1. Модификаторы доступа.

Java обеспечивает контроль доступа через модификаторы доступа, таким образом реализуется принцип инкапсуляции:

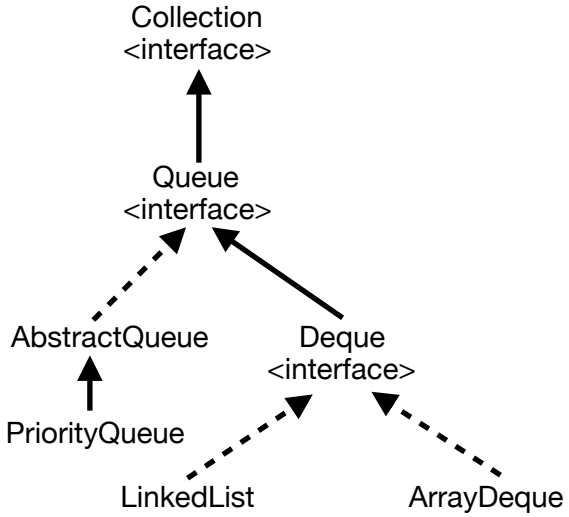
- public - доступ к компоненту из экземпляра любого класса и любого пакета;
- protected - доступ к компоненту из экземпляра родного класса или классов-потомков, а также внутри пакета;
- default (package-private) - доступ к компоненту только внутри пакета;
- private - доступ к компоненту только внутри класса.

Модификаторы доступа могут использоваться с классами, переменными и методами.

С классом могут быть использованы только модификаторы public и default: доступ к public классу можно получить из любого другого класса, в исходном файле может существовать только один public класс, а имя исходного файла должно совпадать с именем public класса.

Внутри класса, то есть с переменными-членами и методами класса могут использоваться все 4 модификатора, однако они не должны быть более доступны, чем сам класс.

2. Очереди (Queue, Dequeue). Конкретные классы очередей.



Очередь (Queue) - это очередь, обычно (но необязательно) строится по принципу FIFO (First-In-First-Out), соответственно извлечение элемента осуществляется с начала очереди, а вставка элемента в конец очереди. Этот принцип нарушает, к примеру, приоритетная очередь (PriorityQueue). Очередь предназначена для хранения элементов перед их обработкой. Кроме базовых методов Collection, очередь предоставляет дополнительные методы по добавлению, извлечению и проверке элементов.

Методы интерфейса Queue:

boolean offer(E o)	добавляет в конец очереди новый элемент, возвращает true, если добавление прошло успешно
E peek()	возвращает первый элемент очереди
E poll()	возвращает и удаляет первый элемент очереди
E element()	возвращает головной элемент очереди
E remove()	возвращает и удаляет головной элемент очереди

PriorityQueue – это класс очереди с приоритетами. По умолчанию очередь с приоритетами размещает элементы согласно естественному порядку сортировки используя Comparable. Элементу с наименьшим значением присваивается наибольший приоритет. Если несколько элементов имеют одинаковый наивысший элемент – связь определяется произвольно. Также можно указать специальный порядок размещения, используя Comparator. Конструктор PriorityQueue() создает очередь с начальной емкостью

11, а элементы размещаются согласно естественному порядку сортировки. Также есть конструкторы, в которых можно задать свою емкость и свой порядок сортировки. При добавлении элементов емкость растет автоматически. Добавление, поиск и извлечение элементов выполняется за O(logn).

Интерфейс Deque расширяет Queue - это двусторонняя очередь, может строиться по принципам FIFO и LIFO. В этой очереди элементы можно добавлять как в начало, так и в конец, а также брать элементы тоже можно и из начала, и из конца очереди.

Методы интерфейса Deque аналогичный методам Queue, но позволяют "работать" с двух сторон:

void addFirst(E e);	аналогичен offerFirst, но выбрасывает exception, если вставка не удалась
void addLast(E e);	
boolean offerFirst(E e);	
boolean offerLast(E e);	
E peekFirst();	
E peekLast();	
E pollFirst();	
E pollLast();	
E removeFirst();	
E removeLast();	
E getFirst();	аналогичен peekFirst, но выбрасывает NoSuchElementException, если очередь пустая
E getLast();	

ArrayDeque - реализация интерфейса Deque переменного размера. Элементы в ArrayDeque расположены линейно и связаны с двумя соседями в обе стороны. Емкость по умолчанию - 16, но при создании можно задать свое значение емкости. Извлечение любого элемента выполняется за O(n), а концевых элементов за O(1).

LinkedList - двусвязный список, реализация интерфейсов List и Deque, однако это "больше" список, чем очередь. Но так как LinkedList позволяет добавлять элементы в начало и конец списка за константное время, это хорошо подходит для реализации интерфейса Deque.

---

3. Блок try-с ресурсами.

В Java 7 реализована возможность автоматического закрытия ресурсов в блоке try с ресурсами (try with resources). В операторе try открывается ресурс (файловый поток ввода), который затем читается. При завершении блока try данный ресурс автоматически закрывается, поэтому нет никакой необходимости явно вызывать метод close() у потока ввода, как это было в предыдущих версиях Java. Автоматическое управление ресурсами возможно только для тех ресурсов, которые реализуют интерфейс java.lang.AutoCloseable: InputStreamReader, FilterInputStream, FileReader и многие другие.

```
try (BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {
    System.out.println(br.readLine());
} catch (IOException e) {
    e.printStackTrace();
}
```

---

4. Потоки-демоны: назначение, создание, случаи применения.

