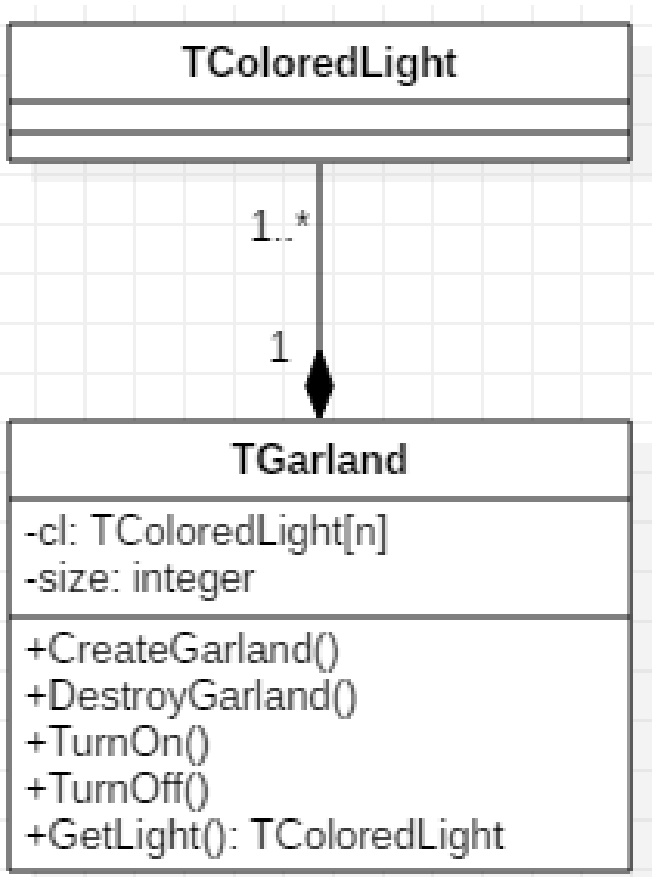
Отношение обощения

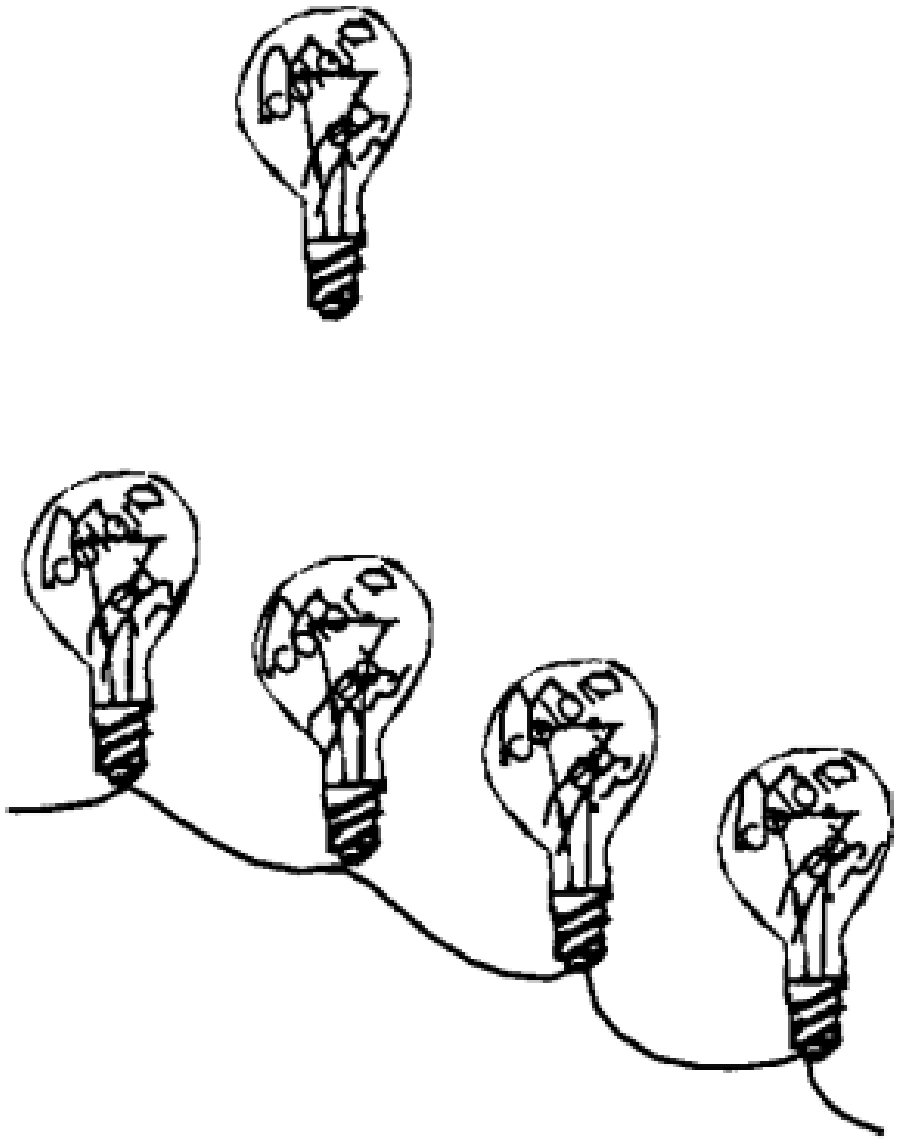
```
begin
  light2 := TColoredLight.Create;

  light2.SetColor( 0 );

  light2.TurnOn();
  light2.TurnOff();
end.
```

