1. Наследование: преимущества и недостатки, альтернатива.

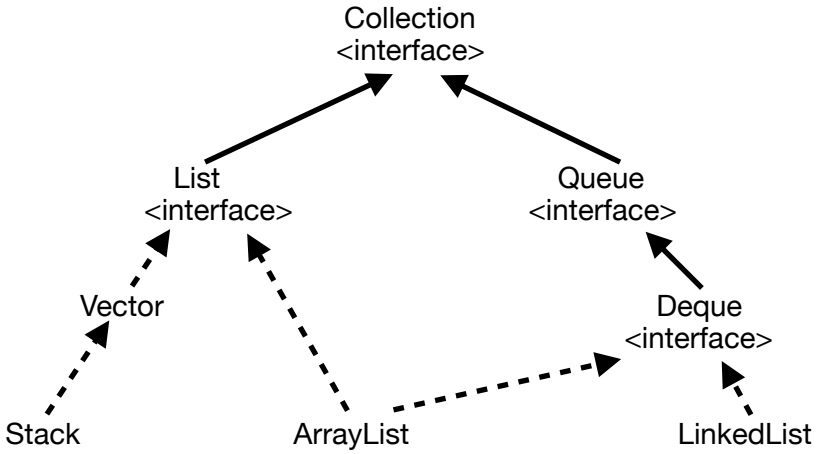
Наследование - это свойство системы, позволяющее описать новый класс на основе уже существующего класса, при этом функциональность может быть заимствована частично или полностью. Класс, от которого производится наследование, называется базовым или суперклассом, а новый класс называется потомком, дочерним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

Главное преимущество наследования - это предотвращение дублирования кода. Общее поведение для группы классов абстрагируется и помещается в один базовый класс. Если необходимо будет внести какие-то изменения в функциональность всей иерархии, то придется это сделать только один раз - в базовом классе. Таким образом облегчается сопровождение программы и увеличивается скорость разработки. Благодаря наследованию мы получаем преимущества полиморфизма, а это значит, что появляется возможность писать универсальный и чистый код, который не только быстро писать, но и легко масштабировать.

Главный, но не единственный, недостаток наследования - это сильная связанность кода: каждый производный класс зависит от реализации базового класса, а каждое изменение базового класса затронет любой производный класс. Более того, часто требуется совмещать в объекте поведение, характерное для двух и более независимых иерархий. В некоторых языках программирования эта возможность реализована за счет множественного наследования, однако в Java множественное наследование запрещено по разным причинам, например, для избежания так называемого ромбовидного наследования. Альтернативой множественного наследования в Java являются интерфейсы. Интерфейсы - это классы, в которых реализация методов не представлена, то есть все методы абстрактные. Класс в Java может реализовывать (implements) произвольное число интерфейсов, а проблема дублирования одноименных методов (по одному от каждого родителя) отсутствует, так как в интерфейсах методы не реализованы.

Другая особенность наследования заключается в том, что оно относится к поведению объектов класса - наследники должны быть устроены так, чтобы отличия в их внутреннем устройстве никак не влияло на абстракцию их поведения. Внутренне устройство подчеркивается в таком механизме, как композиция. Композиция (или агрегация) - это описание объекта как состоящего из других объектов. И если наследование характеризуется отношением "is-a", то композиция - "has-a". Композиция позволяется объединить отдельные части в единую, более сложную систему. Композиция во многих случаях может служить альтернативой множественному наследованию, причем тогда, когда необходимо унаследовать от двух и более классов их поля и методы. Кроме того, композиция позволяет повторно использовать код даже из final класса.

2. Основные характеристики списков (List). Конкретные классы списков.



Список (List) - упорядоченная коллекция, в которой могут содержаться повторяющиеся элементы. В списке сохраняется последовательность добавления элементов, поэтому доступно обращение к элементу по индексу.
Особенности:

- работа с элементами, основанная на их позиции в списке (на их индексе): get, set, add, addAll, remove
- обеспечение возможности поиска элемента и возвращение его индекса в списке: indexOf, lastIndexOf
- обеспечение получения подписков, которые являются представление изначального списка
- возможность двустороннего обхода списка, выполнения вставки и замещения элементов

Методы:

E get(int index);	возвращает объект, находящийся в позиции index
E set(int index, E element);	заменяет элемент, находящийся в позиции index объектом element
boolean add(E element);	добавляет элемент в список
boolean addAll(Collection c)	добавляет все элементы коллекции в список
E remove(int index);	удаляет элемент, находящийся на позиции index
void clear()	удаляет все элементы из списка
int indexOf(Object o);	возвращает индекс первого появления элемента
int lastIndexOf(Object o);	возвращает индекс последнего появления элемента
List<E> subList(int from, int to);	возвращает новый список, представляющий собой часть данного (с from до to-1 включительно)

