1. Опишите процедуру инициализации полей класса и полей экземпляра класса. Когда инициализируются поля класса, а когда – поля экземпляров класса. Какие значения присваиваются полям по умолчанию? Где еще в классе полям могут быть присвоены начальные значения?

Как известно, в Java поля (fields) могут принадлежать классу или объекту. Поля, принадлежащие классу, являются статическими, а поля, принадлежащие объекту, – нестатическими. Статические поля доступны без создания объекта класса. Соответственно инициализироваться статические и нестатические поля должны в разное время: одни до создания объекта класса, а другие после.

Существуют следующие методы инициализации полей:

– Инициализация в месте объявления поля Поля класса, поля объекта. Применяется, если инициализация может быть произведена коротким выражением и доступен контекст, необходимый для ее проведения.

– Инициализационный блок Поля класса, поля объекта. Применяется, если инициализационный код неудобно записывать одним выражением или же, например, нужна обработка проверяемых исключений. В случае объектов может применяться для инициализации полей объектов анонимных классов.

– Конструктор класса Поля объекта. Применяется, если для инициализации нужны параметры конструктора. Значения по умолчанию:

int => 0
boolean => false
double => 0.0
float => 0.0
char => ' ' - нуль-символ
long => 0
byte => 0
reference => null

2. Дайте определение перегрузке методов. Как вы думаете, чем удобна перегрузка методов? Укажите, какие методы могут перегружаться, и какими методами они могут быть перегружены? Можно ли перегрузить методы в базовом и производном классах? Можно ли private метод базового класса перегрузить public методов производного? Можно ли перегрузить конструкторы, и можно ли при перегрузке конструкторов менять атрибуты доступа у конструкторов?

В программе мы можем использовать методы с одним и тем же именем, но с разными типами и/или количеством параметров. Такой механизм называется перегрузкой методов (method overloading).

Перегрузка методов – это создание внутри одного класса методов с одинаковыми именами, но разными входными параметрами. Удобство в отсутствии необходимости придумывать новые имена методам, выполняющих схожие задачи. Любые методы могут быть перегружены. Методы с разными агрументами в базовом и производном классах не являются перегрузкой. Private метод базового класса нельзя перегрузить public методом производного. Конструкторы можно перегружать, также можно при перегрузке конструкторов менять атрибуты доступа у конструкторов.

3. Объясните, что такое раннее и позднее связывание? Перегрузка – это раннее или позднее связывание? Объясните правила, которым следует компилятор при разрешении перегрузки; в том числе, если методы перегружаются примитивными типами, между которыми возможно неявное приведение или ссылочными типами, состоящими в иерархической связи.

Итак, фундаментальное различие между статическим и динамическим связыванием в Java состоит в том, что первое происходит рано, во время компиляции на основе типа ссылочной переменной, а второе – позднее, во время выполнения, с использованием конкретных объектов.

Статическое связывание происходит во время компиляции, а динамическое – во время выполнения. Поскольку статическое связывание происходит на ранней стадии жизненного цикла программы, его называют ранним связыванием. Аналогично, динамическое связывание называют также поздним связыванием, поскольку оно происходит позже, во время работы программы.

Статическое связывание используется в языке Java для разрешения перегруженных методов, в то время как динамическое связывание используется в языке Java для разрешения переопределенных методов.

Аналогично, приватные, статические и терминальные методы разрешаются при помощи статического связывания, поскольку их нельзя переопределять, а все виртуальные методы разрешаются при помощи динамического связывания. В случае статического связывания используются не конкретные объекты, а

информация о типе, то есть для обнаружения нужного метода используется тип ссылочной переменной. С другой стороны, при динамическом связывании для нахождения нужного метода в Java используется конкретный объект.

4. Объясните, как вы понимаете, что такое неявная ссылка this? В каких методах эта ссылка присутствует, а в каких – нет, и почему?

this – это ссылка на текущий экземпляр класса. Присутствует в нестатических методах, отсутствует в статических, потому что не на что ссылаться.

5. Что такое финальные поля, какие поля можно объявить со спецификатором final? Где можно инициализировать финальные поля?

Финальные поля – это поля, инициализировать которые можно только один раз, и после этого изменить нельзя. Со спецификатором final можно объявить классы, методы, поля. Инициализировать финальные поля можно сразу при объявлении, в блоке инициализации, в конструкторе, а static final в статическом блоке инициализации.

6. Что такое статические поля, статические финальные поля и статические методы. К чему имеют доступ статические методы? Можно ли перегрузить и переопределить статические методы? Наследуются ли статические методы?

Кроме обычных методов и полей класс может иметь статические поля, методы, константы и инициализаторы. Для объявления статических переменных, констант, методов и инициализаторов перед их объявлением указывается ключевое слово static.

При создании объектов класса для каждого объекта создается своя копия нестатических обычных полей. А статические поля являются общими для всего класса. Поэтому они могут использоваться без создания объектов класса. Также статическими бывают константы, которые являются общими для всего класса.

