1. **Основні принципи ООП**
   - інкапсуляція
   - наслідування
   – поліморфізм

2. **Що таке наслідування**
   Наслідування - це процес завдяки якому один обєкт може придбати властивості іншого обєкта (успадкування всіх властивостей одного обєкта іншим) та додавати риси характерні лише для нього самого!!

   ```
   class Dog extends Animal
   {...}
   ```

   Суперклас -> Підклас
   Батьківський -> Дочірний

3. **Що таке поліморфізм**
   Поліморфізм - це властивість яка дозволяє одне і теж імя використовувати для вирішення декількох технічно різних задач. На практиці це значить спроможність об'єктів вибирати внутрішній метод або процедуру, виходячи із типу даних, прийнятих в повідомленні.

4. **Що таке інкапсуляція**
   Інкапсуляція - це механізм який поєднує дані та методи для обробки цих даних і захищає і те і інше від зовнішнього впливу або неправильного використання. В середині обєкта дані та методи можуть мати різні модифікатори доступу

   public - доступ для всіх
   private - тільки з даного класу
   protected - з даного класу і його нащадків
   Без модифікатора - для всіх класів даного пакету

5. **Расскажите про классы- загрузчики и про динамическую зарузку классов.**

   Любой класс, используемый в джава программу так или иначе был загружен в контекст программы каким-то загрузчиком.
   Все виртуальные машины джава включают хотябы один загрузчик классов, так называем базовый загрузчик. Он загружает все основные классы, это классы из rt.jar. Интересно то, что этот загрузчик никак не связан с программой, тоесть мы не можем получить например у java.lang.Object имя зарузчика, метод getClassLoader() вернет нам null.

   Следующий загрузчик – это загрузчик расширений, он загружает классы из $JAVA_HOME/lib/ext.

   Далее по иерархии идет системный загрузчик, он загружает классы, путь к которым указан в переменно класпаз.

   Для примера предположим что у нас есть некий пользовательский класс MyClass и мы его используем. Как идет его загрузка… :
   Сначала системный загрузчик пытается найти его в своем кэше загрузок его, если найден – класс успешно загружается, иначе управление загрузкой передается загрузчику расширений, он также проверяет свой кэш загрузок и в случае неудачи передает задачу базовому загрузчику. Тот проверяет кэш и в случае неудачи пытается его загрузить, если загрузка прошла успешно – загрузка закончена. Если нет – передает управление загрузчику расширений. Загрузчик расширений пытается загрузить класс и в случае неудачи передает

это дело системному загрузчику. Системный загрузчик пытается загрузить класс и в случае неудачи возбуждается исключение java.lang.ClassNotFoundException.
Вот так работает загрузка классов в джава. Так называемое делегирование загрузки.

Если в системе присутствуют пользовательские загрузики, то они должны быть унаследованы от класса java.lang.ClassLoader .

Что же такое статическая и что такое динамическая загрузка класса?

Статическая загрузка класса происходит при использовании оператора «new»

Динамическая загрузка происходит «на лету» в ходе выполнения программы с помощью статического метода класса Class.forName(имя класса). Для чего нужна динамическая загрузка? Например мы не знаем какой класс нам понадобится и принимаем решение в ходе выполнения программы передавая имя класса в статический метод forName().

6. **Для чего в джаве статические блоки?**

Статические блоки в джава выполняются до выполнения конструктора, с помощью них инициализируют статические поля к примеру.

```
static final int i;
static{

i=10;

}
```

Еще один ньюанс, блок статической инициализации может создаваться сам при компиляции программы:
Например public static int MAX = 100;

Будет создан код:
Public static int MAX;

```
static{
MAX = 100;
}
```

7. **Имеется выражение «является» и «имеет». Что они подразумевают в плане принципов ООП? В чем разница между композицией и агрегацией?**

«является» - наследование
«имеет» - композиция

В качестве примера предположим что у нас есть классы Строение, Дом и Ванная комната. Так вот Дом является строением, что нельзя сказать про Ванну, которая не является домом. А вот Дом имеет\включает в себя Ванну.

Если вы хотите использовать повторно код, то не обязательно использовать наследование.

Если нет отношения «является», то лучше тогда использовать композицию для повторного использования кода.

Не используйте наследование для получение полиморфизма, если нет ключевой зависимости «является». Используйте интерфейсы для полиморфизма.

Из спецификации можно узнать, что
Ассоциация обозначает связь между объектами.
Агрегация и композиция это частные случаи ассоциации.
Агрегация предполагает, что объекты связаны взаимоотношением "part-of" (часть).
Композиция более строгий вариант агрегации. Дополнительно к требованию part-of накладывается условие, что "часть" не может одновременно принадлежать разным "хозяевам", и заканчивает своё существование вместе с владельцем.

Например,
мотоцикл -> сумка с багажём - ассоциация. Отношение "имеет".
мотоцикл -> колесо - композиция.
группа по интересам -> человек - агрегация. человек часть группы, но может принадлежать нескольким разным группам.