Реализации интерфейса List:

LinkedList - двунаправленный список, то есть каждый элемент списка содержит указатели на предыдущий и следующий элемент в списке. Итератор поддерживает обход в обе стороны. Реализует методы получения, удаления и вставки в любое место списка, при этом быстрое добавление и и удаление элементов является преимуществом структуры. Позволяет добавлять любые элементы, включая null. Поиск элементов выполняется за O(n), вставка за O(1). Рекомендуется использовать, если необходимо часто вставлять и удалять элементы.

ArrayList - списочный массив. Массивы в Java имеют фиксированную длину, которая не может быть изменена после создания массива. ArrayList имеет возможность менять свой размер после создания. Размер по умолчанию - 10, который каждый раз при заполнении увеличивается в 1.5 раза. Поиск элементов выполняется за O(1), вставка и удаление элемента в конце массива выполняется за O(1), а вставка и удаление из произвольного места за 0(n). В ArrayList нет дополнительных расходов по памяти на хранение связок между элементами.

Vector - аналогичен ArrayList, но потокобезопасный. Работает медленнее, чем ArrayList.

Stack - коллекция, объединяющая элементы в стек. Позволяет создавать очередь типа LIFO (Last-In-First-Out). "Взять" можно только тот элемент, который был добавлен последним. Является потокобезопасным. Методы:

E peek()	возвращает верхний элемент
E pop()	возвращает и удаляет верхний элемент
E push(E item)	добавляет элемент в вершину стека
int search(Object o)	ищет элемент в стеке, возвращая количество операций pop, которые требуются для того, чтобы перевести элемент в вершину стека. Если элемент не найдет, возвращает -1

---

3. Особенности работы блока finally.

Возможна ситуация, при которой нужно выполнить некоторые действия по завершению программы вне зависимости от того, произошло исключение или нет. В этом

случае используется блок finally, который обязательно выполняется после инструкций try или catch. Каждому блоку try должен соответствовать хотя бы один блок catch или finally. Блок finally обычно используется для освобождения ресурсов, захваченных при выполнении методы, например, для закрытия потоков и файлов. Этот блок выполняется перед выходом из метода даже если перед ним были выполнены такие инструкции, как return, break и тд:

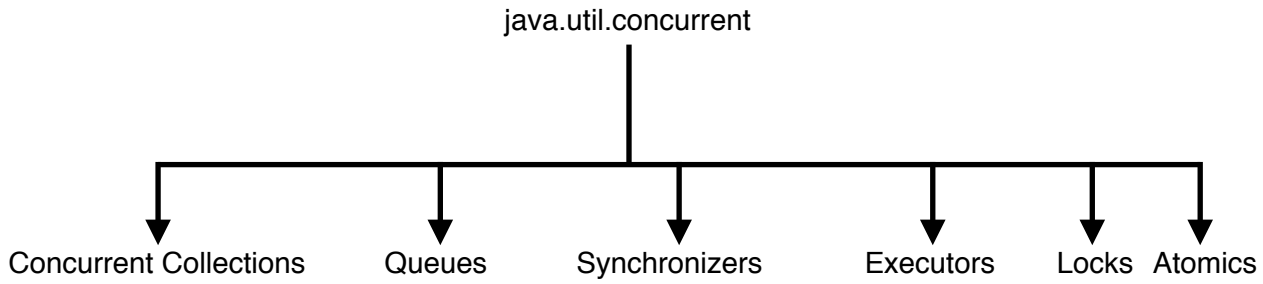
```
private int getState() {
    try {
        System.out.println("try");
        return 1;
    } finally {
        System.out.println("finally");
        return 2;
    }
}

private void printState() {
    System.out.println(getState());
}
```

```
/* Output:
try
finally
2
```

---

4. Классы синхронизированных коллекций пакета java.util.concurrent.

В версии Java 5 добавлен пакет java.util.concurrent, классы которого обеспечивают высокую производительность при построении потокобезопасных приложений.



