Реализация полиморфизма. Полиморфные функции и методы

- Понятие полиморфизма
- Формы полиморфизма функций и методов
- Параметрический полиморфизм
- Чистый полиморфизм
- Перегрузка или полиморфизм ad hoc
- Параметрическая перегрузка
- Замещение методов
- Переопределение методов

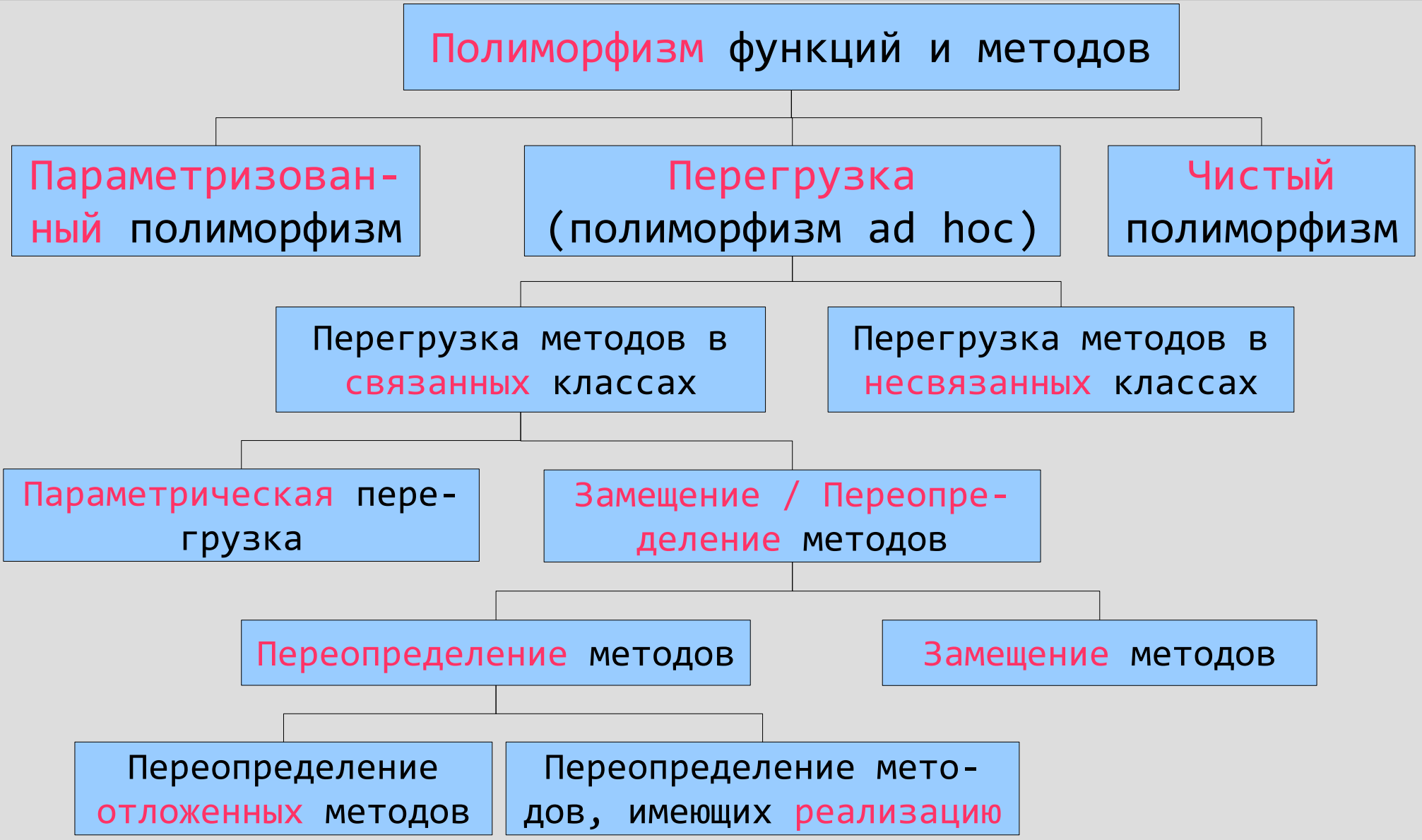
Понятие полиморфизма

- Полиморфизм в языке программирования означает многозначность переменных и функций
- Полиморфной функцией является такая функция, которая может вызываться с аргументами различного типа, а фактический выполняемый код зависит от типа аргументов