8. **Что вы подразумеваете под полиморфизмом, инкапсуляцией и динамическим связыванием?**

Полиморфизм означает способность переменно данного типа, которая ссылается на объекты разных типов, при этом вызывается метод, характерный для конкретного типа ссылки на объект.
В чем преимущество полиморфизма? Он позволяет добавлять новые классы производных объектов, не нарушая при этом код вызова.
Также использование полиморфизма называют динамическим связыванием объектов.

Рассмотрим пример полиморфизма:
Имеется классы: Фигура, круг и треугольник.
Круг и треугольник наследуется от фигуры соответственно.
Каждый класс имеет метод «рисовать». В круге и треугольнике этот метод переопределен.
Так вот, создаем объект с типом «Фигура» и присваиваем ей ссылку на объект типа «Круг» и вызываем на этом объекте метод «рисовать». В итоге вызывается метод класса «Круг», а не класса «Фигура» как ожидалось.

```
Фигура ф = new Круг();
Ф.рисовать();
```

Также вместо класса родителя «Фигура» к примеру можно использовать интерфейс «Фигура», определив там метод рисовать. Этот интерфейс мы имплементируем в классах «Круг» и «Треугольник». Далее на интерфейсе создаем объект и присваиваем ему ссылку на объект какого-то из реализующих этот интерфейс классов.

Это удобно например если у нас есть некий метод:

```
public void drawShape(Фигура ф){
ф.рисовать();
}
```

Обратите внимание что в метод мы передаем параметр с типом интерфейса, т.е. мы не знаем какой именно тип объекта будем, но реализация будет таже. Далее мы можем просто создать еще класс, к примеру «Трапеция», имплементировать и имплементировать интерфейс «Фигура» и просто передать экземпляр класса в метод, ничего не меняя в реализации и вызове.

Наследование это включение поведения(методы) и состояния(поля) базового класса в производный от него.
В результате этого мы избегаем дублирования кода и процесс исправления ошибок в коде также упрощается.

В джава есть два вида наследования:
- наследование классов. Каждый наследник может иметь толко одного родителя.
-наследование интерфейсов. Интерфейс может иметь сколько угодно родителей.

Некоторые тонкие ньюансы по поводу наследования интрефейсов и классов:

Мы имеем два интерфейса с одинаковыми по имени полями. Имплементируем эти интерфейсы на каком-то классе.
Как нам вызвать поля этих интерфейсов? У нас неоднозначность.
Необходимо объект класса привести к нужному интерфейсу.

```
Класс к = new Класс();
((Интефейс1) к).поле;
```

Хорошо, что будет если мы имеем метод с одинаковой сигнатурой в интерфейсах и реализуем эти интерфейсы на классе. Как нам в классе реализовать два метода с одинаковой сигнатурой???
Ответ: никак, мы просто реализовываем один общий метод в классе.
Это является недостатком, так как нам может потребоваться разная реализация.

И третий случай: У нас есть класс и интерфейс с одинаковым по сигнатуре методом. Мы наследуемся от этого класса и имплементируем этот интерфейс. Что нам нужно делать? ведь необходимо реализовать метод интерфейса по всем правилам.
И вот тут интересно, компилятор не выдает ошибок, так как метод уже у нас реализован в классе родителе.

Инкапсуляция - это свойство которое позволяет закрыть доступ к полям и методам класса другим классам, а предоставлять им доступ только через интерфейс(метод).

9. В чем разница между абстрактным классом и интерфейсом? И когда их нужно использовать?

Абстрактный класс это класс, который помечен как «abstract», он может содержать абстрактные методы, а может их и не содержать.
Экземпляр абстрактного класса нельзя создать.
Класс, который наследуется от абстрактного класса может реализовывать абстрактные методы, а может и не реализовывать, тогда класс наследник должен быть тоже абстрактным.
Также если класс наследник переопределяет реализованный в абстрактном классе родители

метод, его можно переопределить с модификатором абстракт! Т.е отказаться от реализации. Соответственно данный класс должен быть также абстрактным также.

Что касается интерфейса, то в нем находятся только абстрактные методы и константы. При реализации интерфейса, класс обязан реализовать все методы интерфейса. Иначе класс должен быть помечен как абстрактный.
Интересный момент, интерфейс может содержать внутренние классы! И не абстрактные методы в них.

Что же использовать Интерфейс или Абстрактный класс?
Абстрактный класс используется когда нам нужна какая-то реализация по умолчанию. Интерфейс используется когда классу нужно указать конкретное поведение. Часто интерфейс и абстрактный класс комбинируют, т.е. имплементируют интерфейс в абстрактном классе, чтоб указать поведение и реализацию по умолчанию. Это хорошо видно на примере свига:

```
AbstractTableModel implements TableModel{
}

MyTableModel extends AbstractTableModel{
}
```

Мы создаем свою модель таблицы с определенным поведением и уже с реализацией по умолчанию.

10. Почему в некоторых интерфейсах вообще не определяют методов?

Это так называемые интерфейсы –маркеры. Они просто указывают что класс относится к определенной группе классов. Например интерфейс Clonable указывает на то, что класс поддерживает механизм клонирования.
Степень абстракции в данном случае доведен до абсолюта. В интерфейсе вообще нет никаких объявлений.

