Programming with classes

1. OOP, Classes and Objects, Inheritance, Class Object, Interfaces

1. Наследование — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом.

Абстракция — означает выделение главных, наиболее значимых характеристик предмета и наоборот — отбрасывание второстепенных, незначительных.

Инкапсуляция — в Java означает ограничение доступа к данным и возможностям их изменения.

Полиморфизм — это возможность работать с несколькими типами так, будто это один и тот же тип. При этом поведение объектов будет разным в зависимости от типа, к которому они принадлежат.

2. 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 могут присваиваться в блоках инициализации.

3. Java-bean это стандарт в соответствие с которым, все свойства частные (используем геттеры/сеттеры), конструктор без аргументов и должен реализовывать интерфейс serializable (Социализация — это процесс сохранения состояния объекта в последовательность байт.)

4. 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.

5. 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).

6. 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).

7. 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.

8. 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.

9. 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.

10. 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).

11. 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.

12. 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.

13. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Например, у нас уже есть готовый класс Person он имеет поля возраст, голод и другие теперь нам нужно создать класс Student куда практичней создать его на основе Person, потому что он будет иметь те же поля что и Person + свои. Private поля и методы не наследуется они доступны только в том классе, где их создали.

14. При создании производного класса, вначале вызывается конструктор родительского класса, затем уже производного. this представляет текущий экземпляр класса, в то время как super - текущий экземпляр родительского класса. super () используется для вызова конструктора без аргументов, или как его ещё называют, конструктора по умолчанию родительского класса. this и super в Java используются для обращения к переменным экземпляра класса и его родителя. Вообще-то, к ним можно обращаться и без префиксов super и this, но только если в текущем блоке такие переменные не перекрываются другими переменными, т.е. если в нем нет локальных переменных с такими же именами, в противном же случае использовать имена с префиксами придется обязательно. this() и super non-static переменные так что их нельзя использовать в main и других static методах. Внутри конструктора this и super должны стоять выше всех других выражений, в самом начале, иначе компилятор выдаст сообщение об ошибке. Из чего следует, что в одном конструкторе не может быть одновременно и this(), и super().

15. Ссылка базового класса может ссылаться на объекты своих производных типов, но наоборот не работает, потому что базовый класс может не обладать методами производного типа. Так же объект производного класса может быть использован вместо объекта базового, но не наоборот опять из-за того, что базовый может не иметь того же метода что и производный.

16. Это когда мы берем метод класса, от которого мы его унаследовали и меняем то, что он делает. Нам это нужно, например, для того чтобы не создавать новый метод с нуля, а внести легкое изменение в уже готовый, например, у нас есть метод Voice для собаки и кота он будет разный, но только в выводимом значение поэтому мы можем унаследовать его от класса Animal и переопределить для кота и собаки. При переопределении мы можем изменить:

1. Модификатор доступа, если расширять (package -> protected -> public)
2. Возвращаемый тип если выполняется Downcasting (нисходящее преобразование, преобразование вниз по иерархии)
3. Имя аргументов

Возможно вовсе убрать секцию throws в методе, так как она уже определена.

Так же, возможно добавлять новые исключения, которые наследуются от объявленных или исключения времени выполнения.

Но мы не можем изменить количество аргументов или их порядок так как в таком случае будет перегрузка.

Переопределять можно только методы классов родителей или суперклассов.

17. Статические методы не могут быть переопределены в точном смысле слова, но они могут скрыть родительские статические методы. На практике это означает, что компилятор будет решать, какой метод выполнять во время компиляции, а не во время выполнения, как это происходит с переопределенными методами экземпляра.

Переопределение: Переопределение в Java просто означает, что конкретный метод будет вызываться на основе типа времени выполнения объекта, а не типа его времени компиляции (что имеет место с переопределенными статическими методами).

Скрытие: Статические методы родительского класса не являются частью дочернего класса (хотя они доступны), поэтому не стоит переопределять его. Даже если вы добавите другой статический метод в подкласс, идентичный методу в его родительском классе, этот статический метод подкласса является уникальным и отличается от статического метода в его родительском классе

В случае методов экземпляра вызывается метод фактического класса объекта.

18. Обозначая метод класса модификатором final, мы имеем ввиду, что ни один производный класс не в состоянии переопределить этот метод, изменив его внутреннюю реализацию. Класс, помеченный как final, не поддается наследованию и все его методы косвенным образом приобретают свойство final. Применение признак, а final в объявлениях классов и методов способно повысить уровень безопасности кода. Если класс снабжен модификатором final, никто не в Состоянии расширить класс и, вероятно, нарушить при этом его контракт (контракт — это когда мы точно знаем, что, например, equals должен совершить сравнение по всем полям и ничто не в силах этому помешать) Использование модификатора final в объявлениях классов способствует также повышению эффективности некоторых операций проверки типов. В этом случае многие подобные операции могут быть выполнены уже на стадии компиляции и поэтому потенциальные ошибки выявляются гораздо раньше. Если компилятор встречает в исходном тексте ссылку

на класс final, он может быть "уверен", что соответствующий объект относится именно к тому типу, который указан.