Подобъект базового класса



Отношение ассоциации

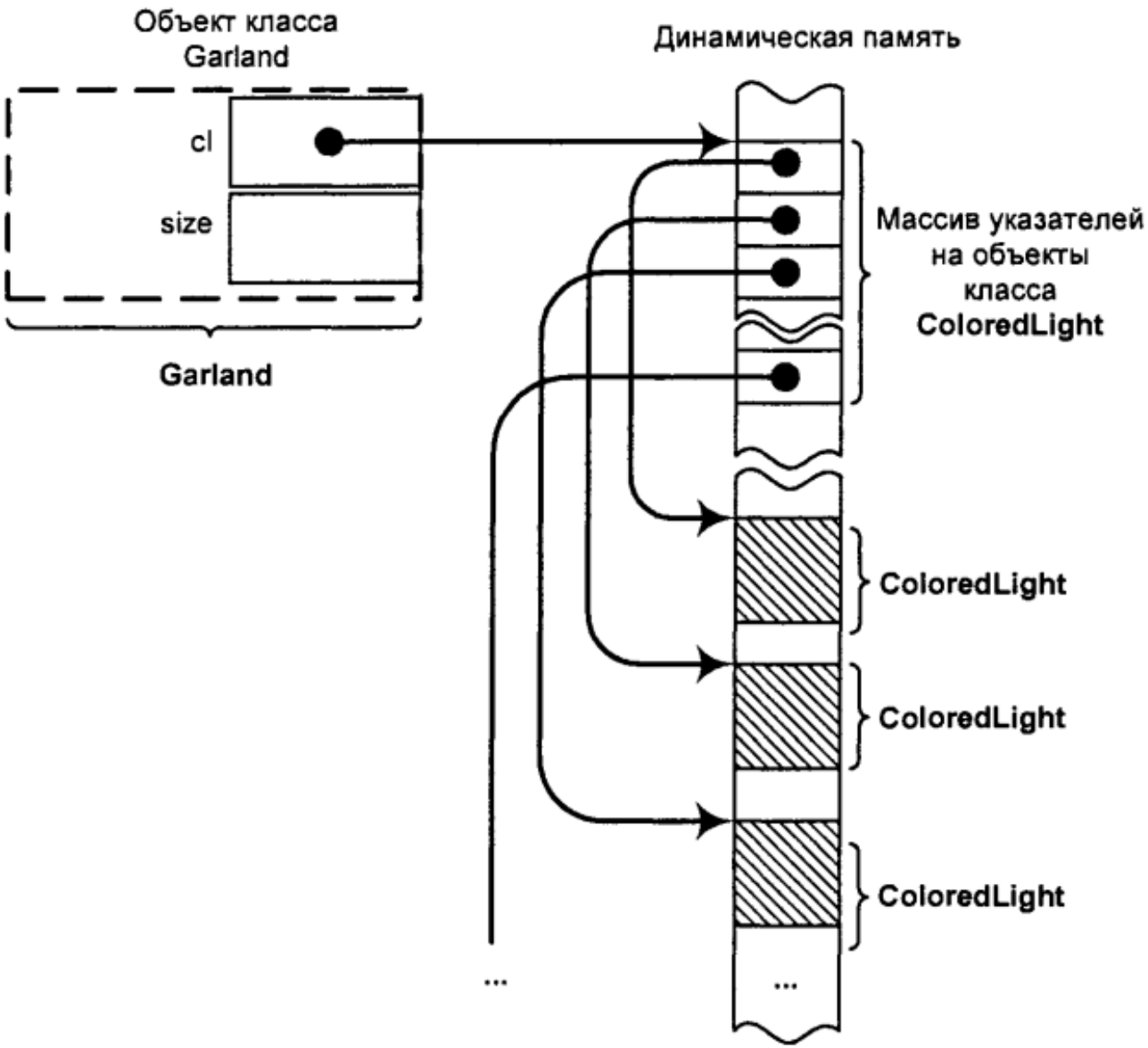




Отношение ассоциации

```
TLightArray = array of TColoredLight;

TGarland = class
  private
    size: integer;
    cl: TLightArray;

    procedure RandomLight(var ALight: TColoredLight);
  public
    procedure Create(ASize: integer);
    procedure Destroy();

    procedure TurnOn();
    procedure TurnOff();