11. Расскажите про Collection framework?

В основе фреймворка лежит два интерфейса, это «Collection» и «Map».
Collection расширяет три интерфейса: List, Set, Queue.

List – хранит упорядоченные елементы(могут быть одинаковые);
Имеет такие реализации как LiskedList, ArrayList и Vector.
Vector синхронизирован, и по этому в одном потоке, он работает медленней остальных реализаций.
ArrayList – его преимущество в навигации по коллекции.
LinkedList – Его преимущество в во вставке и удалении элементов в коллекции.

Set – коллекции, которые не содержат повторяющихся елементов.
Основные реализации: HashSet, TreeSet, LinkedHashSet
TreeSet – упорядочивает элементы по их значениям;
HashSet – упорядочивает элементы по их хэш ключах, хотя на первый взляд может показаться что элементы хранятся в случайном порядке.
LinkedHashSet – хранит элементы в порядке их добавления

Queue – интерфейс для реализации очереди в джава.

Основные реализации: LinkedList, PriorityQueue.
Очереди работают по принципу FIFO – first in first out.

Map – интерфейс для реализации так называемой карты, где елементы хранятся с ихними ключами.
Основные реализации: HashTable, HashMap, TreeMap, LinkedHashMap
HashTable – синхронизированна, объявлена уставревшей.
HashMap – порядок елементов рассчитывается по хэш ключу;
TreeMap – элементы хранятся в отсортированном порядке
LinkedHashMap – элементы хранятся в порядке вставки
Ключи в Map не могут быть одинаковыми!

Синхронизировать не синхронизированные коллекции и карты можно посредством класса Collections.synchronizedMap(MyMap)\ synchronizedList(MyList)

12. Как сравниваются елементы коллекций?

Для сравнения элементов коллекций используется метод equals() и hashcode();
Эти методы унаследованы от класса Object.
Если наш пользовательский класс переопределяет equals(), то он должен и переопределить hashcode()
Если два объекта эквивалентны, то и хэш коды этих объектов тоже должны быть равны
Если поле не используется в equals(), то оно и не должно использоваться в hashcode().

13. Какая основная разница между String, StringBuffer, StringBuilder?

String – неизменяемый класс, тоесть для для добавление данных в уже существующую строку, создается новый объект строки.

StringBuffer и StringBuilder могут изменятся и добавление строки не такое дорогостоющее с точки зрения памяти. Первы – синхронизированный, второй – нет. Это их единственное различие.

Правда если нам нужно сделать подстроку строки, то лучше использовать String, так как ее массив символов не меняется и не создается заново для новой строки. А вот в StringBuffer и StringBuilder для создания подстроки создается новый массив символов.

14. Какова основная разница между передачей по ссылке и по значению?

В java параметры в методы передаются по значению, тоесть создаются копии параметров и с ними ведется работа в методе. В случае с примитивными типами, то при передачи параметра сама переменная не будет меняться так как в метод просто копируется ее значение.
А вот при передачи объекта копируется ссылка на объект, тоесть если в методе мы поменяем состояние объекта, то и за методом состояние объекта тоже поменяется. Но если мы этой копии ссылки ссылки попытаемся присвоить новую ссылку на обьект, то старая ссылка у нас не изменится!

В случае же кода мы передаем String, то состояние не будет менятся, так как String неизменяемый.

15. Что такое сериализация? Как Вы исключите поля из сериализации? transient что значит?

Сериализация – это процес чтения или записи объекта. Это процесс сохранения состояния объекта и считывание этого состояния.
Для реализации сериализации нужен интерфейс – маркер Serializable

Для того чтоб исключить поля из сериализуемого потока, необходимо пометить поле модификатором transient

16. Расскажите про потоки ввода-вывода Java.

Потоки ввода-вывода бывают двух видов:
- байтовый поток(InputStream и OutputStream);
- символный поток(Reader и Writer);

Это все абстрактные классы – декораторы, которым можно добавлять дополнительный функционал, например:
InputStream in = new FileInputStream(new File("file.txt"));

17. Расскажите про клонирование объектов. В чем отличие между поверхностным и глубоким клонированием?

Чтобы объект можно было клонировать, он должен реализовать интерфейс Cloneable(маркер). Использование этого интерфейса влияет на поведение метода «clone» класс Object. Таким образом
myObj.clone() создаст нам клон нашего объекта, но этот клон будет поверхностный.

Что значит поверхностным? Это значит что клонируется только примитивные поля класса, ссылочные поля не клонируются!

Для того, чтоб произвести глубокое клонирование, необходимо в клонируемом классе переопределить метод clone() и в нем произвести клонирование изменяемых полей объекта.

18. В чем разница между переменно экземпляра и статической переменной? Приведите пример.

Статические переменные инициализируются при загрузке класса класслодером, и не зависят от объекта. Переменная экземплра инициализируется при создании класса.

Пример: Например нам нужна глобальная переменная для всех объектов класс, например число посещений пользователей определенной статьи в интернете. При каждом новом посещении статьи создается новый объект и инкрементируется переменная поещений.

