Programming with classes

1. Поля класса инициализируются в момент создания класса, а поля экземпляра класса в момент создания экземпляра класса. Поля класса инициализируются, когда компаратор доходит до создания класса, а поля экземпляра класса, когда компаратор доходит до создания экземпляра. Значение по умолчанию зависит от того какого типа поля примитивные типы 0, boolean false, ссылочные null. Так же значения помогут присваиваться в блоках инициализации о них рассказывается дальше.

2. Перегрузка методов — это когда у нас есть несколько методов с одинаковым названием, но принимающие разное значение и компаратор будет решать какой метод вызывать может быть такое что значение, которое мы передаем в метод не будет совпадать с теми значениями, которые у нас определенны в методе в таком случае компаратор начинает несколько этапный поиск подходящего метода они будут описываться дальше. Перегрузка методов поддерживает полиморфизм, поскольку она является одним из способов реализации парадигмы "один интерфейс - множество методов". Любые методы можно перегружать, главное, чтобы тип и/или число параметров в каждом из перегружаемых методов должны быть разными. Мы можем перегружать метод базового класса, но для этого нужно использовать команду using чтобы перенести ту функцию в наше пространство имен, так же в производном классе разрешается перегружать любые видимые методы (не final). Private методы нельзя переопределять. Перегружать конструкторы можно как и менять у них атрибуты доступа.

3. Ранее связывание производится компаратором, который решает какой метод будет вызваться в данном месте кода, позднее связывание происходит во время выполнения кода, и обычно происходит с переопределенными методами, когда мы при помощи введенных данных решаем, какой метод вызывать. Перегрузка — это ранее связывание, так как мы сразу определяем какой метод вызывать по типу принимаемого значения. Так же во время решения какой метод вызывать компаратор смотрит на возвращаемы тип. В случае если сразу нужный метод не нашелся идет поиск в 3 этапа на первом мы идем вверх по иерархии типов например int стал Integer если так не нашло, то происходит уменьшение типа Integer стал int если это не помогло, то на последнем этапе он компаратор проходит по всей иерархии в поисках подходящего метода (нужно помнить, что шаг для преобразования используется единожды).

4. Неявная ссылка this — эта ссылка на экземпляр класса, к которому мы обращаемся, она невидима для нас, но видна конструктору, и поэтому, когда мы пишем this.name, он будет обращается к полю name экземпляра класса,

для которого мы его вызвали. Данная ссылка присутствует в методах, которые вызываются для определенного экземпляра и обращаются к его полям (т.е. в статических классах они отсутствуют из-за того, что статические методы могут вызываться вне контекста объекта, просто обращаясь к имени класса).

5. Final поля — это поля, которые нельзя модифицировать. Но мы должны сразу задавать значение переменной типа final. Final можно применять как к полям класса, так и к методам, и даже классам, в случае класса — это значит, что у этого класса не может быть наследников. Чистая конечная переменная экземпляра класса должна быть обязательно назначена в конце каждого конструктора класса, в котором она объявлена; аналогично, пустая конечная статическая переменная должна быть определенно назначена в статическом инициализаторе класса, в котором она объявлена; в противном случае в обоих случаях возникает ошибка времени компиляции.

6. Методы и поля, помеченные как static напрямую связаны с классом и будут одинаковые для всех экземпляров класса, и мы можем обращаться к данным полям/методам используя название класса. Статический метод может иметь доступ только к статическим данным. В static методе все переменные должны быть static, а сами static переменные мы можем использовать и в не static методах. Static методы нельзя переопределять, но можно перегружать, и да, они наследуются.

7. Статический блок инициализации – служит для выполнения любых инициализирующих действий в классе до создания конкретных объектов. Логическим блоком называется код, заключенный в фигурные скобки и не принадлежащий ни одному методу текущего класса. Блоки вызываются в порядке их размещения в коде. Размещены они могут быть в любом порядке и их может быть сколько душе угодно.

8. Метод с переменным числом параметров — это метод способный принимать переменное число аргументов. Это значит, что он может принять как 5 аргументов, так и 1 если он примет 1, то остальные 4 будут проинициализированы по умолчанию. Списки аргументов переменной длины обозначаются символом многоточия (...). При перегрузке выбирается метод наиболее подходящий метод (тот в котором больше совпадений).