Concurrent Collections - набор коллекций, которые работает намного эффективней в многопоточной среде нежели стандартные коллекции из java.util пакета. Вместо блокирования доступа ко всей коллекции используются блокировки по сегментам данных или же оптимизируется работа для параллельного чтения данных по wait-free алгоритмам. Некоторые из них:

ConcurrentHashMap	аналог Hashtable, данные представлены в виде сегментов, доступ блокируется по сегментам
ConcurrentLinkedQueue	аналог LinkedList
CopyOnWriteArrayList	аналог ArrayList
CopyOnWriteArraySet	имплементация интерфейса Set, за основу взят CopyOnWriteArrayList

ConcurrentSkipListMap	аналог TreeMap
ConcurrentSkipListSet	аналог TreeSet

Queues - неблокирующие и блокирующие очереди с поддержкой многопоточности: ConcurrentLinkedQueue, ConcurrentLinkedDeque, BlockingQueue, ArrayBlockingQueue, BlockingDeque и др.

Synchronizers - вспомогательные утилиты для синхронизации потоков:

Semaphore	предлагает потоку ожидать завершения действий в других потоках
Exchanger	обмен объектами между двумя потоками

Executors - содержит в себе фрейморки для создания пулов потоков, планирования работы асинхронных задач с получением результатов:

Future	интерфейс для получения результатов работы асинхронной операции
ExecutorService	интерфейс, который описывает сервис для запуска Runnable или Callable задач
Executor	организует запуск пула потоков и службы из планирования

Locks - представляет собой альтернативные и более гибкие механизмы синхронизации потоков по сравнению с базовыми synchronized, wait, notify, notifyAll: Lock, ReadWriteLock.

Atomics - классы с поддержкой атомарных операций над примитивами и ссылками.

---

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. Бинарный поиск, принцип и краткое описание алгоритма

Бинарный поиск – алгоритм поиска объекта по заданному признаку в множестве объектов, упорядоченных по тому же признаку.

Принцип.

На каждом шаге множество объектов делится на 2 части и в работе остается та часть, где находится искомый объект, деление множества на 2 части продолжается до тех пор, пока элемент не будет найден, либо пока не будет установлено, что его нет в заданном множестве.

Алгоритм бинарного поиска (на отсортированной по возрастанию структуре данных):

1. задаем границы: левая граница = 0вой индекс; правая граница = максимальный индекс структуры данных;
2. берем элемент посередине между границами, сравниваем его с искомым;
3. оцениваем результат сравнения:
   - если искомое равно элементу сравнения, возвращаем индекс элемента, на этом работа алгоритма заканчивается;
   - если искомое больше элемента сравнения, то сужаем область поиска таким образом: левая граница = индекс элемента сравнения + 1;
   - если искомое меньше элемента сравнения, то сужаем область поиска таким образом: правая граница = индекс элемента сравнения - 1;
4. Повторяем шаги 1-3 до тех пор, пока правая граница не станет меньше левой;
5. Возвращаем -1 (до этого шага доходим только в случае, если элемент не был найден);

Время выполнения алгоритма – O(log n);

---

6. Типы памяти в Java.

В связи с тем, что Java постоянно развивается, на данный момент существует ряд противоречий в терминологии относительно типов памяти в Java. Например, часть ресурсов термины non-heap и stack приравнивают друг к другу. В своем ответе я разделяю память в Java на три области: stack, heap, non-heap.

Stack (Стек):

- имеет небольшой размер в сравнении с кучей, размер стека ограничен, его переполнение может привести к возникновению исключительной ситуации;
- содержит только локальные переменные примитивных типов и ссылки на объекты в куче;
- стековая память может быть использована только одним потоком, т.е. каждый поток обладает своим стеком;
- работает по принципу LIFO(last-in-first-out), благодаря чему работает быстро;
- когда в методе объявляется новая переменная, она добавляется в стек, когда переменная пропадает из области видимости, она автоматически удаляется из стека, а эта область памяти становится доступной для других стековых переменных.

Heap (Куча):

- куча имеет больший размер памяти, чем стек (stack);
- используется для выделения памяти под объекты, объекты хранятся в куче;
- объекты кучи доступны разным потокам;
- в куче работает сборщик мусора: освобождает память, удаляя объекты, на которые нет ссылок.

Non-heap делится на 2 части:

- Permanent Generation – это пул памяти, который содержит интернированные строки, информацию о метаданных, классах, это пространство зарезервировано для классов, статических данных и т.п.
  Metaspace – замена Permanent Generation в Java 8. Основное различие в том, что Metaspace может динамически расширять свой размер во время выполнения. Размер Metaspace по умолчанию не ограничен.
- Code Cache – используется JVM при JIT-компиляции, здесь кэшируется скомпилированный код.