19. Существует расширяющее и сужающее приведение.

Когда мы напишем:

Animal animalCat = new Cat();

Animal animalDog = new YorkshireTerrier();

Это расширяющее приведение (или неявное). Мы расширили ссылки animalCat и animalDog. Они ссылаются на объекты Cat и Dog. При таком приведении мы не можем через ссылку animalCat/animalDog вызвать методы, которые есть в Cat/Dog, но которых нету в Animal.

Сужающее приведение (или явное) происходит в обратную сторону:

Animal animalCat = new Cat();

Animal animalDog = new YorkshireTerrier();

Cat cat = (Cat) animalCat;

YorkshireTerrier dog = (YorkshireTerrier) animalDog;

В данном случае мы явно указали к какому типу хотим привести данный объект.

Опасность явного приведения в том, что мы можем написать

YorkshireTerrier dog = (YorkshireTerrier) animalCat; и компилятор примет это, а вот во время исполнения произойдет ошибка. Она произойдет из-за того, что RunTime видит, что Cat и YorkshireTerrier два разных класса избежать этого можно, используя команду instanceof если не знаете, что будет дальше или просто не делать явное приведение. Приведение нам нужно тогда, когда мы хотим собрать несколько классов, наследующихся от одного и вызвать общий метод для них.

20. getClass() возвращает объект Class который содержит некоторые методанные о классе:

1. название
2. пакет
3. методы
4. поля
5. конструкторы
6. 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.

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

Equals:

1. Рефлективность - для любого заданного значения x, выражение x.equals(x) должно возвращать true. x!=0
2. Симметричность-для любых заданных значений x и y, x.equals(y) должно возвращать true только в том случае, когда y.equals(x) возвращает true.
3. 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.
4. 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.
5. 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:

1. 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.
2. 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).
3. 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:

1. Должен вызываться для всех полей объекта.
2. Всегда возвращает String

22. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. При объявлении абстрактных классов используется ключевое слово abstract и использовать конструкторы для создания объекта абстрактного класса нельзя, произойдёт ошибка если попытаться, потому что это нарушит саму мысль абстрактного класса так же в абстрактном классе иногда отсутствует реализация некоторых методов. В следствие чего

можно понять, что у методов в абстрактном классе тело может быть, а может и не быть.

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный. Абстрактный класс может содержать внутри себя не абстрактные методы, но абстрактные методы могут быть только в абстрактном классе. Абстрактные классы нужны нам, когда у нас есть несколько классов для которых было бы удобно сделать общие поля и методы, которые будут немного отличатся в таком случаем мы используем абстрактный класс и переопределяем его методы. Да, в абстрактном классе в Java можно объявить и определить конструкторы. Поскольку создавать экземпляры абстрактных классов нельзя, вызвать такой конструктор можно только при формировании цепочки конструкторов, то есть при создании экземпляра конкретного класса-реализации.

23. Интерфейс описывает поведение, которым должны обладать классы, реализующие этот интерфейс. «Поведение» — это совокупность методов. Создание интерфейса очень похоже на создание обычного класса, только вместо слова class мы указываем слово interface. Определить интерфейс можно по ключевому слову interface и по отсутствию тела у методов данного класса. Для того чтобы воспользоваться методом определенном в интерфейсе нужно имплементировать интерфейс и реализовать его метод. Классы имплементирующие методы класса интерфейса должны реализовывать методы интерфейса иначе будет ошибка. Так же преимущество интерфейсов в том, что множественного наследования в java нет, а вот интерфейсов класс может реализовывать сколько захочет. Методы в классе интерфейсе могут иметь спецификатор default, что будет указывать на то, что данный метод не нужно реализовывать, и он будет общий у всех. В интерфейсе нельзя реализовывать конструкторы. Анонимные классы могут реализовывать интерфейсы без ключевого слова implements. Когда вы объявляете интерфейс, вы объявляете новый ссылочный тип данных. Вы можете использовать название интерфейса в качестве типа данных так же, как и любые другие типы. Если вы объявляете переменную типа интерфейса, то вы можете присвоить ей объект любого класса, который реализует этот интерфейс.

Переопределять интерфейсы нельзя (вдруг резко решили добавить новый метод) нельзя, потому что классы, которые уже реализовали интерфейс сломаются так что лучше определить новый интерфейс, который расширит старый.