Статические инициализаторы предназначены для инициализации статических переменных, либо для выполнения таких действий, которые выполняются при создании самого первого объекта.

Статические методы также относятся ко всему классу в целом. При использовании статических методов надо учитывать ограничения: в статических методах мы можем вызывать только другие статические методы и использовать только статические переменные. Статический метод нельзя переопределить. По аналогии с переменными, можно сказать, что этот метод "один для класса и его наследников" – так же, как статическая переменная "одна для класса и всех его объектов".

Все доступные методы наследуются подклассами. Подкласс наследует все открытые и защищенные члены своего родителя, независимо от того, в каком пакете находится подкласс. Если подкласс находится в том же пакете, что и его родитель, он также наследует члены package-private родителя. Наследуемые элементы можно использовать как есть, заменять их, скрывать или дополнять новыми члены единственное отличие от унаследованных статических (class) методов и унаследованных нестатических (instance) методов заключается в том, что при написании нового статического метода с той же сигнатурой старый статический метод просто скрыт, а не переопределен.

7. Что такое логические и статические блоки инициализации? Сколько их может быть в классе, в каком порядке они могут быть размещены и в каком порядке вызываются?

Логические блоки инициализации – это участки кода, находящиеся внутри класса, но вне методов. Статические блоки инициализации – тоже самое, но со словом static. Сначала вызываются статические блоки в порядке объявления, потом нестатические, тоже в порядке объявления.

8. Что представляют собой методы с переменным числом параметров, как передаются параметры в такие методы и что представляет собой такой параметр в методе? Как осуществляется выбор подходящего метода, при использовании перегрузки для методов с переменным числом параметров?

Методы с переменным числом параметров могут принимать в качестве агрументов неопределенное кол-во параметров, которые передаются списком через запятую. Объявляется такой метод как func_name(type ... v). Выбор функции при перегрузке осуществляется как и в обычном методе.

9. Чем является класс Object? Перечислите известные вам методы класса Object, укажите их назначение.

Хотя мы можем создать обычный класс, который не является наследником, но фактически все классы наследуются от класса Object. Все остальные классы, даже те, которые мы добавляем в свой проект, являются неявно производными от класса Object. Поэтому все типы и классы могут реализовать те методы, которые определены в классе Object.

- public final Class<?> getClass() - возвращает класс объекта
- public int hashCode() - возвращает хеш-код объекта
- public boolean equals(Object obj) - возвращает истинность одинаковости объектов
- protected Object clone() - создает и возвращает копию объекта
- public String toString() - возвращает строковое представление объекта и другие.

10. Что такое хэш-значение? Объясните, почему два разных объекта могут сгенерировать одинаковые хэш-коды?

Метод hashCode(). Зачем он нужен? Ровно для той же цели — сравнения объектов. Но ведь у нас уже есть equals()! Зачем же еще один метод? Ответ прост: для повышения производительности.
Хэш-функция, которая представлена в Java методом hashCode(), возвращает числовое значение фиксированной длины для любого объекта. В случае с Java метод hashCode() возвращает для любого объекта 32-битное число типа int.
Сравнить два числа между собой — гораздо быстрее, чем сравнить два объекта методом equals(), особенно если в нем используется много полей.
Если в нашей программе будут сравниваться объекты, гораздо проще сделать это по хэш-коду, и только если они равны по hashCode() — переходить к сравнению по equals().
Таким образом, кстати, работают основанные на хеше структуры данных — например, известная тебе HashMap!
По-хорошему, у разных объектов хешкод должен быть разный. Но на практике иногда происходит по другому.
Очень часто это происходит из-за несовершенства формулы для вычисления хешкода.

11. Что такое объект класса Class? Чем использование метода getClass() и последующего сравнения возвращенного значения с Type.class отличается от использования оператора instanceof?

Объект класса Class представляет классы и интерфейсы в запущенном приложении. Сравнение class и getClass() будет true, только если классы совпадают, instanceof же вернет true, также когда сравниваются класс и его родитель.

12. Укажите правила переопределения методов equals(), hashCode() и toString().

Соблюдать контракты hashCode и equals. toString желательно переопределять всегда (чтобы не выводил неинформативную информацию). Контракт equals (equals должен быть):

- рефлексивным (x.equals(x) = true)
- симметричеым (y.equals(x) = x.equals(y))
- транзитивным (если x.equals(y) = true и y.equals(z) = true, то x.equals(z) = true)
- консистентным (если не менялись объекты, то и equals не должен изменить возвращаемое значение)

Контракт hashCode:

* внутренняя консистентность (не менялся объект – не меняется и hashCode)
* консистентность с equals (если x.equals(y) = true, то и x.hashCode() = y.hashCode())
* коллизии (даже если x.equals(y) = false, но hashCode могут совпадать)