Контрольные вопросы:

1. Приведите определения рефакторинга. Что отличает рефакторинг от оптимизации производительности?

**Рефа́кторинг** — процесс изменения внутренней структуры [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы.

Цель **оптимизации** — улучшение **производительности** программы, а **рефакторинга** — улучшение понятности кода. После **оптимизации** исходный код может стать сложнее для понимания. После **рефакторинга** программа может начать работать быстрее, но главное — её код становится проще и понятнее.

1. Укажите цели рефакторинга.

Цель рефакторинга — сделать код программы более легким для понимания; без этого рефакторинг нельзя считать успешным.

1. Когда следует проводить рефакторинг и когда он не нужен?

Рефакторинг нужно применять постоянно при разработке кода. Основными стимулами его проведения являются следующие задачи:

1. необходимо добавить новую функцию, которая недостаточно укладывается в принятое [архитектурное решение](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F);
2. необходимо исправить ошибку, причины возникновения которой сразу не ясны;
3. преодоление трудностей в командной разработке, которые обусловлены сложной логикой программы.
4. Приведите список признаков проблемного кода («запахи» кода).

Во многом при рефакторинге лучше полагаться на интуицию, основанную на опыте. Тем не менее имеются некоторые видимые проблемы в коде, требующие рефакторинга:

1. [дублирование кода](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%B1%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0);
2. длинный метод;
3. большой класс;
4. длинный список параметров;
5. «жадные» функции — это метод, который чрезмерно обращается к данным другого объекта;
6. избыточные временные переменные;
7. классы данных;
8. несгруппированные данные.
9. Перечислите приемы рефакторинга, касающиеся составления методов. Опишите приемы «выделение метода» и «встраивание метода».

Значительная часть рефакторинга посвящается правильному составлению методов. В большинстве случаев, корнем всех зол являются слишком длинные методы. Хитросплетения кода внутри такого метода, прячут логику выполнения и делают метод крайне сложным для понимания, а значит и изменения. Рефакторинги этой группы призваны уменьшить сложность внутри метода, убрать дублирование кода и облегчить последующую работу с ним.

* [**Извлечение метода**](https://refactoring.guru/ru/extract-method)
* [**Встраивание метода**](https://refactoring.guru/ru/inline-method)
* [**Извлечение переменной**](https://refactoring.guru/ru/extract-variable)
* [**Встраивание переменной**](https://refactoring.guru/ru/inline-temp)
* [**Замена переменной вызовом метода**](https://refactoring.guru/ru/replace-temp-with-query)
* [**Расщепление переменной**](https://refactoring.guru/ru/split-temporary-variable)
* [**Удаление присваиваний параметрам**](https://refactoring.guru/ru/remove-assignments-to-parameters)
* [**Замена метода объектом методов**](https://refactoring.guru/ru/replace-method-with-method-object)
* [**Замена алгоритма**](https://refactoring.guru/ru/substitute-algorithm)

1. Перечислите приемы рефакторинга, касающиеся перемещения функций между объектами. Опишите приемы «перемещение метода» и «перемещение поля».

Eсли вы разместили функциональность по классам не самым удачным образом — это еще не повод отчаиваться. Рефакторинги этой группы показывают как безопасно перемещать функциональность из одних классов в другие, создавать новые классы, а также скрывать детали реализации из публичного доступа.

* [**Перемещение метода**](https://refactoring.guru/ru/move-method)
* [**Перемещение поля**](https://refactoring.guru/ru/move-field)
* [**Извлечение класса**](https://refactoring.guru/ru/extract-class)
* [**Встраивание класса**](https://refactoring.guru/ru/inline-class)
* [**Сокрытие делегирования**](https://refactoring.guru/ru/hide-delegate)
* [**Удаление посредника**](https://refactoring.guru/ru/remove-middle-man)
* [**Введение внешнего метода**](https://refactoring.guru/ru/introduce-foreign-method)
* [**Введение локального расширения**](https://refactoring.guru/ru/introduce-local-extension)

1. Перечислите приемы рефакторинга, касающиеся реорганизации данных.

Рефакторинги этой группы призваны облегчить работу с данными, заменив работу с примитивными типами богатыми функциональностью классами. Кроме того, важным моментом является уменьшение связанности между классами, что улучшает переносимость классов и шансы их повторного использования.