19. Что такое final модификатор? Объясните другие модификаторы.

Нельзя наследоваться от файнал класса. Нельзя переопределить файнал метод. Нельзя изменить сначение файнал поля.

volatile – указывает на то, что поле синхронизировано для нескольких потоков
synchronized – указывает на то что метод синхронизированный или же в методе может находится такой блок синхронизации.
transient – указывает на то, что переменная не подлежит сериализации
native – говорит о том, что реализация метода написана на другой программной платформе

20. Что такое final, finally и finalize()?

final – Нельзя наследоваться от файнал класса. Нельзя переопределить файнал метод. Нельзя изменить сначение файнал поля.
finally – используется при обработке ошибок, вызывается всегда, даже если произошла ошибка(кроме System.exit(0)). Удобно использовать для освобождения ресурсов.
finalize() – вызывается перед тем как сборщик мусора будет проводить освобождение памяти. Не рекомендуется использовать для освобождения системных ресурсов, так как не известно когда сборщик мусора будет производить свою очистку. Вообще данный метод мало кто использует. Единственно что можно использовать этот метод для закрытия ресурса что должен работать на протяжении всей работы программы и закрываться по ее окончанию. Еще можно использовать метод для защиты от так называемых «дураков», проверять, освобождены ли ресурсы, если нет, то закрыть их.

21. Расскажите про модель памяти в джава?

В Джаве память устроена следующим образом, есть два вида:
- куча
- стек

В стеке происходит весь процесс выполнения программы, а также хранятся примитивные типы полей и переменных.

Куча состоит из статического контекста и самой кучи
В статическом контексте находятся загружаемые классы.
Перейдем к куче. Куча состоит из двух частей:
-Так называемая новая куча
- старая куча

Новая куча в свою очередь состоит из двух частей:
-Eden(назовем ее первая) куча
- Survival(выжившая) куча

Итак, как работает гебедж коллектор?
Во-первых что стоит сказать… у сборщика мусора есть несколько алгоритмов работы, он не один.

Когда происходит очистка памяти? Если память в Первой куче полностью заполнена, то туда идет сборщик мусора и делает свою работу) Какую именно, зависит от обстоятельств… Например если в первой кучи много мусора(т.е. объектов с нулевой ссылкой), то сборщик мусора помечает эти объекты, далее те что остались объекты со ссылками он их переносит в Выжившую кучу, а в первой куче он просто все удаляет.
Ситуация другая… в первой кучи мало мусора, но очень много рабочих объектов. Как поступает в этом случае сборщик мусора?

Он помечает мусор, удаляет его и оставшиеся объекты компонует.

Также следует заметить что при нехватке места в Выжившей куче, объекты переносятся в старую кучу, там хранятся как правило долго живущие объекты.

Также следует заметить что сборщик мусора вызывается сам периодически, а не только когда памяти не хватает.

22. Расскажите про внутренние классы. Когда вы их будете использовать?

Внутренний класс – это класс, который находится внутри класса или интерфейса. При этом он получает доступ ко всем полям и методам своего внешнего класса.
Для чего он может применятся? Например чтоб обеспечить какую-то дополнительную логику класса. Хотя использование внутренних классов усложняет программу, рекомендуется избегать их использование.

Какие внутренние классы бывают?
Есть так называемый вложенный (с модификатором static)
А также есть анонивный внутренний класс.

23. Расскажите про привидение типов. Что такое понижение и повышение типа? Когда вы получаете ClassCastException?

Приведение типов это установка типа переменной или объекта отличного от текущего. В ждава есть два вида приведения:
- автоматическое
- не автоматическое

Автоматическое происходит например:

byte-> short->int->long->float->double

тоесть если мы расширяем тип, то явное преобразование не требуется, приведение происходит автоматически. Если же мы сужаем, то необходимо явно указывать привидение типа.

В случае же с объектами, то мы можем сделать автоматическое приведение от наследника к родителю, но никак не наоборот, тогда вылетит ClassCastException

24. Чем отличается процесс от потока?

Процесс - это объект,которому операционная система выделяет уникальное адресное пространство. Это происходит при запуске прораммы.Также создается поток в этом процессе. А вернее группа потоков маин.

25. Какие способы запуска потоков вы знаете?

- Наследоваться от класса Thread, создать объект и вызвать метод старт.
- Имплементировать интерфейс Runnable, создать объект класса Thread ив качестве параметра конструктору передать объект нашего класса.

26. В каких состояниях может быть поток в джава?

Как вообще работает поток?
У нас ест текущий поток, в котором выполняется метод main. Этот поток имеет свой стек и этот стек начинается с вызова метода main.
Далее в методе main мы создаем новый поток, что происходит… создается новый поток и для него выделяется свой стек с первоначальным методом run().
Когда мы запускаем несколько потоков, то мы не можем гарантировать определенный порядок их вызовов. Планированием потоков занимается планировщик потоков JVM, выбирая из пулов потоков поток. Мы даже не можем гарантировать что если первый поток начался выполнятся первым, то он и закончит выполнятся первым, он может закончить выполнятся последним.
Еще такой ньюанс, что поток, который закончил свое выполнение, не может быть повторно запущен! Он находится в состоянии «мертвый», а для запуска потока нового потока, объект должен находится в состоянии «новый».