    function GetLight(Index: integer): TColoredLight;
end;
```

Размещение объектов в памяти



Пример

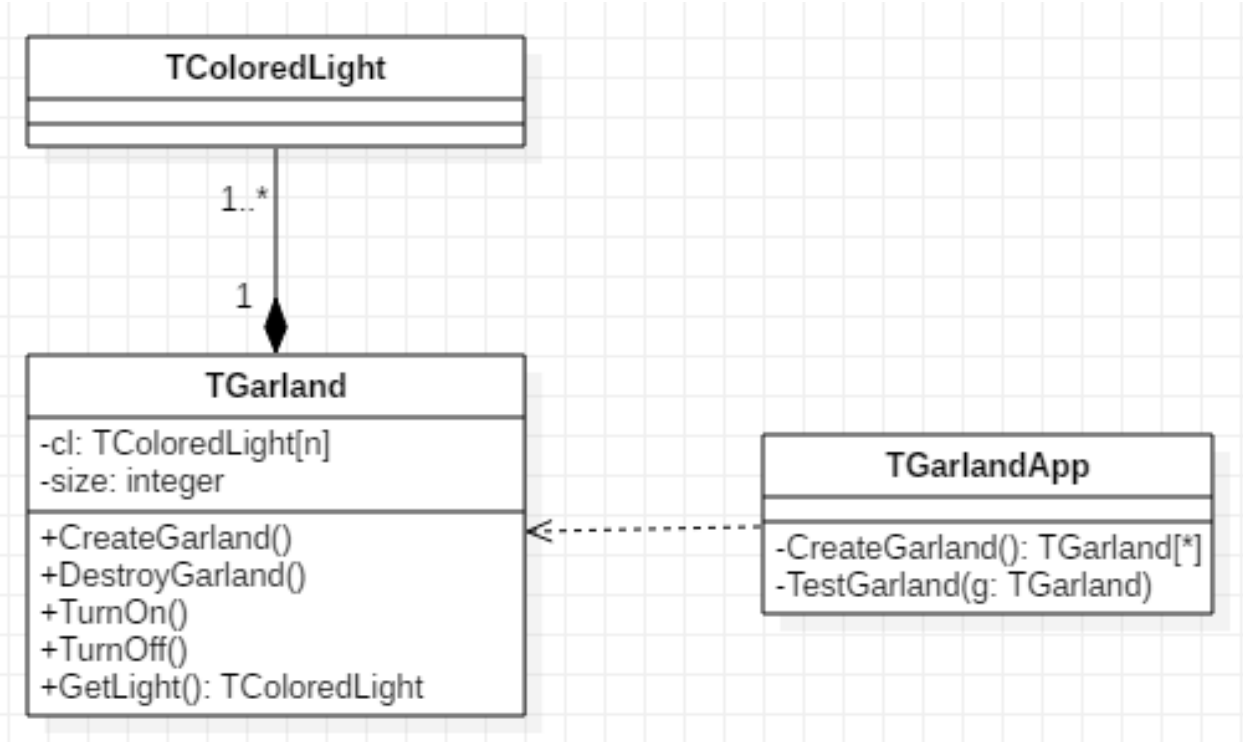
```
var
  g: TGarland;

  light: TColoredLight;

  i: integer;
```

```
begin
  randomize;

  g := TGarland.Create;
  g.CreateGarland(10);

  for i := 0 to g.GetSize() - 1 do
  begin
    light := g.GetLight(i);
    writeln(light.ToString())
  end;
```