* [**Замена значения ссылкой**](https://refactoring.guru/ru/change-value-to-reference)
* [**Замена ссылки значением**](https://refactoring.guru/ru/change-reference-to-value)
* [**Дублирование видимых данных**](https://refactoring.guru/ru/duplicate-observed-data)
* [**Самоинкапсуляция поля**](https://refactoring.guru/ru/self-encapsulate-field)
* [**Замена простого поля объектом**](https://refactoring.guru/ru/replace-data-value-with-object)
* [**Замена поля-массива объектом**](https://refactoring.guru/ru/replace-array-with-object)
* [**Замена однонаправленной связи двунаправленной**](https://refactoring.guru/ru/change-unidirectional-association-to-bidirectional)
* [**Замена двунаправленной связи однонаправленной**](https://refactoring.guru/ru/change-bidirectional-association-to-unidirectional)
* [**Инкапсуляция поля**](https://refactoring.guru/ru/encapsulate-field)
* [**Инкапсуляция коллекции**](https://refactoring.guru/ru/encapsulate-collection)
* [**Замена магического числа символьной константой**](https://refactoring.guru/ru/replace-magic-number-with-symbolic-constant)
* [**Замена кодирования типа классом**](https://refactoring.guru/ru/replace-type-code-with-class)
* [**Замена кодирования типа подклассами**](https://refactoring.guru/ru/replace-type-code-with-subclasses)
* [**Замена кодирования типа состоянием/стратегией**](https://refactoring.guru/ru/replace-type-code-with-state-strategy)
* [**Замена подкласса полями**](https://refactoring.guru/ru/replace-subclass-with-fields)

1. Перечислите приемы рефакторинга, касающиеся упрощения вызовов методов.

Эти рефакторинги делают вызовы методов проще и яснее для понимания. Это, в свою очередь, упрощает интерфейсы взаимодействия между классами.

* [**Добавление параметра**](https://refactoring.guru/ru/add-parameter)
* [**Удаление параметра**](https://refactoring.guru/ru/remove-parameter)
* [**Переименование метода**](https://refactoring.guru/ru/rename-method)
* [**Разделение запроса и модификатора**](https://refactoring.guru/ru/separate-query-from-modifier)
* [**Параметризация метода**](https://refactoring.guru/ru/parameterize-method)
* [**Замена параметров объектом**](https://refactoring.guru/ru/introduce-parameter-object)
* [**Передача всего объекта**](https://refactoring.guru/ru/preserve-whole-object)
* [**Удаление сеттера**](https://refactoring.guru/ru/remove-setting-method)
* [**Замена параметра набором специализированных методов**](https://refactoring.guru/ru/replace-parameter-with-explicit-methods)
* [**Замена параметра вызовом метода**](https://refactoring.guru/ru/replace-parameter-with-method-call)
* [**Сокрытие метода**](https://refactoring.guru/ru/hide-method)
* [**Замена конструктора фабричным методом**](https://refactoring.guru/ru/replace-constructor-with-factory-method)
* [**Замена кода ошибки исключением**](https://refactoring.guru/ru/replace-error-code-with-exception)
* [**Замена исключения проверкой условия**](https://refactoring.guru/ru/replace-exception-with-test)

1. Перечислите приемы рефакторинга, касающиеся реорганизации условных выражений.

Логика условного выполнения имеет тенденцию становиться сложной, поэтому ряд рефакторингов направлен на то, чтобы упростить ее.

* [**Объединение условных операторов**](https://refactoring.guru/ru/consolidate-conditional-expression)
* [**Объединение дублирующихся фрагментов в условных операторах**](https://refactoring.guru/ru/consolidate-duplicate-conditional-fragments)
* [**Разбиение условного оператора**](https://refactoring.guru/ru/decompose-conditional)
* [**Замена условного оператора полиморфизмом**](https://refactoring.guru/ru/replace-conditional-with-polymorphism)
* [**Удаление управляющего флага**](https://refactoring.guru/ru/remove-control-flag)
* [**Замена вложенных условных операторов граничным оператором**](https://refactoring.guru/ru/replace-nested-conditional-with-guard-clauses)
* [**Введение Null-объекта**](https://refactoring.guru/ru/introduce-null-object)
* [**Введение проверки утверждения**](https://refactoring.guru/ru/introduce-assertion)

1. Перечислите приемы рефакторинга, касающиеся решения задач обобщения.

Обобщение порождает собственную группу рефакторингов, в основном связанных с перемещением функциональности по иерархии наследования классов, создания новых классов и интерфейсов, а также замены наследования делегированием и наоборот.

* [**Подъём поля**](https://refactoring.guru/ru/pull-up-field)
* [**Подъём метода**](https://refactoring.guru/ru/pull-up-method)
* [**Подъём тела конструктора**](https://refactoring.guru/ru/pull-up-constructor-body)
* [**Спуск поля**](https://refactoring.guru/ru/push-down-field)
* [**Спуск метода**](https://refactoring.guru/ru/push-down-method)
* [**Извлечение подкласса**](https://refactoring.guru/ru/extract-subclass)
* [**Извлечение суперкласса**](https://refactoring.guru/ru/extract-superclass)
* [**Извлечение интерфейса**](https://refactoring.guru/ru/extract-interface)
* [**Свёртывание иерархии**](https://refactoring.guru/ru/collapse-hierarchy)
* [**Создание шаблонного метода**](https://refactoring.guru/ru/form-template-method)
* [**Замена наследования делегированием**](https://refactoring.guru/ru/replace-inheritance-with-delegation)
* [**Замена делегирования наследованием**](https://refactoring.guru/ru/replace-delegation-with-inheritance)