Потоки имеют такие состояния:
-новый(это когда только создали экземпляр класса Thread)
-живой или работоспособный(переходит в это состояние после запуска метода start(), но это не означает что поток уже работает! Или же он может перейти в это состояние из состояние работающий или блокированный)
- работающий(это когда метод run() начал выполнятся)
- Ожидающий (waiting)/Заблокированный (blocked)/Спящий(sleeping). Эти состояния характеризуют поток как не готовый к работе. Я объединил эти состояния т.к. все они имеют общую черту – поток еще жив (alive), но в настоящее время не может быть выполнен. Другими словами поток уже не работает, но он может вернутся в рабочее состояние.
Поток может быть заблокирован, это может означать что он ждет освобождение каких-то ресурсов. Поток может спать, если встретился метод sleep(long s) , или же он может ожидать, если встретился метод wait(), он будет ждать пока не вызовится метод notify() или notifyall().

- мертвы(состояние когда метод run() завершил свою работу)

27. Что значит усыпить поток?

Перевести поток в спящее состояние можно с помощью метода sleep(long ms) ms – время в миллисекундах.
При вызове этого метода, поток переходит в спящее состояние, после сна, поток переходит в пул потоков и находится в состоянии «работоспособный», т.е. не гарантируется что после пробуждения он будет сразу выполняться. Также поток не может усыпить другой поток, так как метод sleep – это статический метод! Вы просто усыпите текущий поток и не более того!
Также метод sleep() может возбуждать InterruptedException().

28. Что значит приоритет потока?

Приоритет потока – это число от 1 до 10, в зависимости от которого, планировщик потоков выбирает какой поток запускать. Однако полагаться на приоритеты для предсказуемого выполнения многопоточной программы нельзя!

29. Что делает метод yield()?

Метод yield() пытается сказать планировщику потоков, что нужно выполнить другой поток, что ожидает в очереди на выполнение. Метод не пытается перевести текущий поток в состояние блокировки, сна или ожидания. Он просто пытается его перевести из состояние «работающий» в состояние «работоспособный». Однако выполнение метода может вообще не произвести никакого эффекта.

30. Расскажите про метод join()?

Метод join() вызывается для того, чтобы привязать текущий поток в конец потока для которого вызывается метод. Тоесть второй поток будет в режиме блокировки пока первый поток не выполнится.

31. Расскажите про синхронизацию потоков.

Представьте себе ситуацию что два потока одновременно изменяют состояние какого-то объекта, это недопустимо. Для этого необходимо синхронизировать потоки. Как это сделать? Ключевое слово synchronized позволяет это сделать установив в сигнатуре метода. Или же в методе можно описать блок synchronized, только в качестве параметра необходимо передать объект, который будет блокироватся.

Представьте себе ситуацию когда один поток ждет пока разблокируется объект… а если это ждут несколько потоков? Нет гарантии что тот объект что больше всех ждал снятия блокировки будет выполнятся первым.

Статические синхронизированные методы и нестатические синхронизированные методы не будет блокировать друг друга, никогда. Статические методы блокируются на экземпляре класса Class в то время как нестатические методы блокируются на текущем экземпляре (this). Эти действия не мешают друг другу.

wait() – отказывается от блокировки
остальные методы сохраняют блокировку

32. Что такое взаимная блокировка ?

Это когда один поток А получил блокировку на объект А1, а поток В получил блокировку на объект В1. В то время как поток А пытается получит блокировку на объект В1, а поток В на А1.

33. Что такое потоки –демоны в джава?

Это потоки, которое работают в фоновом режиме и не гарантируют что они завершатся. Тоеть если все потоки завершились, то поток демон просто обрывается вместе с закрытием приложения.

34. Может ли объект получить доступ к private-переменной класса? Если, да, то каким образом?

Вообще доступ у приватной переменной класса можно получить только внутри класса, в котором она объявлена. Для доступа из других классов можно применить инкапсуляцию.

Также доступ к приватным переменным можно осуществить через механизм отражений.

35. Что такое класс Object? Какие в нем есть методы?

Object это базовый класс для всех остальных объектов в Java. Каждый класс наследуется от Object. Соответственно все классы наследуют методы класса Object.

Методы класса Object:

public final native Class getClass() -Этот метод возвращает объект класса Class, который описывает класс(имяб методы, поля), от которого был порожден этот объект.

public native int hashCode()-данный метод возвращает значение int. Цель hashCode()– представить любой объект целым числом.Конечно, нельзя потребовать, чтобы различные объекты возвращали различные хэш-коды, но, по крайней мере, необходимо, чтобы объекты, равные по значению (метод equals()возвращает true), возвращали одинаковые хэш-коды.

public boolean equals(Object obj)- equals()служит для сравнения объектов по значению, а не по ссылке. Сравнивается состояние объекта, у которого вызывается этот метод, с передаваемым аргументом.

```
Point p1=new Point(2,3);
Point p2=new Point(2,3);
print(p1.equals(p2));
```

Результатом будет false.

protected native Object clone() throws CloneNotSupportedException

При выполнении метода clone()сначала проверяется, можно ли клонировать исходный объект. Если разработчик хочет сделать объекты своего класса доступными для клонирования через Object.clone(), то он должен реализовать в своем классе интерфейс Cloneable. В этом интерфейсе нет ни одного элемента, он служит лишь признаком для виртуальной машины, что объекты могут быть клонированы. Если проверка не выполняется успешно, метод порождает ошибку CloneNotSupportedException.