Преимущества использования полиморфизма

- Полиморфизм позволяет записывать алгоритмы лишь однажды и затем повторно их использовать для различных типов данных, которые, возможно, еще не существуют (обобщенные действия или алгоритмы)
- Полиморфизм сужает концептуальное пространство, т.е. уменьшает количество информации, которое необходимо помнить программисту

Формы полиморфизма функций и методов



Параметризованный полиморфизм

- Обеспечивается за счет так называемых обобщенных функций, которые в языке Си++ называются шаблонами
- Аргументом обобщенной функции является тип, который используется при ее параметризации

Параметрический полиморфизм

- С помощью механизма шаблонов можно создать функцию, которая бы работала с разнотипными аргументами
- Примером таких функций являются обобщенные алгоритмы из STL

Пример параметризованной функции

```
// Минимум двух объектов
// Тип T (объект) должен поддерживать
// следующие операции и методы:
// operator <, operator =
template <class T>
T & min(T &value1, T &value2)
{
   return value1 < value2 ? value1 : value2;
}

// Примеры вызова параметризованной функции
int minInt = min(3, 7);
QString minStr = min(QString("Строка 1"),
                     QString("Строка 2"));
```

Чистый полиморфизм

- Чистый полиморфизм имеет место, когда одна и та же функция применяется к аргументам различных типов
- В случае чистого полиморфизма имеется одна функция (тело кода) и несколько ее интерпретаций

Чистый полиморфизм

- Реализация чистого полиморфизма возможна только при наличии полиморфных переменных, а точнее полиморфных аргументов
- Чистый полиморфизм позволяет реализовывать обобщенные алгоритмы

Пример применения чистого полиморфизма

- Пусть имеется класс `Enumerable`, в котором объявлены операции отношения `>`, `<`, `>=`, `<=`, `==` и `!=`, и имеется функция `min()`:

```
// Минимум двух значений
Enumerable & min(Enumerable &first, Enumerable &second)
{
   return first < second ? first : second ;
}
```

- Всем классам, производным от класса `Enumerable`, будет доступна операция `min()`

Перегрузка или полиморфизм ad hoc

- Перегрузка возникает, когда имеется два или более кода, связанных с одним именем
- Главное назначение перегрузки – сужение концептуального пространства

Перегрузка методов в несвязанных классах

- Все ОО-языки разрешают использовать методы с одинаковыми именами в несвязанных между собою классах – это перегрузка методов
- В этом случае привязка перегруженного имени производится за счет информации о классе, к которому относится получатель сообщения

Параметрическая перегрузка

- Стиль перегрузки, при котором функциям и методам в одном и том же контексте разрешается использовать совместно одно имя, а двусмысленность снимается за счет анализа числа и типов аргументов, называется параметрической перегрузкой

Примеры применения перегрузки

```
// ПЕРЕГРУЗКА МЕТОДОВ В НЕСВЯЗАННЫХ КЛАССАХ
// Проверка корректности времени
bool QTime::isValid () const;
// Проверка корректности даты
bool QDate::isValid () const;
// Проверка корректности цвета
bool QColor::isValid () const;

// ПАРАМЕТРИЧЕСКАЯ ПЕРЕГРУЗКА (В ОДНОМ КЛАССЕ)
// Добавить в конец строки
QString &  append ( const QString & str );
QString &  append ( const QByteArray & ba );
QString &  append ( const char * str );
QString &  append ( QChar ch );
```

Замещение методов

- Замещение возникает, когда в базовом и производном классах имеются два метода с одинаковым именем и параметрами
- В этом случае метод базового класса перекрывается методом производного класса с точки зрения пользователя класса

