ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждение высшего ПРОФЕССИЛНАЛЬНОГО образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

РЕФЕРАТ

по дисциплине ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

на тему «Контрактное программирование»

Выполнил студент группы АВТ-612

Ларионова Дарья Николаевна

Принял Астапчук Виктор Андреевич

Новосибирск

2018 г

Оглавление

[Введение 3](#_Toc516113712)

[1 История 3](#_Toc516113713)

[2 Описание 3](#_Toc516113714)

[3 Реализация в языках программирования 5](#_Toc516113715)

[3.1 Поддержка DbC на языковом уровне 5](#_Toc516113716)

[3.2 Поддержка DbC с помощью сторонних библиотек 5](#_Toc516113717)

[4 Общие инструменты 6](#_Toc516113718)

[5 Предусловия 6](#_Toc516113719)

[6 Постусловия 8](#_Toc516113720)

[8 Новые возможности 10](#_Toc516113721)

[Заключение 11](#_Toc516113722)

[Список использованной литературы 12](#_Toc516113723)

# Введение

Контрактное программирование (design by contract (DbC), programming by contract, contract-based programming) — это метод проектирования программного обеспечения. Он предполагает, что проектировщик должен определить формальные, точные и верифицируемые спецификации интерфейсов для компонентов системы. При этом, кроме обычного определения абстрактных типов данных, также используются предусловия, постусловия и инварианты. Данные спецификации называются «контрактами» в соответствии с концептуальной метафорой условий и ответственности в гражданско-правовых договорах.

# 1 История

Термин предложил Бертран Мейер в связи с разработкой языка [Eiffel](https://ru.wikipedia.org/wiki/%D0%AD%D0%B9%D1%84%D0%B5%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Эйфель (язык программирования)). Контрактное программирование выросло из [формальной верификации](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D1%80%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F), [формальной спецификации](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) и [логики Хоара](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0_%D0%A5%D0%BE%D0%B0%D1%80%D0%B0). Контрактное программирование — это не только простая метафора, указывающая способ проектирования. Условия, облегчающие применение контрактного программирования:

* наличие [наследования](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и возможность [динамического связывания](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%81%D0%B2%D1%8F%D0%B7%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5),
* способность обрабатывать [исключения](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9),
* возможность автоматического [документирования ПО](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

# 2 Описание

Основная идея контрактного программирования — это модель взаимодействия элементов программной системы, основывающаяся на идее взаимных *обязательств* и *преимуществ*. Как и в бизнесе, *клиент* и *поставщик* действуют в соответствии с определённым *контрактом*. Контракт некоторого метода или функции может включать в себя:

* конкретные обязательства, которые любой клиентский модуль должен выполнить перед вызовом метода — [предусловия](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%B8%D0%B5&action=edit&redlink=1), которые дают преимущество для поставщика — он может не проверять выполнение предусловий;
* конкретные свойства, которые должны присутствовать после выполнения метода — [постусловия](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%BE%D1%81%D1%82%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%B8%D0%B5&action=edit&redlink=1), которые входят в обязательства поставщика;
* обязательства по выполнению конкретных свойств — инвариантов, которые должны выполняться при получении поставщиком сообщения, а также при выходе из метода.

Многие [языки программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) позволяют учитывать такие обязательства. Контрактное программирование подразумевает эти требования критическими для корректности программ, поэтому они должны быть утверждены при проектировании. Таким образом, контрактное программирование предписывает начинать писать код с написания формальных утверждений корректности (assertions).

В [объектно-ориентированном программировании](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) контракт метода обычно включает следующую информацию:

* возможные типы входных данных и их значение;
* типы возвращаемых данных и их значение;
* условия возникновения [исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), их типы и значения;
* присутствие [побочного эффекта](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B1%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) метода;
* предусловия, которые могут быть ослаблены (но не усилены) в подклассах;
* постусловия, которые могут быть усилены (но не ослаблены) в подклассах;
* инварианты, которые могут быть усилены (но не ослаблены) в подклассах;
* (иногда) гарантии производительности, например, [временная сложность](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0) или [сложность по памяти](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0&action=edit&redlink=1).

