1. Что такое объектно-ориентированный подоход.

Объектно-ориентированный подход - это такой способ программирование, который напоминает процесс человеческого мышления. ООП более структурировано, чем другие способы программирования (например, процедурное) и позволяет создавать модульные программы с представление данных на определенном уровне абстракции. Основная цель ООП - это повышение эффективности разработки программ.

Весь окружающий мир состоит из объектов, которые представляются как единое целое, и такие объекты взаимодействуют друг с другом. Базом в ООП является понятие объекта, который имеет определенные свойства. Каждый объект знает как решать определенные задачи, то есть располагает методами решения. Программа, написанная с использованием ООП, состоит из объектов, которые могут взаимодействовать друг с другом. Все объекты с одинаковыми наборами атрибутов принадлежат к одному классу. Объединение объектов в классы определяется семантикой, то есть смыслом. Каждый класс имеет свои особенности поведения, которые определяют этот класс. Один класс отличается от другого именем и набором "сообщений", которые можно посылать объектам данных классов (интерфейсом).

Характеристики ООП:

- все является объектом
- объекты взаимодействуют между собой путем обмена сообщениями, при котором один объект требует, чтобы другой объект выполнил некоторое действие.
- каждый объект является представителем класса, который выражает общие свойства объектов.
- в классе задается функциональность (поведение) объекта, а все объекты одного класса могут выполнять одни и те же действия.
- классы организованы в иерархическую структуру.

ООП основывается на 4х принципах:

Абстракция - выделение некоторых существенных характеристик объекта, которые выделяют его из всех других объектов. Абстракция позволяет отделить существенные особенности поведения объекта от деталей их реализации. Например, главной характеристикой объекта "директор" будет то, что этот объект чем-то управляет, а чем и как именно (финансами или персоналом), то есть детали реализации, являются второстепенной информацией.

Инкапсуляция - это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя. Иными словами, инкапсуляция - это договор для объекта, что он должен скрыть, а что открыть для доступа другим объектам. Это свойство позволяет защитить важные данные для компонента, кроме того, оно дает возможность пользователю не задумываться о сложности реализации используемого компонента, а взаимодействовать с ним посредством предоставляемого интерфейса - публичных данных и методов.

Наследование - это свойство системы, позволяющее описать новый класс на основе уже существующего класса, при этом функциональность может быть заимствована частично или полностью. Класс, от которого производится наследование, называется базовым или суперклассом, а новый класс называется потомком, дочерним или производным классом. Производный класс **расширяет** функциональность базового класса благодаря добавлению новой функциональности, специфической только для производного класса.

Полиморфизм - это свойство системы, позволяющее использовать один и тот же интерфейс для общего класса действий. При этом каждое действие зависит от конкретного объекта. То есть полиморфизм - это способность выбирать более конкретные методы для исполнения в зависимости от типа объекта.

При обсуждении принципов ООП следует упомянуть 5 принципов SOLID:

1. Единственность ответственности (Single Responsibility)
2. Принцип открытости и закрытости (Open/Close Principle)
3. Принцип замещения Лисков (The Liskov Substitution Principle)
4. Принцип разделения интерфейса (The Interface Segregation Principle)
5. Инверсия зависимости (The Dependency Inversion Principle)

6. Работа с объектами типа StringBuilder и StringBuffer.

Строки в Java являются неизменяемыми объектами, но часто необходимо получить такой строковый объект, который можно будет изменять. Для этого используются классы StringBuilder и StringBuffer, объекты которых являются изменяемыми. Единственным отличием StringBuilder от StringBuffer является потокобезопасность StringBuffer. В однопоточном использовании StringBuilder практически всегда значительно быстрее, чем StringBuffer. Для этих классов не переопределены методы equals() и hashCode(), то есть сравнить содержимое двух объектов невозможно, а хэш-коды всех объектов этого типа вычисляются так же, как и для класса Object.

Конструкторы StringBuilder:
StringBuilder(String str) – создает StringBuilder, значение которого устанавливается в передаваемую строку, плюс дополнительные 16 пустых элементов в конце строки

StringBuilder(CharSequence charSeq) – создает StringBuilder, содержащий те же самые символы, что в CharSequence, плюс дополнительные 16 пустых элементов, конечных CharSequence

StringBuilder(int length) – создает пустой StringBuilder с указанной начальной вместимостью

StringBuilder() – создает пустой StringBuilder из 16 пустых элементов

Чтение и изменение объекта StringBuilder:
int length() – возвращает количество символов в строке

char charAt(int index) – возвращает символьное значение, расположенное на месте index

void setCharAt(int index, char ch) – символ, расположенный на месте index, заменяется символом ch

CharSequence subSequence(int start, int end) – возвращает новую подстроку

7. Определение понятия исключения и исключительной ситуации, оператор try-catch.

Исключительные ситуации (исключения) возникают во время выполнения программы, когда появившаяся проблема не может быть решена в текущем контексте. При возникновении исключения в приложении создается объект, описывающий это исключение, текущий ход выполнения приложения останавливается, и включается механизм обработки исключений. При этом ссылка на объект-исключение передается обработчику исключений, который пытается решить возникшую проблему и продолжить выполнение программы. Исключение — это источник дополнительной информации о ходе выполнения приложения. Такая информация позволяет выявить ошибки на ранней стадии.

Каждой исключительной ситуации поставлен в соответствие некоторый класс, экземпляр которого инициируется при исключительной ситуации. Если подходящего класса не существует, то он может быть создан разработчиком.