9. Object – является суперклассом, от которого происходят все остальные классы. HashCode – генерирует число на основе конкретного объекта (уникальное значение). toString-возвращает значение объекта в виде строки. Equals- сравнивает 2-а объекта на равенство. getClass-позволяет получить тип данного объекта. Finalize – деструктор. Clone – создает копию объекта и возвращает на нее ссылку. Notify – снятие паузы. wait – пауза.

10. функция hashCode () для объекта возвращает номер ячейки памяти, где объект сохраняется (данное значение называется хэш-значение). Поэтому, если изменение в код приложения не вносятся, то функция должна выдавать одно и то же значение. При незначительном изменении кода значение hashCode также изменится. Так же хэш код бита обычно совпадает с его юникодом. Значение hashCode зависит от реализации. например, String класс реализует функцию hashCode() в зависимости от значения. это означает что String a=new String("b"); String b=new String("b");
будет иметь тот же hashCode, но это два разных объект, и a==b вернется false.

11. getClass() возвращает объект Class который содержит некоторые методанные о классе:
    a. название
    b. пакет
    c. методы
    d. поля
    e. конструкторы
    f. аннотации

так же можно провести некоторые полезные проверки, например, можно написать так obj.getClass(). isAnnotationPresent(Processable.class) данная строка кода смотрит на, то есть ли у объекта класса аннотация Processable. каждый объект в java имеет (принадлежит) класс и имеет соответствующий объект Class, который содержит метаданные о нем, который доступен во время выполнения. Объект Class является своего рода мета-объектом, описывающим класс объекта (чертеж реального класса).

Оператор instanceof нужен, чтобы проверить, был ли объект, на который ссылается переменная X, создан на основе какого-либо класса Y. Он нам может пригодится для проверки можно ли привести один ссылочный тип к другому.

Instanceof возвращает значение типа Boolean, а getClass возвращает объект Class.

Как мы знаем конструктор базового класса, если он есть, всегда вызывается первым при создании любого объекта. Instanceof руководствуется именно этим принципом, когда пытается определить, был ли объект А создан на основе класса Б. Если конструктор базового класса вызван, значит никаких сомнений быть не может.

instanceof проверяет, является ли ссылка объекта в левой части (LHS) экземпляром типа в правой части (RHS) или некоторым подтипом. getClass () == проверяет, идентичны ли типы.

12. Правила переопределения данных методов весьма просты и существуют, потому что это контрактные методы класса Object.

Equals:

a. Рефлективность - для любого заданного значения x, выражение x.equals(x) должно возвращать true. x!=0
b. Симметричность-для любых заданных значений x и y, x.equals(y) должно возвращать true только в том случае, когда y.equals(x) возвращает true.
c. Транзитивность-для любых заданных значений x, y и z, если x.equals(y) возвращает true и y.equals(z) возвращает true, x.equals(z) должно вернуть значение true.
d. Согласованность-для любых заданных значений x и y повторный вызов x.equals(y) будет возвращать значение предыдущего вызова этого метода при условии, что поля, используемые для сравнения этих двух объектов, не изменялись между вызовами.
e. Сравнение null-для любого заданного значения x вызов x.equals(null) должен возвращать false.
   Только в случае, если ваш переопределенный метод соответствует всему перечисленному считается что вы выполнили контракт.

HashCode:

a. Вызов метода hashCode один и более раз над одним и тем же объектом должен возвращать одно и то же хэш-значение, при условии, что поля объекта, участвующие в вычислении значения, не изменялись.
b. вызов метода hashCode над двумя объектами должен всегда возвращать одно и то же число, если эти объекты равны (вызов метода equals для этих объектов возвращает true).
c. Вызов метода hashCode над двумя неравными между собой объектами должен возвращать разные хэш-значения. Хотя это требование и не является обязательным, следует учитывать, что его выполнение положительно повлияет на производительность работы хэш-таблиц.

Методы HashCode и equals необходимо переопределить попарно, потому что если один переопределить, а второй нет то может случится так что по методу equals они будут равны, а вот не переопределённый HashCode будет считать, что это 2 разных объекта и поместив некий объект в hash таблицу мы рискуем его не получить обратно. В случае если мы определим метод Hashcode, а equals нет то мы все еще не сможем найти наш объект в hash таблице так как для успешного поиска объекта в хэш-таблице помимо сравнения хэш-значений ключа используется также определение логического равенства ключа с искомым объектом. Т. е. без переопределения метода equals никак не получится обойтись.

toString:

a. Должен вызываться для всех полей объекта.
b. Всегда возвращает String