При использовании контрактов сам код не обязан проверять их выполнение. Обычно в таких случаях в коде делают *жёсткое падение*[[*уточнить*](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82%D0%BD%D1%8B%D0%B5_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8)] («*fail-fast*»), таким образом облегчая отладку выполнения контрактов. Во многих языках, таких как [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Delphi (язык программирования)), [PHP](https://ru.wikipedia.org/wiki/PHP), такое поведение реализуется оператором assert. В [релизовом](https://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B5%D0%BB%D0%B8%D0%B7_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)&action=edit&redlink=1" \o "Релиз (программное обеспечение) (страница отсутствует)) варианте кода это поведение может быть сохранено, либо проверки могут быть убраны чтобы повысить производительность.

[Юнит-тесты](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) проверяют модуль изолированно от других, проверяя, что модуль удовлетворяет предположениям контракта, а также свои контракты выполняют используемые им модули. [Интеграционные тесты](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) проверяют, что модули работают корректно вместе.

Контрактное программирование может повысить уровень [повторного использования кода](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0), поскольку обязательства модуля чётко документированы. Вообще, контракт модуля можно рассматривать также как способ [документации программного обеспечения](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

# 3 Реализация в языках программирования

## 3.1 Поддержка DbC на языковом уровне

Языки, изначально поддерживающие средства для контрактного программирования:

* [Active Oberon](https://ru.wikipedia.org/wiki/Active_Oberon)
* [Ada 2012](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [Clojure](https://ru.wikipedia.org/wiki/Clojure)
* Cobra
* [Component Pascal](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C)
* [D](https://ru.wikipedia.org/wiki/D_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [Eiffel](https://ru.wikipedia.org/wiki/%D0%AD%D0%B9%D1%84%D0%B5%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [Fortress](https://ru.wikipedia.org/wiki/Fortress)
* [Lisaac](https://ru.wikipedia.org/wiki/Lisaac)
* [Modula-2](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0-2)
* [Modula-3](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0-3)
* Nice
* [Oberon-2](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%B5%D1%80%D0%BE%D0%BD-2_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [Oberon-07/11](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%B5%D1%80%D0%BE%D0%BD_(%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0))
* [Oxygene](https://ru.wikipedia.org/wiki/Oxygene_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) (бывший Chrome)
* [Racket](https://ru.wikipedia.org/wiki/Racket_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* Sather
* [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* SPARK посредством [статического анализа](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%BA%D0%BE%D0%B4%D0%B0) программ на [Аде](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* Spec#
* [SeC](http://www.unitesk.ru/download/papers/ctesk/ce/CTesK2.2CELanguageReference.rus.pdf) (расширение языка [С](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)))
* [J@va](http://www.springerlink.com/content/6agpf7u9nfwrh8ef/) (расширение языка [Java](https://ru.wikipedia.org/wiki/Java" \o "Java))
* [C#](https://ru.wikipedia.org/wiki/C_Sharp) (с [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) 4.0)
* [Zonnon](https://ru.wikipedia.org/wiki/Zonnon)

