* Рефакторинг – это изменение во внутренней структуре программного кода, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения.
*  Расширение функциональности – это добавление нового поведения в систему или же изменение старого поведения определенным образом.
*  Оптимизация производительности – ставит перед собой цель – достичь максимального быстродействия не меняя при этом поведения системы. При этом, в процессе оптимизации думать о красоте кода практически некогда.
*  Рефакторинг – это непрерывный процесс, который длится в течении всей работы над кодом и даже после того, когда код написан.
*  Запахи – именно так Мартин Фаулер и Кент Бек назвали группы основных проблем, с которыми сталкиваются программисты во время работы над программным кодом.
*  Дублирование кода – обнаруживается, когда в разных методах существует код, выполняющий одни и те же действия.
*  Длинные методы – В ООП методы, размер которых превышает размер экрана, как правило, говорит о том, что вы пытаетесь втиснуть ногу структурного программирования в объектно-ориентированный ботинок.
*  Большой класс – Если класс имеет множество никак не связанных аспектов ответственности, смело разбивайте его на несколько классов, так чтобы каждый из них специализировался на своей части.
*  Длинный список параметров - Как правило, хорошо факторизованные программы включают много небольших, хорошо определенных методов, не нуждающихся в большом числе параметров.
*  Расходящиеся модификации – Программы нужно структурировать так, чтобы облегчить их модификацию, в конце концов, программы тем и отличаются от «железа», что программы менять можно безболезненно.
*  Стрельба дробью – Этот запах похож на предыдущий, но отличается тем, что при выполнении любых модификаций приходится вносить много мелких изменений во многих классах.
*  Завистливые функции – Появляется тогда, когда метод некоторого класса больше интересуется состоянием другого класса, чем состоянием своего собственного.
*  Группы данных - Если вы неоднократно используете один и тот же набор элементов данных, рассмотрите целесообразность объединения этих данных и выполняемых над ними операций в отдельный класс.
*  Одержимость элементарными типами - Элементарные типы данных могут представлять бесконечное число сущностей реального мира и, тем самым, усложнять понимание своего конкретного применения.
*  Операторы типа switch – В ООП частое использование таких операторов - это дурной тон.
*  Параллельные иерархии наследования - Если при создании каждого нового подкласса одного класса вам приходится создавать подкласс другого класса, вы имеете дело с особым видом параллельного изменения.
*  Ленивый класс – Существование некоторого класса не окупается его функциональной полезностью.
*  Теоретическая общность – Иногда бывает так, что в проект пытаются добавить функциональные возможности, которые сейчас не нужны.
*  Временное поле – Иногда обнаруживается, что в некотором объекте значение атрибуту присваивается только в определенной ситуации.
*  Цепочки сообщений – Тут все просто, проблемы возникают тогда, когда объект запрашивает данные у объекта, который запрашивает данные у объекта, который запрашивает данные у объекта…
*  Посредник – Если роль класса сводится к перенаправлению вызовов методов в другие классы, подумайте, не устранить ли его и вызывать другие классы непосредственно.
*  Неуместная близость - Инкапсуляция (сокрытие информации) — наверное, самый эффективный способ улучшения интеллектуальной управляемости программ и минимизации волновых эффектов изменений кода.
* Альтернативные классы с разными интерфейсами – Возникает тогда, когда у вас есть классы, выполняющие одни и те же действия, но отличающиеся друг от друга интерфейсом взаимодействия со своим пользователем.
*  Неполнота библиотечного класса – Использовать библиотеки очень удобно. Именно благодаря им никто не узнает, забыли вы какой-то способ сортировки или вообще никогда его не знали. Но разработчики библиотек тоже люди, и они не всеведущи.
*  Классы данных – Класс не делает ничего, кроме хранения данных и предоставления доступа к ним.
*  Отказ от наследства - Обычно такая ситуация возникает, когда подкласс создается потому, что базовый класс по чистой случайности содержит нужные ему методы, а не потому, что подкласс логически является потомком базового класса.
*  Комментарии - В важности комментариев трудно усомниться, но их не следует использовать для объяснения плохого кода.

**Закрепление материала**

*  Что такое рефакторинг?
*  В чем его отличия от оптимизации производительности?
*  Когда следует производить рефакторинг?
*  Когда от рефакторинга стоит воздержаться?
*  Каковы преимущества рефакторинга?
*  Что такое «запахи» в коде?
*  С какими проблемами чаще всего сталкиваются программисты при работе над большими проектами?