Если интерфейс Cloneable реализован, то порождается новый объект от того же класса, от которого был создан исходный объект. При этом копирование выполняется на уровне виртуальной машины, никакие конструкторы не вызываются. Затем значения всех полей, объявленных, унаследованных либо объявленных в родительских классах, копируются. Полученный объект возвращается в качестве клона.

Обратите внимание, что сам класс Objectне реализует интерфейс Cloneable, а потому попытка вызова new Object().clone()будет приводить к ошибке. Метод clone()предназначен скорее для использования в наследниках, которые могут обращаться к нему с помощью выражения super.clone().Примитивные поля копируються и далее существуют независимо в исходном и клонированном объектах. Изменение одного не сказывается на другом.

А вот ссылочные поля копируются по ссылке, оба объекта ссылаются на одну и ту же область памяти(исходный объект) . Поэтому изменения, происходящие с исходным объектом, сказываются на клонированном.

public String toString()-Этот метод позволяет получить текстовое описание любого объекта. Создавая новый класс, данный метод можно переопределить и возвращать более подробное описание. Для класса Object и его наследников, не переопределивших toString(), метод возвращает следующее выражение:

```
getClass().getName()+"@"+hashCode()
```

Метод getName()класса Class уже приводился в пример, а хэш-код еще дополнительно обрабатывается специальной функцией для представления в шестнадцатеричном формате. Например:

```
print(new Object());
```

Результатом будет:

java.lang.Object@92d342

В результате этот метод позволяет по текстовому описанию понять, от какого класса был порожден объект и, благодаря хеш-коду, различать разные объекты, созданные от одного класса.

Именно этот метод вызывается при конвертации объекта в текст, когда он передается в качестве аргумента оператору конкатенации строк.

Методы wait(), notify(), notifyAll()используются для поддержки многопоточности

- public final native void notify()
- public final native void notifyAll()

public final native void wait(long timeout) throws InterruptedException
public final void wait(long timeout, int nanos) throws InterruptedException
public final void wait() throws InterruptedException

protected void finalize() throws Throwable-данный метод вызывается при уничтожении объекта автоматическим сборщиком мусора (garbage collector). В классе Object он ничего не делает, однако в классе-наследнике позволяет описать все действия, необходимые для корректного удаления объекта, такие как закрытие соединений с БД, сетевых соединений, снятие блокировок на файлы и т.д. В обычном режиме напрямую этот метод вызывать не нужно, он отработает автоматически. Если необходимо, можно обратиться к нему явным образом.

В методе finalize() нужно описывать только дополнительные действия, связанные с логикой работы программы. Все необходимое для удаления объекта JVM сделает сама.

36. Что такое метод equals(). Чем он отличается от операции ==.

Метод equals() обозначает отношение эквивалентности объектов. Эквивалентным называется отношение, которое является симметричным, транзитивным и рефлексивным.

- Рефлексивность: для любого ненулевого x, x.equals(x) вернет true;
- Транзитивность: для любого ненулевого x, y и z, если x.equals(y) и y.equals(z) вернет true, тогда и x.equals(z) вернет true;
- Симметричность: для любого ненулевого x и y, x.equals(y) должно вернуть true, тогда и только тогда, когда y.equals(x) вернет true.

Также для любого ненулевого x, x.equals(null) должно вернуть false.

Отличия equals() от операции == в классе Object нет. Это видно, если взглянуть исходный код метода equals класса Object:

```
public boolean equals(Object obj) {
return (this == obj);
}
```

Однако, нужно не забывать, что, если объект ни на что не ссылается(null), то вызов метода equals этого объекта приведет к NullPointerException. Также нужно помнить, что при сравнении объектов оба они могут быть null и операция obj1 == obj2 в данном случае будет true, а вызов equals приведет к исключению NullPointerException.

Как мы видим, при помощи операции == сравниваются ссылки на объекты. Но мы можем переопределять метод equals, тем самым задавая логику сравнения двух объектов. Например, рассмотрим сравнение двух одинаковых чисел, созданных при помощи класса Integer:

```
Integer a = new Integer(6);
Integer b = new Integer(6);
System.out.println(a == b); // false т.к. это разные объекты с разными ссылками
System.out.println(a.equals(b)); // true, здесь уже задействована логика сравнения
```

Если взглянуть внутрь метода equals класса Integer, то мы увидим:

```
public boolean equals(Object obj) {
if (obj instanceof Integer) {
return value == ((Integer)obj).intValue();
```

}
return false;
}
Понятно, что тут уже нет сравнения ссылок, а сравниваются int значения.
Итого: применяйте “==”для примитивов, equals- для объектов
Изначально смысл этих методов сравнения одинаков(сравнение по значению, в случае объектов сравниваются ссылки на объекты)Но equals всегда можно переопределить согласно логике.