## 3.2 Поддержка DbC с помощью сторонних библиотек

* [C](https://ru.wikipedia.org/wiki/C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) и [C++](https://ru.wikipedia.org/wiki/C%2B%2B) посредством [CTESK](http://www.unitesk.ru/content/category/5/13/32/), библиотеки [Contract++](http://contractpp.sourceforge.net/), [препроцессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) *DBC for C*, *GNU Nana* или компилятора C++ от [Digital Mars](https://ru.wikipedia.org/wiki/Digital_Mars" \o "Digital Mars).
* [C#](https://ru.wikipedia.org/wiki/C_Sharp) посредством *[Code Contracts](http://research.microsoft.com/en-us/projects/contracts/)*
* [Java](https://ru.wikipedia.org/wiki/Java_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) посредством [JavaTESK](http://www.unitesk.ru/content/category/5/25/60/), iContract2, [Contract4J](http://www.contract4j.org/contract4j), [jContractor](http://jcontractor.sourceforge.net/), Jcontract, [C4J](http://c4j.sourceforge.net/), CodePro Analytix, [STclass](http://www-valoria.univ-ubs.fr/stclass/), Jass preprocessor, [OVal](http://oval.sourceforge.net/) with AspectJ, [Java Modeling Language](https://ru.wikipedia.org/w/index.php?title=Java_Modeling_Language&action=edit&redlink=1" \o "Java Modeling Language (страница отсутствует)) (JML), [SpringContracts](http://springcontracts.sourceforge.net/) для [Spring Framework](https://ru.wikipedia.org/wiki/Spring_Framework" \o "Spring Framework), или [Modern Jass](http://modernjass.sourceforge.net/), [Custos](http://custos.dev.java.net/) с использованием AspectJ, [JavaDbC](http://code.google.com/p/javadbc/) с использованием AspectJ, [cofoja](http://code.google.com/p/cofoja/) (разработанная компанией [Google](https://ru.wikipedia.org/wiki/Google_(%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Google (компания))).
* [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) посредством [Cerny.js](http://www.cerny-online.com/cerny.js/), [dbc-code-contracts](https://www.npmjs.com/package/dbc-code-contracts) или [ecmaDebug](http://weblogs.mozillazine.org/weirdal/archives/016921.html).
* [Lisp](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF)
  + [Common Lisp](https://ru.wikipedia.org/wiki/Common_Lisp) с помощью макросов или [протокола метаобъектов](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%B0%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82&action=edit&redlink=1) [CLOS](https://ru.wikipedia.org/wiki/Common_Lisp_Object_System).
  + [Scheme](https://ru.wikipedia.org/wiki/Scheme_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) посредством расширения PLT, а именно тот факт, что любое нарушение контракта должно указывать на виновного и иметь точное объяснение.
* [Nemerle](https://ru.wikipedia.org/wiki/Nemerle) с помощью макросов.
* [Perl](https://ru.wikipedia.org/wiki/Perl) с помощью [CPAN](https://ru.wikipedia.org/wiki/CPAN)-модулей Class::Contract (Damian Conway) или Carp::Datum (Raphael Manfredi).
* [PHP](https://ru.wikipedia.org/wiki/PHP) с помощью [PhpDeal](https://github.com/lisachenko/php-deal)
* [Python](https://ru.wikipedia.org/wiki/Python_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) с помощью пакета zope.interface, PyDBC или Contracts for Python.
* [Ruby](https://ru.wikipedia.org/wiki/Ruby) с помощью DesignByContract (от Brian McCallister), Ruby DBC или ruby-contract.