Назначение механизма замещения методов

- Замещение происходит прозрачно (незаметно) для пользователя класса, и, как в случае перегрузки, два метода представляются семантически как одна сущность
- Главное назначение замещения методов – сужение концептуального пространства

Пример замещения метода

```
class MyEllipse
{
public:
   float area() const
   { /* численный метод расчета */ }

   void print() { printf( "area = %f\n", area() ); }
};
class MyCircle: public MyEllipse
{
public:
   float area() const
   {//использ. более эффективн. алгоритм расчета
      return 3.14*Radius1*Radius2;
   }
};
```

Пример замещения метода

```
MyEllipse ellipse;
MyCircle  circle;

// Будет вызван метод MyEllipse::area()
printf("Ellipse area= %f\n", ellipse.area());

// Будет вызван метод MyEllipse::area()
ellipse.print();

// Будет вызван метод MyCircle::area()
printf("Circle area= %f\n",  circle.area());

// ВНИМАНИЕ!!! Будет вызван метод MyEllipse::area()
circle.print();
```

Переопределение методов

- При замещении метод базового класса перекрывается методом производного класса только снаружи. Внутри класса вызывается метод базового класса (см. предыдущий пример)
- Переопределение метода возникает, когда метод производного класса подменяет метод базового класса не только снаружи, но и внутри класса
- В языке Си++ для переопределения метода необходимо использовать механизм динамического связывания, т.е. объявить метод виртуальным

Пример переопределения метода

```
class MyEllipse
{
public:
   virtual float area() const
   { /* численный метод расчета */ }

   void print() { printf( "area = %f\n", area() ); }
};
class MyCircle: public MyEllipse
{
public:
   float area() const
   {//использ. более эффективн. алгоритм расчета
      return 3.14*Radius1*Radius2;
   }
};
```

Пример переопределения метода

```
MyEllipse ellipse;
MyCircle  circle;

// Будет вызван метод MyEllipse::area()
printf("Ellipse area= %f\n", ellipse.area());

// Будет вызван метод MyEllipse::area()
ellipse.print();

// Будет вызван метод MyCircle::area()
printf("Circle area= %f\n",  circle.area());

// ВНИМАНИЕ!!! Будет вызван метод MyCircle::area()
circle.print();
```

Назначение механизма переопределения методов

- Наличие механизма переопределения методов позволяет реализовать в базовом классе общую часть поведения, подразумевая, что отдельные действия будут доопределены (переопределены) в производных классах
- Таким образом, главное назначение механизма переопределения методов - сокращение объема программы

Отложенные методы

- Отложенный метод – это частный случай переопределения, когда метод базового класса не имеет реализации, а любая полезная деятельность задается в методе дочернего класса

Отложенные методы в языке Си++

- В языке Си++ отложенный метод должен быть описан в явном виде с ключевым словом `virtual`
- Тело отложенного метода не определяется, вместо этого функции «приписывается» значение `0`

Пример использования отложенных методов

```
// Базовый класс содержит чисто виртуальный метод,
// т.к. невозможно отрисовать нечто неопределенное.
// Описание метода в базовом классе гарантирует его
// наличие в производных классах
class MyGraphicsPrimitive2D
{
public:
   virtual void draw() = 0;
};

class MyEllipse:   public MyGraphicsPrimitive2D
{
public
   virtual void draw()
    { /* собственная реализация */ }
};
```

Пример использования отложенных методов

```
class MyRectangle: public MyGraphicsPrimitive2D
{
public
   virtual void draw()
    { /* собственная реализация */ }
};

class MyCircle: public MyEllipse
{
   virtual void draw()
    { /* собственная реализация */ }
};
```

Пример использования отложенных методов

```
// Создаем массив для хранения ЛЮБЫХ
// 3-х графических примитивов
MyGraphicsPrimitive2D *primitives[3];

// Заполняем массив разнородными графическими
// примитивами
primitives[0]= new MyCircle();
primitives[1]= new MyEllipse();
primitives[2]= new MyRectangle();

// Отрисовываем графические примитивы
for(int i = 0; i < 3; i++)
{ primitives[i]->draw(); }
```