Отношение зависимости

Отношение зависимости

является таким типом отношений между классами, когда изменение в спецификации или реализации одного класса влияет на спецификацию или реализацию другого класса.

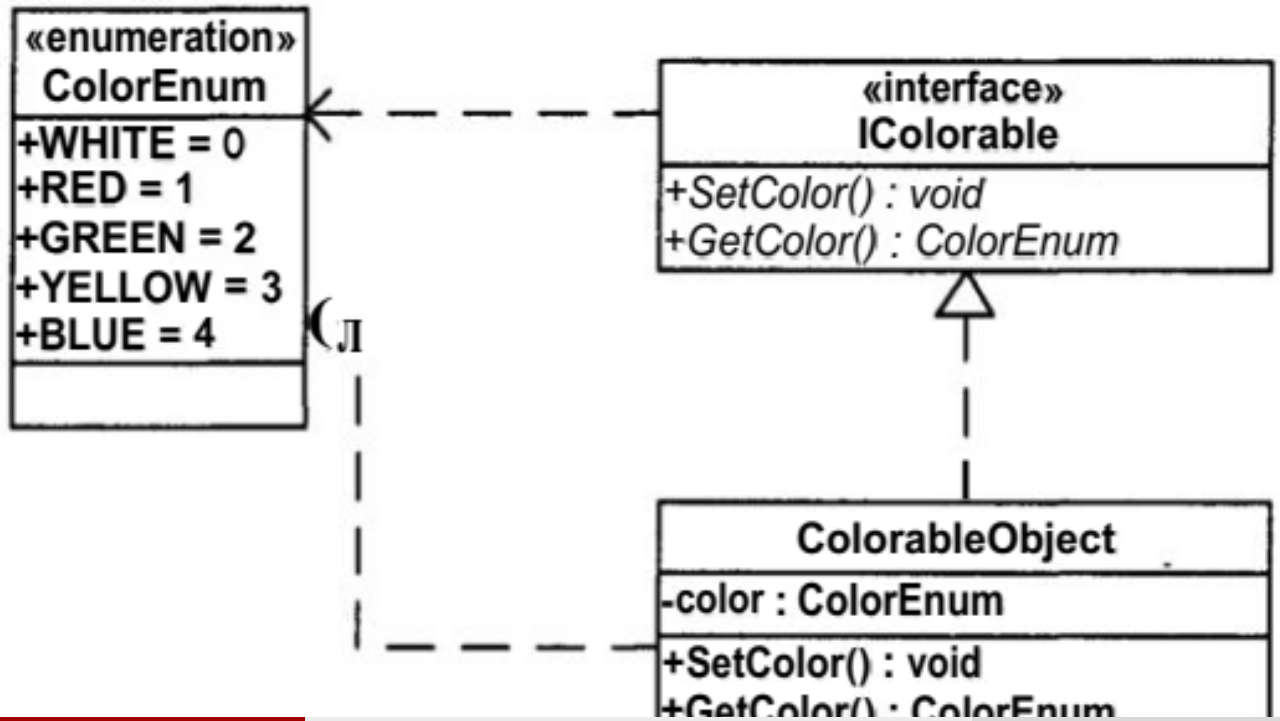


Отношение реализации