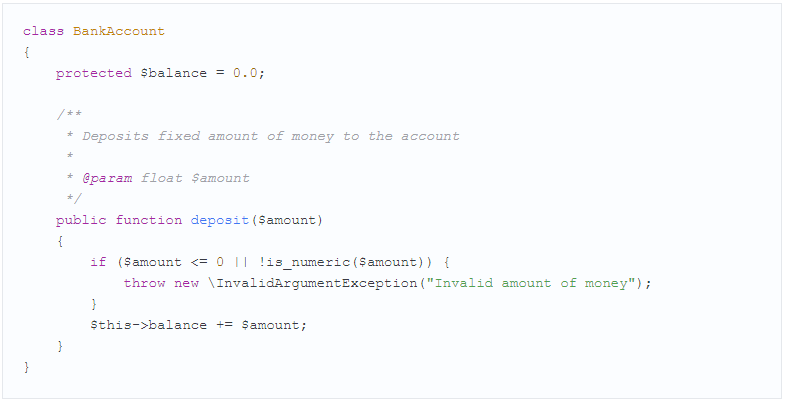
# 4 Общие инструменты

[Perfect Developer](http://www.eschertech.com/products/) посредством Perfect specification language может проверять контракт, используя [статический анализ кода](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%BA%D0%BE%D0%B4%D0%B0) и генерируя программы на языках типа [C++](https://ru.wikipedia.org/wiki/C%2B%2B) и [Java](https://ru.wikipedia.org/wiki/Java" \o "Java).

# 5 Предусловия

Самым простым вариантом контракта являются *предусловия* — требования, которые должны быть выполнены перед конкретным действием. В рамках ООП все действия описываются методами в классах, поэтому предусловия применяются к методами, а их проверка происходит в момент вызова метода, но до выполнения самого тела метода. Очевидное использование — проверка валидности переданных параметров в метод, их структуры и корректности. То есть с помощью предусловий мы описываем в контракте все то, с чем мы точно работать не будем. Это же здорово! 

Давайте рассмотрим пример:



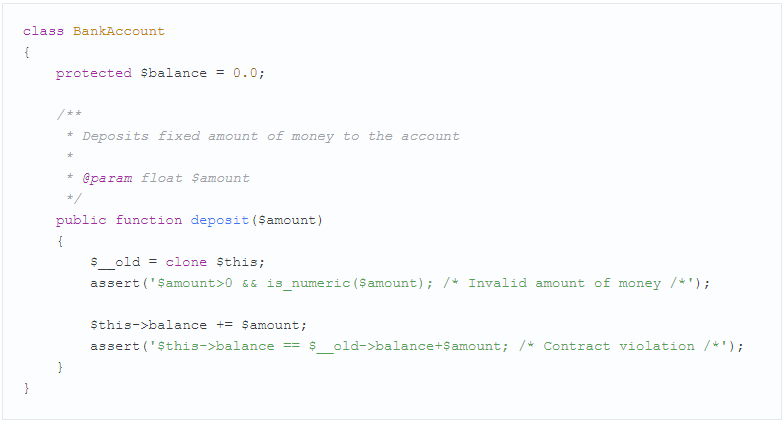
Мы видим, что метод пополнения баланса в неявном виде требует числового значения величины суммы пополнения, которая также должна быть строго больше нуля, в противном случае будет выброшено исключение. Это типичный вариант предусловия в коде. Однако он имеет несколько минусов: мы вынуждены искать глазами эти проверки и, находясь в другом классе, не можем быстро оценить наличие/отсутствие таких проверок. Также, без наличия явного контракта, нам придется помнить о том, что в коде класса есть необходимые проверки входящих аргументов и нам не надо волноваться за них. Еще один фактор: эти проверки выполняются всегда, как в режиме разработки, так и боевом режиме работы приложения, что незначительно влияет в отрицательную сторону на скорость работы приложения.  
  
В плане реализации предусловий, в PHP существует специальная конструкция для проверки утверждений — [assert()](http://php.net/assert). Большое ее преимущество в том, что проверки можно отключать в боевом режиме, заменяя весь код команды на единственный NOP. Давайте посмотрим на то, как можно описать предусловие с помощью данной конструкции:



Хочу обратить внимание на то, что предусловия в рамках контрактов служат для проверки логики работы программы и не отвечают за валидность параметров, переданных от клиента. Контракты отвечают только за взаимодействие внутри самой системы. Поэтому пользовательский ввод должен всегда фильтроваться с помощью фильтров, так как утверждения могут быть отключены. 

