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. 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

---

3. 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
}
```

---

4. Метод join () класса Thread.

В Java существует механизм, который позволяет одному потоку ожидать завершения выполнения другого потока. Этот механизм "включается" при помощи метода join(). Вызывающий поток дожидается, пока указанный поток присоединится к нему. Для

того, чтобы заставить главный поток дождаться завершения работы побочного потока необходимо в главном потоке вызвать этот метод для побочного потока:

newThread.join();

Как только поток newThread закончит свою работу, он присоединится к главному потоку, и главный поток сможет продолжить выполнение. Если поток будет прерван при помощи метода interrupt(), будет выброшен InterruptedException.

Существуют дополнительные формы метода join(), которые позволяют указывать максимальный промежуток времени, в течение которого требуется ожидать завершения указанного потока исполнения:

public final void join(long millis) throws InterruptedException

public final void join(long millis, int nanos) throws InterruptedException

---

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. Radix Sort, принцип и краткое описание алгоритма.

Поразрядная сортировка может быть двух типов LSD (least significant digit) и MSD (most significant digit): отличие в том, с какого разряда начинается анализ, т.е. с меньшего или большего соответственно.

Принцип LSD сортировки:

Алгоритм представляет собой цикл по номеру разряда от младшего к старшему. На каждой итерации в результирующем массиве числа упорядочиваются по разряду с помощью какой-либо устойчивой сортировки. Чаще всего в качестве вспомогательной сортировки используется сортировка подсчетом (Counting Sort).

Алгоритм

1. Находится максимальный элемент среди элементов массива;
2. В цикле по номеру разряда сортируется результирующий массив по разряду с помощью сортировки подсчетом;

Сортировка подсчетом (Counting Sort)

(входной массив ar, вспомогательный массив count для счетчика, массив res для результата)

1. заполняем массив count нулями;
2. для каждого ar[i] увеличиваем count[a[i]] на 1;
3. подсчитываем количество элементов меньше или равных j-ому индексу для каждого элемента. Для этого каждый count[j], начиная с j = 1, увеличиваем на count[j - 1]. Таким образом, в последней ячейке будет находиться количество элементов равное количеству элементов в массиве.
4. В цикле (по всем элементам входного массива) входной массив читается с конца, значение count[ar[i]] уменьшается на 1 и в res[count[ar[i]]] записывается a[i]. Проход по массиву с конца обеспечивает устойчивость сортировки.

Свойства

1. время работы: лучшее – O(nk), среднее – O(nk), худшее – O(nk);
2. затраты памяти – O(n + k);
3. устойчивая;

Плюсы

- линейная сложность алгоритма;
- независимо от входных данных стабильное время работы;
- наиболее быстрая сортировка для массива, отсортированного в обратную сторону;
- хороша для сортировки строк;

Минусы

- нужно O(n + k) дополнительной памяти;
- по умолчанию не может обрабатывать отрицательные числа;
- по умолчанию не может обрабатывать числа с плавающей точкой;
- сложна в реализации;

---

4. JRE – определение, назначение

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.).