Отношение реализации

используется для определения отношения между интерфейсом и классом, реализующим интерфейс.

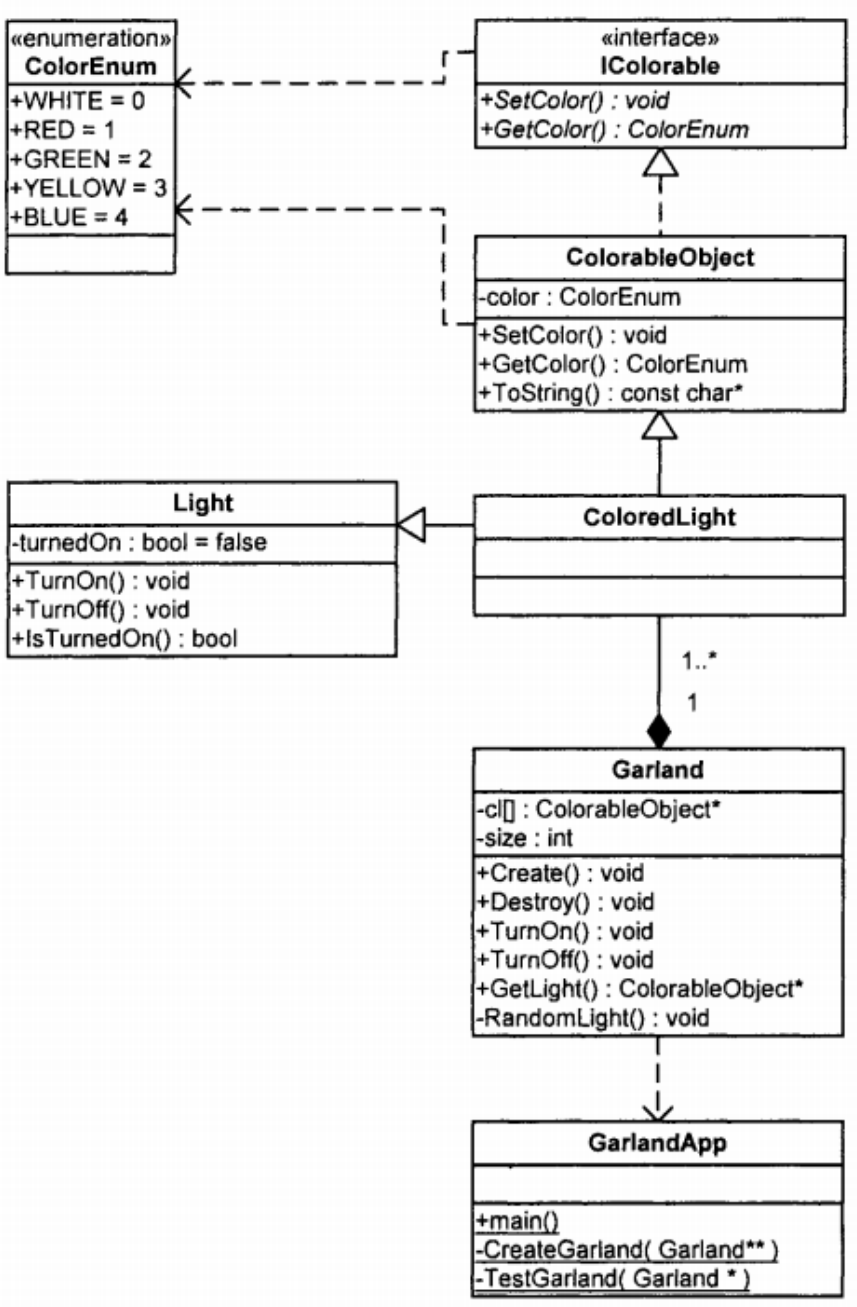
Интерфейс определяет набор элементов (как правило, действий), характерных для объектов, обладающих определенными свойствами.



