Парадигмы программирования

**парадигмы программирования** – это правила который объединяют структурированные подходу к решению задач. Так-же их можно представить как ограничения на определённые языковые конструкции, которые вынуждают использовать определённый стиль.

Парадигмы программирования определяют стандарты написания кода.

**Две основные парадигмы, императивная и декларативная, включают в себя другие.**

Императивная парадигма

**Императивная** — это самая простая и часто используемая парадигма. Её смысл в последовательном выполнении действий.

Императивной парадигмы придерживаются десятки языков, у неё много разновидностей. Раньше большинство программистов писало на процедурных языках, а теперь процедурное программирование можно назвать устаревшим. Им пользуются, только если по какой-то причине нельзя использовать объектно-ориентированное программирование, которое занимает основную нишу в императивной вселенной и относится к современным парадигмам.

Примеры языков: C, Pascal, COBOL, ALGOL, BASIC, Fortran.

Императивное программирование развивалось как процедурное, где основное понятие — это функция. Функция, или процедура — это последовательность действий, которые записали и назвали. Например, инструкция по сборке шкафа или рецепт тыквенного супа — это функции. Процедура и функция — одно и то же понятие, но процедурное и функциональное программирование различаются. Первое относится к императивной парадигме, а второе — к декларативной.

Декларативная парадигма

Декларативная парадигма требует от программиста ответа на вопрос «что должно получиться?», а не «что нужно сделать?». Компьютер ищет способ получить требуемый результат.

Если при декларативном программировании слишком размыто обозначить результат, то компьютер может начать делать не то, что нужно. Например, вместо полного описания разработчик просто сообщил, что нужен тыквенный суп. Компьютер начинает перебирать все существующие элементы и смешивать их. При таком подходе сначала может получиться куриный суп, затем грибной, и так до тех пор, пока программа не подберёт подходящие ингредиенты для тыквенного. С точки зрения декларативной парадигмы это абсолютно корректный подход — ведь в итоге он помогает достичь нужного результата. Но он совсем не эффективный, ведь будет потрачено много продуктов, денег и времени.

Объектно-ориентированное программирование

Примеры языков: Java, Python, C++, Ruby, C#, Objective-C, PHP.

В парадигме объектно-ориентированного программирования появляются объекты, которые сами выполняют функции. При таком подходе принято считать, что приготовление супа выполняет некий объект, который создаётся внутри программы. В реальности все действия в компьютере выполняет процессор, но в рамках объектно-ориентированного подхода объекты — это сущности, которые могут сами производить операции.

Объектно-ориентированное программирование позволяет регулировать связи между частями программы, которые отвечают за разные действия. За счёт этого программу легче разделить между разработчиками, проще поддерживать и легче автоматически протестировать.

Функциональное программирование

Примеры языков: Haskell, Erlang, Scala, F#, OCaml, ELM, Lisp.

К современной декларативной парадигме прежде всего относится функциональное программирование. В строгом функциональном подходе считается, что все функции чистые и не имеют побочных эффектов. Гость может заказать хоть тысячу раз тыквенный суп, и каждый раз официант принесёт ему одно и то же блюдо. Результат функции всегда будет одинаковым. Но в реальном ресторане на десятый раз закончились бы сливки 20% и их бы заменили на 10%, на двадцатый — закончилась бы и тыква, а функция стала бы выдавать ошибку «невозможно приготовить тыквенный суп». Это и есть побочный эффект — то, что изменилось в мире при работе функции, о чём пользователь может не знать.

Чтобы решить эту проблему, при функциональном подходе нельзя просто заказать суп, а нужно предъявить все продукты, необходимые для приготовления. То, что от них останется, будет возвращено в качестве результата вместе с самим супом.