Управление обработкой исключений в Java осуществляется с помощью пяти ключевых слов: try, catch, throw, throws и finally. Для задания блока программного кода, который требуется защитить от исключений, используется ключевое слово try. В этом блоке и генерируется объект исключения, если возникает исключительная ситуация, после чего управление передается в соответствующий блок catch. Блок catch помещается сразу после блока try, в нем задается тип исключения, которое требуется обработать. После блока try может быть несколько блоков catch для разной обработки разных исключений, может быть один блок catch, объединяющий разные исключения для их

одинаковой обработки, а может и вообще не быть блока catch, но в этом случае обязательно должен быть блок finally. Общая форма обработки выглядит так:

```
try {
   \\код, в котором может возникнуть исключительная ситуация
} catch (тип_исключения_1 e) {
   \\обработка исключения 1
} catch (тип_исключения_2 e) {
   \\обработка исключения 2
} finally {
   \\действия, которые должны быть выполнены независимо от того, возникла исключительная ситуация или нет
}
```

---

8. Что такое "главный поток"? Как получить ссылку на главный поток выполнения программы?

При запуске программы автоматически создается главный поток — поток, который выполняет метод main(), то есть главный метод программы. В рамках главного потока могут создаваться (запускаться) дочерние потоки (подпотоки), в которых, в свою очередь также могут запускаться потоки, и так далее. Главный поток от прочих потоков отличается тем, что создается первым. Главный поток, как правило, является и последним потоком, завершающим выполнение программы.

Несмотря на то что главный поток исполнения создается автоматически при запуске программы, им можно управлять через объект класса Thread. Для этого достаточно получить ссылку на него, вызвав метод currentThread(). Этот метод возвращает ссылку на тот поток исполнения, из которого он был вызван. Если этот метод вызывать в главном методе программы (инструкция вида Thread.currentThread()), получим ссылку на главный поток. Получив ссылку на главный поток, можно управлять им таким же образом, как и любым другим потоком исполнения.

---

9. Анонимные классы: объявление и правила работы.

Анонимный класс - это локальный класс без имени. Анонимные или безымянные классы декларируются внутри методов основного класса и могут быть использованы только внутри этих методов. Анонимный класс расширяет другой класс, применяется для придания уникальной функциональности отдельно взятому экземпляру. То есть можно объявить анонимный класс, который будет расширять другой класс или реализовывать интерфейс при объявлении единственного объекта, когда остальным объектам этого класса будет соответствовать реализация метода, определенная в самом классе.

Основное требование к анонимным классами - они должны наследовать или расширять уже существующий класс или интерфейс. Анонимные классы не могут содержать статические поля и методы, кроме final static.

Классический пример использования анонимного класса - запуск потоков:

```
new Thread(new Runnable() {
        public void run() {
                //какой-то код
   }
}).start();
```

Синтаксис создания анонимного класса базируется на использовании оператора new с именем класса (интерфейса) и телом анонимного класса. Невозможно определить или переопределись конструктор анонимного класса. Анонимные классы могут быть

вложены друг в друга, хотя так делать не рекомендуется из-за сложность понимания такого кода.

Ситуации, в которых следует использовать внутренние классы:

• выделение самостоятельной логической части сложного класса;
• сокрытие реализации;
• одномоментное использование переопределенных методов. В том числе обработка событий;
• запуск потоков выполнения.

---

10. Selection Sort, принцип и краткое описание алгоритма

Принцип

На каждом i-ом шаге алгоритма находится минимальный элемент в неотсортированной части массива и меняется местами с i-ым элементом массива.

Алгоритм

на каждом i-ом шаге алгоритма (всего n шагов алгоритма):

1) находим индекс минимального элемента среди всех неотсортированных элементов;
2) меняем местами элемент с найденным индексом и i-ый элемент;

Свойства

- время работы: лучшее – O(n^2), среднее – O(n^2), худшее – O(n^2);
- затраты памяти – O(1);
- неустойчивая;
- количество обменов – обычно O(n) обменов;

Плюсы

- проста в реализации;
- не нужна дополнительная память;
- не больше O(n) обменов;

Минусы

- не ускоряется на частично отсортированных массивах;
- неустойчивая;
- медленная;

---

11. Компиляция и запуск в консольном режиме.

Сначала нужно скомпилировать класс, то есть перевести его в байт-код. Для этого в состав JDK входит компилятор Java кода javac:

E:\EJC>javac -sourcepath src -d out src/tasks/task_01/Main.java

-sourcepath задает каталог исходного кода, что позволяет компилятору найти не только тот класс, который мы отправили на компиляцию, но и найти в каталоге исходного кода другие классы, в т.ч. классы из других пакетов, которые используются в классе, отправленном на компиляцию.

С помощью **-d** мы отметили директорию, в которой будут лежать скомпилированные классы. Именно в **out** компилятор автоматически создает структуру каталогов идентичную структуре каталогов в **src**. Далее компилятор находит класс, который мы отправили на компиляцию **Main.java**, и компилируется он и все классы, которые он использует, в байткод в соответствующие каталоги в каталоге **out**.

Теперь мы можем запустить нашу программу с помощью java:

E:\EJC>java -cp out tasks.task_01.Main

С помощью **-cp** (или **-classpath**) указываем из какого каталога мы загружаем файлы, указываем класс, который запускаем.

Чтобы не прописывать полный путь к **java** и **javac** в консоли, можно задать переменную среды окружения. Если этого не делать, то придется полностью прописывать путь размещения **javac** и **java**.