Отношение реализации

Наличие интерфейсов предоставляет возможность работать с объектами разных типов в той части, которая поддерживает соответствующий интерфейс.

```
procedure SetWhite(var ic: IColorable);
begin
  ic.SetColor(WHITE);
end;
```


«enumeration»
ColorEnum
+WHITE = 0
+RED = 1
+GREEN = 2
+YELLOW = 3
+BLUE = 4
«interface»
IColorable
+SetColor() : void
+GetColor() : ColorEnum
ColorableObject
-color : ColorEnum
+SetColor() : void
+GetColor() : ColorEnum
+ToString() : const char*
Light
-turnedOn : bool = false
+TurnOn() : void
+TurnOff() : void
+IsTurnedOn() : bool
ColoredLight
1..*
1
Garland
-cl[] : ColorableObject*
-size : int
+Create() : void
+Destroy() : void
+TurnOn() : void
+TurnOff() : void
+GetLight() : ColorableObject*
-RandomLight() : void
GarlandApp
+main()
-CreateGarland(Garland**)
-TestGarland(Garland *)

Типы структурных иерархий

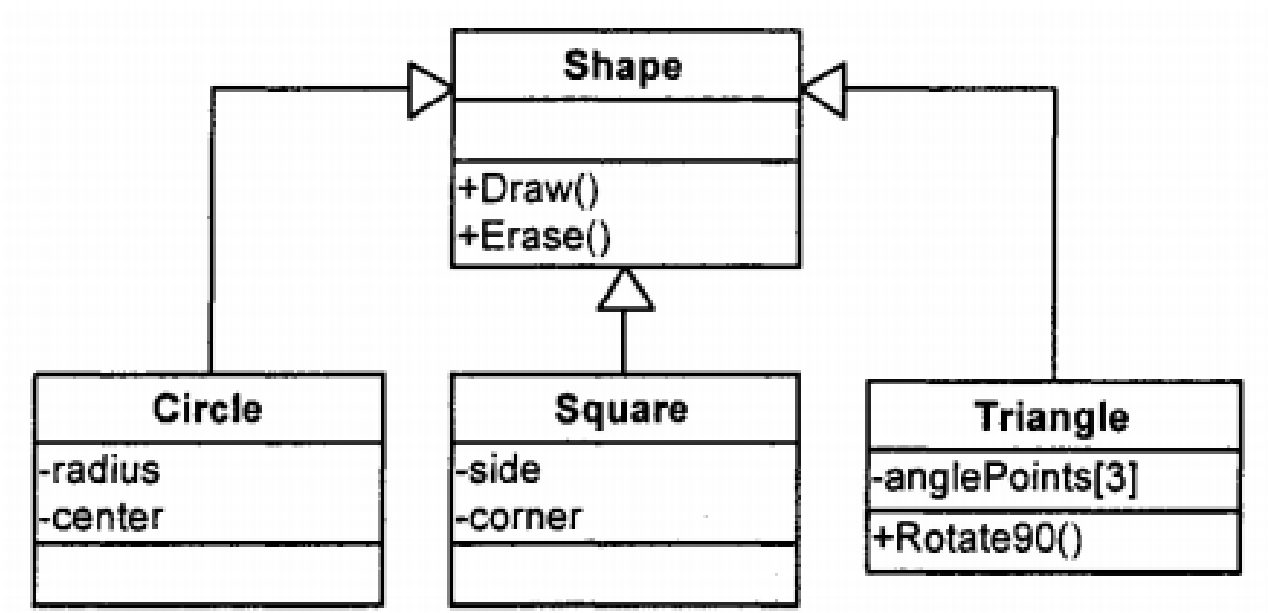
Иерархическая декомпозиция и иерархическая организация ПО образуют один из основных систематических методов преодоления сложности программного обеспечения.