37. Если вы хотите переопределить equals(), какие условия должны удовлетворяться для переопределенного метода?

Метод equals() обозначает отношение эквивалентности объектов. Эквивалентным называется отношение, которое является симметричным, транзитивным и рефлексивным.

- Рефлексивность: для любого ненулевого x, x.equals(x) вернет true;
- Транзитивность: для любого ненулевого x, y и z, если x.equals(y) и y.eqals(z) вернет true, тогда и x.equals(z) вернет true;
- Симметричность: для любого ненулевого x и y, x.equals(y) должно вернуть true, тогда и только тогда, когда y.equals(x) вернет true.

Также для любого ненулевого x, x.equals(null) должно вернуть false.

38. Для чего нужен метод hashCode()?

Существуют коллекции(HashMap, HashSet), которые используют хэш код, как основу при работе с объектами. А если хэш для равных объектов будет разным, то в HashMap будут два равных значения, что является ошибкой. Поэтому необходимо соответствующим образом переопределить метод hashCode().
X Хеширование— преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями или функциями свёртки, а их результаты называют хешем или хеш-кодом.
Хе Хеш-таблице— это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.
Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i = hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i].
Одним из методов построения хеш-функции есть метод деления с остатком (division method) состоит в том, что ключу k ставится в соответствие остаток от деления k на m, где m— число возможных хеш-значений:

39. Какая связь между hashCode и equals?
Объекты равны, когда a.equals(b)=true и a.hashCode==b.hashcode ->true Но необязательно, чтобы два различных объекта возвращали различные хэш коды(такая ситуация называется коллизией).

40. Каким образом реализованы методы hashCode и equals в классе Object?
Реализация метода equals в классе Object сводится к проверке на равенство двух ссылок:

```
public boolean equals(Object obj) {
  return (this == obj);
```

}
Реализация же метода hashCode класса Object сделана нативной, т.е. определенной не с помощью Java-кода:
public native int hashCode();
Он обычно возвращает адрес объекта в памяти.

41. Какие есть оссобенности класса String? что делает метод intern()

1) Внутреннее состояние класса String нельзя изменить после его создания, т.е. этот класс неизменяемый (immutable) поэтому когда вы пишете String str = "One" + "Two"; создается три! объекта класса String.
2) От него нельзя унаследоваться, потому что класс String объявлен как final: public final class String
3) Метод hashCode класса String переписан и возвращает: s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
4) У класса String есть метод public String intern(), который возвращает строку в каноническом ее представлении из внутреннего пула строк, поддерживаемого JVM.
он нужен чтобы вместо String.equals() использовать ==.
Понятно, что оператор сравнения ссылок выполняется гораздо быстрее, чем посимвольное сравнение строк.
Используют в основном, где приходится сравнивать много строк, например в каких нибудь XML парсерах.
А вообще по увеличению производительности ещё вопрос. Ибо метод intern() тогда должен выполняться быстрее чем equals().
каждый раз когда вы вызываете метод intern() просматривается пул строк на наличие такой строки и если такая уже есть в пуле, то возвращается ссылка на нее.Сравниваются они через equal().

42. Что будет, если переопределить equals не переопределяя hashCode? Какие могут возникнуть проблемы?

Они будут неправильно хранится в контейнерах, использующих хэш коды, таких как HashMap, HashSet.
Например HashSet хранит элементы в случайном (на первый взгляд) порядке. Дело в том, что для быстрого поиска HashSet расчитывает для каждлого элемента hashCode и именно по этому ключу ищет и упорядочивает элементы внутри себя

43. Есть ли какие-либо рекомендации о том, какие поля следует использовать при подсчете hashCode?

Есть. Необходимо использовать уникальные, лучше примитивные поля, такие как id, uuid, например. Причем, если эти поля задействованы при вычислении hashCode, то нужно их задействовать при выполнении equals.
Общий совет: выбирать поля, которые с большой долью вероятности будут различаться.

44. Можно ли перегрузить static метод?

Статические методы могут перегружаться нестатическими и наоборот – без ограничений.
А вот в переопределении статического метода смысла нет

45. Какие типы классов бывают в java (вложенные… и.т.д.)