# 6 Постусловия

Следующая категория контрактов — *постусловия*. Как можно догадаться из названия, данный тип проверки выполняется после того, как было выполнено тело метода, но до момента возврата управления в вызывающий код. Для нашего метода deposit из примера мы можем сформировать следующее постусловие: баланс счета после вызова метода должен равняться предыдущему значению баланса плюс величина пополнения. Осталось дело за малым — описать все это в виде утверждения в коде. Но вот здесь нас поджидает первое разочарование: как же сформировать это требование в коде, ведь мы сперва изменим баланс в теле самого метода, а потом попытаемся проверить утверждение, где нужно старое значение баланса.

Здесь может помочь клонирование объекта перед выполнением кода и проверка постусловий:  


Как здесь описать контрактное условие, что метод должен возвращать текущий баланс? Так как постусловие выполняется после тела метода, то мы наткнемся на return раньше, чем сработает наша проверка. Поэтому придется изменить код метода, чтобы сохранить результат в переменную $\_\_result и сравнить потом с $this->balance:



И это для простого метода, не говоря уже о том случае, когда метод большой и в нем несколько точек возврата. Как вы уже догадались, на этом этапе идеи об использовании контрактного программирования в проекте на PHP быстро умирают, так как язык не поддерживает необходимых управляющих конструкций.   
  
7 Инварианты

Нам осталось рассмотреть еще один важный тип контрактов: *инварианты*. Инварианты — это специальные условия, которые описывают целостное состояние объекта. Важной особенностью инвариантов является то, что они проверяются всегда после вызова любого публичного метода в классе и после вызова конструктора. Так как контракт определяет состояние объекта, а публичные методы — единственная возможность изменить состояние извне, то мы получаем полную спецификацию объекта. Для нашего примера хорошим инвариантом может быть условие: баланс счета никогда не должен быть меньше нуля. Однако, с инвариантами в PHP дело обстоит еще хуже чем с постусловиями: нет никакой возможности легко добавить проверку во все публичные методы класса, чтобы после вызова любого публичного метода можно было проверить необходимое условие в инварианте. Также нет возможности обращаться к предыдущему состоянию объекта $\_\_old и возвращаемому результату $\_\_result. Без инвариантов нет контрактов, поэтому долгое время не было никаких средств и методик для реализации данного функционала.

# 8 Новые возможности

Встречайте, PhpDeal — экспериментальный DbC-[фреймворк для контрактного программирования в PHP](https://github.com/lisachenko/php-deal).   
После того, как был разработан [фреймворк Go! AOP для аспектно-ориентированного программирования в PHP](https://github.com/lisachenko/go-aop-php), у меня в голове крутились мысли насчет автоматической валидации параметров, проверки условий и много-много другого. Триггером к созданию проекта для контрактного программирования послужило обсуждение на [PHP.Internals](http://grokbase.com/t/php/php-internals/141wpxrw6r/dbc-for-php). Удивительно, но с помощью АОП задача решалась всего в пару действий: нужно было описать аспект, который будет перехватывать выполнение методов, помеченных с помощью контрактных аннотаций, и выполнять нужные проверки до или после вызова метода.

# Заключение

Контрактное программирование в PHP — абсолютно новая парадигма, которая может использоваться для защитного программирования, для улучшения качества кода и обеспечения читаемости контрактов, определяемых в виде требований и спецификаций. Большой плюс данной реализации в том, что код классов остается читаемым, сами аннотации читаются как документация, а также то, что в боевом режиме проверка может быть полностью отключена и не требует абсолютно никакого времени на дополнительные ненужные проверки в коде. Интересный факт: сам фреймоврк содержит лишь пару аннотаций и один класс аспекта, который связывает эти аннотации с конкретной логикой.

# Список использованной литературы

1) Контрактное программирование в PHP   
[URL: <https://habr.com/post/214371/>]

2) Контрактное программирование  
[URL: [https://ru.wikipedia.org/wiki/Контрактное\_программирование](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)]

3) [Фреймворк PhpDeal для контрактного программирования в PHP](https://github.com/lisachenko/php-deal)  
 [URL: <https://github.com/php-deal/framework>]