Потоки-демоны используются для работы в фоновом режиме вместе с программой, но не являются неотъемлемой частью логики программы. Примером такого потока может служить сборщик мусора. Если какой-либо процесс может выполняться на фоне работы основных потоков выполнения и его деятельность заключается в обслуживании основных потоков приложения, то такой процесс может быть запущен как поток-демон. С помощью

метода setDaemon(boolean value), вызванного вновь созданным потоком до его запуска, можно определить поток-демон. Метод boolean isDaemon() позволяет определить, является ли указанный поток демоном или нет.

Базовое свойство потоков-демонов заключается в возможности основного потока приложения завершить выполнение потока-демона (в отличие от обычных потоков) с окончанием кода метода main(), не обращая внимания на то, что поток-демон еще работает.

---

5. Пакеты (package) - правила создания, правила именования, назначение.

Пакеты – это контейнеры классов, которые используются для разделения пространства имен классов и позволяют логически объединить классы в наборы. Основные классы java входят в пакет java.lang. Различные вспомогательные классы располагаются в пакете в java.util.

Структура пакетов в точности отображает структуру файловой системы. Все файлы с исходными кодами и байт-кодами, образующие один пакет, хранятся в одном каталоге файловой системы. Пакет может содержать подпакеты.

Наименование пакета может быть любым, но необходимо соблюдать его уникальность в проекте. Компоненты доменного имени в объявлении package перечисляются в обратном порядке: package com.google.android.maps.

Все имена классов и интерфейсов в пакете должны быть уникальными. Имена классов в разных пакетах могут совпадать. Чтобы указать, что класс принадлежит определенному пакету, следует использовать директиву package, после которой указывается наименование (путь) пакета:

```
package company.common;

public class HelloWorld{
   public static void main(String[] args){
      System.out.println ("Hello, World!");
   }
}
```

Если оператор package не указан, классы попадают в безымянное пространство имен, используемое по умолчанию.

Пакеты регулируют права доступа к классам и подклассам. Если ни один модификатор доступа не указан, то класс, метод или переменная является доступной всем методам в том же самом пакете.

Чтобы использовать какой-то класс в коде другого класса, необходимо его импортировать, написав до объявления класса строку: import company.common.HelloWorld. Можно импортировать весь пакет: import company.common.*

---

6. Что такое сортировка, зачем она нужна, какие виды сортировок бывают, какие у них есть свойства.

Сортировка – это алгоритм упорядочивания элементов по какому-либо признаку. Сортировка необходима для обеспечения удобства поиска информации. Например, у нас есть список из 1000 людей разного возраста. В данный момент нам может быть необходимо найти 20 самых младших людей, можно проходить каждый раз по всему массиву данных о людях, находить самого младшего, записывать в отдельную структуру данных, потом проходить вновь, искать следующего младшего, проверяя, что такого уже не записано в нашу результирующую структуру данных. Это не очень удобно. В отсортированных данных по возрасту нам нужно было бы просто отобрать первые 20 (или последние – в случае убывающей сортировки) человек и все. К тому же это в данный момент нам нужно было отобрать 20 самых младших людей, а спустя пять минут будет нужно найти 4 старших человека или людей определенного возраста. Поиск информации в беспорядочно расположенных данных трудоемок. В отсортированных данных поиск

информации происходит гораздо быстрее, тем более для отсортированных данных могут применяться специальные виды поиска (например, бинарный поиск), которые ищут информацию с еще большей производительностью.

Основные свойства сортировок:

- **время работы** – это свойство зачастую считается наиболее важным. Оценивается худшее, среднее и лучшее время работы алгоритма. У большинства алгоритмов временные оценки бывают O(n*log n), O(n^2);
- **дополнительная память** – свойство сортировки, показывающее, сколько дополнительной памяти требуется алгоритму. Сюда входят дополнительный массив, переменные, затраты на стек вызовов. Обычно доп. затраты памяти составляют O(1), O(log n), O(n);
- **устойчивость** – устойчивой называется сортировка, не меняющая порядок объектов с одинаковыми ключами. Ключ – поле элемента, по которому проводим сортировку;
- **количество обменов** – этот параметр может быть важен в том случае, если обмениваемые объекты имеют большой размер, т.к. при большом количестве обменов время работы сильно увеличивается;

Некоторые виды сортировок:

- **Пузырьковая сортировка**
  - время работы: лучшее – O(n), среднее – O(n^2), худшее – O(n^2);
  - затраты памяти – O(1);
  - устойчивая;
  - количество обменов – обычно O(n^2) обменов;
- **Сортировка выбором**
  - 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;
- **Сортировка вставками**
  - время работы: лучшее – O(n), среднее – O(n^2), худшее – O(n^2);
  - затраты памяти – O(1);
  - устойчивая;
  - количество обменов – обычно O(n^2) обменов;
- **Быстрая сортировка**
  - время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n^2);
  - затраты памяти – O(log n);
  - неустойчивая;
  - количество обменов – обычно O(n*log n) обменов;
- **Сортировка слиянием**
  - время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n*log n);
  - затраты памяти – O(n);
  - устойчивая;
  - количество обменов – обычно O(n*log n) обменов;
- **Поразрядная сортировка**
  - время работы: лучшее – O(nk), среднее – O(nk), худшее – O(nk);
  - затраты памяти – O(n + k);
  - устойчивая;
  - количество обменов – обычно O(nk) обменов;

---

7. JDK - определение, назначение.

Java Development Kit – по сути является комплектом разработчика приложений на языке Java, включает в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, утилиты и **JRE**.

Java Runtime Enviroment (JRE) – минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки. Состоит из Java Virtual Machine и библиотеки Java-классов.

Java Virtual Machine (JVM) – виртуальная машина Java, является основной частью **JRE**. **JVM** интерпретирует байт-код Java, предварительно созданный из исходного текста Java-программы компилятором (javac). **JVM** может также использоваться для выполнения программ, написанных на других языках программирования (Kotlin, Scala и т.п.).