Классы могут быть внутренними-определение одного класса содержится внутри другого.

```
public class Parcel1 {
  class Contents {
```

```
  private int i = 11;
  public int value() { return i; }
 }
 class Destination {
  private String label;
  Destination(String whereTo) {
   label = whereTo;
  }
  String readLabel() { return label; }
 }
```

Вложенный класс- это статический внутренний класс(ссылка на объект внешнего класса не нужна)

Безымянный внутренний клас

```
//: innerclasses/Parcel7.java
// Returning an instance of an anonymous inner class.

public class Parcel7 {
 public Contents contents() {
  return new Contents() { // Insert a class definition
   private int i = 11;
   public int value() { return i; }
  }; // Semicolon required in this case
 }
 public static void main(String[] args) {
  Parcel7 p = new Parcel7();
  Contents c = p.contents();
 }
} ///:~
```

Это объявление эквивалентно:

The next example looks a little odd:

```
/: innerclasses/Parcel7b.java
// Expanded version of Parcel7.java

public class Parcel7b {
 class MyContents implements Contents {
  private int i = 11;
  public int value() { return i; }
 }
 public Contents contents() { return new MyContents(); }
 public static void main(String[] args) {
  Parcel7b p = new Parcel7b();
  Contents c = p.contents();
 }
} ///:~
```

Локальный внутренний клас- внутренний класс расположенный внутри блока кода или метода

46. Что такое статический класс, какие особенности его использования?

Статическим классом может быть только внутренний клас(определение класса размещается внутри другого класса). В объекте обычного внутреннего класса хранится ссылка на объект внешнего класса. Внутри статического внутреннего класса такой ссылки нет.
То есть:
Для создания объекта статического внутреннего класса не нужен объект внешнего класса.
Из объекта статического вложенного класса нельзя обращаться к нестатическим членам внешнего класса напрямую
И еще обычные внутренние классы не могут содержать статические методы и члены.

Зачем вообще нужны внутренние классы? –Каждый внутренний класс способен независимо наследовать определенную реализ-ю.Таким образом внутренний класс не ограничен при наследовании в ситуациях, когда внешний класс уже наследует реализацию. То есть это как бы вариант решения проблемы множественного наследования.

47. Каким образом из вложенного класса получить доступ к полю внешнего класса.

Если класс внутренний то: Внешнийкласс.this.Поле внешнего класса
Если класс статический внутренний(вложенный),то в методе нужно создать объект внешнего класса, и получить доступ к его полю.Или второй вариант объявить это поле внешнего класса как static

48. Какие виды исключений в Java вы знаете, чем они отличаются?

Все исключительные ситуации можно разделить на две категории: проверяемые(checked) и непроверяемые(unchecked).
Все исключения, порождаемые от Throwable, можно разбить на три группы. Они определяются тремя базовыми типами: наследниками Throwable- классами Errorи Exception, а также наследником Exception - RuntimeException.
Ошибки, порожденные от Exception (и не являющиеся наследниками RuntimeException), являются проверяемыми. Т.е. во время компиляции проверяется, предусмотрена ли обработка возможных исключительных ситуаций. Как правило, это ошибки, связанные с окружением программы (сетевым, файловым вводом-выводом и др.), которые могут возникнуть вне зависимости от того, корректно написан код или нет. Например, открытие сетевого соединения или файла может привести к возникновению ошибки и компилятор требует от программиста предусмотреть некие действия для обработки возможных проблем. Таким образом повышается надежность программы, ее устойчивость при возможных сбоях.
Исключения, порожденные от RuntimeException, являются непроверяемыми и компилятор не требует обязательной их обработки.
Как правило, это ошибки программы, которые при правильном кодировании возникать не должны (например, IndexOutOfBoundsException- выход за границы массива, java.lang.ArithmeticException- деление на ноль). Поэтому, чтобы не загромождать программу, компилятор оставляет на усмотрение программиста обработку таких исключений с помощью блоков try-catch.
Исключения, порожденные от Error, также не являются проверяемыми. Они предназначены для того, чтобы уведомить приложение о возникновении фатальной ситуации, которую программным способом устранить практически невозможно (хотя формально обработчик допускается). Они могут свидетельствовать об ошибках программы, но, как правило, это неустранимые проблемы на уровне JVM. В качестве примера можно привести StackOverflowError (переполнение стека), OutOfMemoryError (нехватка памяти).
Методы, код которых может порождать проверяемые исключения, должны либо сами их обрабатывать, либо в заголовке метода должно быть указано ключевое слово throws с

перечислением необрабатываемых проверяемых исключений. На непроверяемые ошибки это правило не распространяется.
Переопределенный (overridden) метод не может расширять список возможных исключений исходного метода.

49. Возможно ли использование блока try-finally (без catch)?

try может быть в паре с finally, без catch. Работает это точно так же – после выхода из блока try выполняется блок finally. Это может быть полезно, например, в следующей ситуации. При выходе из метода вам надо произвести какое-либо действие. А return в этом методе стоит в нескольких местах. Писать одинаковый код перед каждым return нецелесообразно. Гораздо проще и эффективнее поместить основной код в try, а код, выполняемый при выходе – в finally.

50. Всегда ли исполняется блок finally?

1.Существуют потоки- Демоны- потоки предоставляющие некие сервисы, работая в фоновом режиме во время выполнения программы,но при этом не являются ее неотъемлеммой частью.Таким образом когда все потоки не демоны завершаются, программа завершает свою работу. В потоках демонах блок finally не выполняеться, они прерываются внезапно.

2.System.exit(0)
3.если в блоке finally произошло исключение и нет обработчика, то оставшийся код в блоке finally может не выполнятся