- структурная иерархия "is-part-of"(агрегирование как разновидность ассоциации)
- Структурные иерархии "is-a"и "is-like-a"

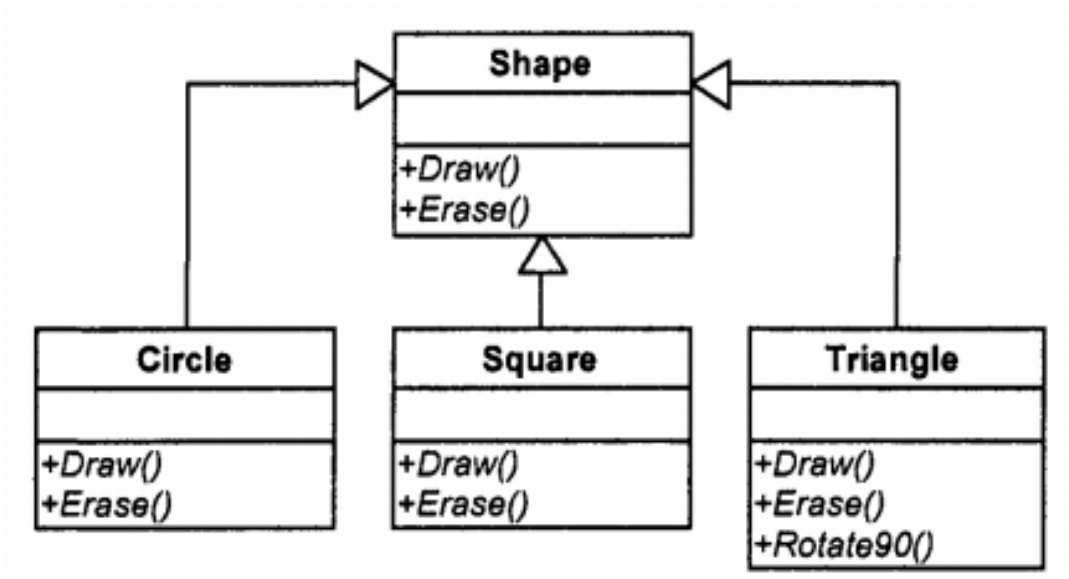
Обобщение

означает, что объекты классапотомка могут использоваться везде, где допустимы объекты родительского класса.

- Создавая базовый тип (базовый класс), программист выражает наиболее общие идеи относительно объектов, из которых конструируется программа.
- В производных классах программист уточняет различия в реализации конструкций базового класса.
- Производный класс полностью дублирует интерфейс базового класса, т. е. дублирует наблюдаемое поведение объекта базового класса.



Поведение объекта производного класса может отличаться от поведения объекта базового класса.



- может потребоваться изменение реализации уже существующих методов; (рис.
- дочерний класс предоставляет операции, замещающие операции родителя с такой же сигнатурой ("is-like-a")

Замещение (перекрытие, overriding) метода класса

механизм изменения реализации метода класса с сохранением интерфейса, определяемого базовым классом.

Полиморфизм

Поведение объектов, на которых вызываются видоизмененные методы, интерфейс к которым заявлен в базовом классе, называют **полиморфным**, а механизм, обеспечивающий такое поведение, называют **полиморфизмом**.

Мы имеем возможность:

- работая с объектами разных типов, представляющих различные абстракции, мы можем использовать методы, которые имеют одинаковый интерфейс, что, в конечном счете, упрощает и реализацию программы, и ее восприятие;
- разрабатывать общие процедуры обработки объектов.