24. Clone делает по битовую копию. Интерфейс Clonable нужен для создания клонов, которые смогли бы существовать отдельно от оригинала. Например, при обыкновенном присваивание объектов (obj1 = obj2;) передаются ссылки на объект. В итоге два экземпляра ссылаются на один объект, и изменение одного приведет к изменению другого, а после использовании команды clone у нас будет 2 независимых версии объекта. метод clone () может выбрасывать исключение CloneNotSupportedException. Данное исключение возникает в случае, когда клонируемый класс не имеет реализации интерфейса Cloneable. Интерфейс Cloneable не реализует ни одного метода. Он является всего лишь маркером, говорящим, что данный класс реализует клонирование объекта. Само клонирование осуществляется вызовом родительского метода clone(). Например, class User implements Cloneable после этого мы можем реализовать метод clone

```
public User clone() throws CloneNotSupportedException {
    User clone = (User)super.clone();
    return clone;
    }
```

При помощи которого копировать экземпляры класса User и делать с клонами что хотим без изменения оригинала. Для того чтобы клонировать поля класса, к которому мы обращаемся мы должны вызывать команду клон на класс через команду super. В случае если мы хотим добавить некое значение нашему клону из библиотеки и прировнять ее нашему полю мы должны написать так clone.birthday = (GregorianCalendar) birthday.clone(); где GregorianCalendar библиотека, а birthday поле нашего клона.

25. Если мы хотим сравнить и отсортировать объекты некого класса они должны применять интерфейс Comparable<E>. При применении интерфейса он типизируется текущим классом. Интерфейс Comparable содержит один единственный метод int compareTo(E item), который сравнивает текущий объект с объектом, переданным в качестве параметра. Если этот метод возвращает отрицательное число, то текущий объект будет располагаться перед тем, который передается через параметр. Если метод вернет положительное число, то, наоборот,

после второго объекта. Если метод возвратит ноль, значит, оба объекта равны.

Пример compareTo: работает только по 1 полю.

```
 public int compareTo(Person p){
    return name.length()-p.getName().length();
}
```

Так же существует более гибкий способ на случай если разработчик не реализовал Comparable это интерфейс Comparator<E>. интерфейс Comparator содержит ряд методов, ключевым из которых является метод compare():

Пример compare: сколько угодно полей сортировать

```
public int compare(Person a, Person b){
 return a.getName().compareTo(b.getName());
   }
```

Метод compare также возвращает числовое значение - если оно отрицательное, то объект a предшествует объекту b, иначе - наоборот. А если метод возвращает ноль, то объекты равны. Для применения интерфейса нам вначале надо создать класс компаратора, который реализует этот интерфейс.

Используйте Comparable:

если объект находится под вашим контролем.

если поведение сравнения является основным поведением сравнения.

Используйте Comparator:

если объект находится вне вашего контроля, и вы не можете заставить их реализовать Comparable.

если вы хотите сравнить поведение, отличное от по умолчанию (которое указано в Comparable).

2. Generic classes and Interfaces, Enums

1. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую. Перечисление фактически представляет новый тип, поэтому мы можем определить переменную данного типа и использовать ее. Перечисление — это класс. Объявляя enum, мы неявно создаем класс производный от java.lang.Enum. следовательно элементы перечисления — это элементы класса enum. Мы можете определить методы abstract в объявлении enum, если и только если все значения перечисления имеют собственные тела классов с реализациями этих методов (то есть, нет конкретного значения перечисления может отсутствовать реализация). Enum может наследовать интерфейсы. Поскольку Enum тип схож с классом и интерфейсом, он может наследовать интерфейс. Enum может содержать static методы примером может послужить метод статический метод values, который возвращает массив, содержащий значения объекта в порядке их объявления данный метод добавляется компилятором автоматически. Типы enum также могут использоваться в выражениях switch.

2. Поскольку типы перечислений гарантируют, что в JVM существует только один экземпляр констант, мы можем смело использовать оператор «==» для сравнения двух переменных. Сравнивать мы может как сими переменные, так и некоторые поля, например, имя переменной enum всегда хранится в виде String и его можно сравнить с другой строкой, например, tour.getTourType().name().equals(str). (плохо ответил мало инф)

3. Анонимные классы — это классы, что не имеют имени и их создание происходит в момент инициализации объекта.

4. Шаблон/параметризованный класс/ дженерик – это некоторый шаблон, на основе которого можно строить другие классы. Этот класс можно рассматривать как некоторое описание множества классов, отличающихся только типами их данных. Он определяет целое семейство или множество классов, каждый из которых, может быть получен связыванием этих параметров с действительными значениями.

```
public class Matrix<T> {
    private T[] array;

    public Matrix(T[] array) {
        this.array = array.clone();
    }

    public static void main(String[] args) {
        Matrix<Double> doubleMatrix = new Matrix<>(new Double[2]);
        Matrix<Integer> integerMatrix = new Matrix<>(new Integer[4]);
        Matrix<Byte> byteMatrix = new Matrix<>(new Byte[7]);
    }
}
```

Несмотря на то, что doubleMatrix и integerMatrix имеют тип Matrix<T>, они являются ссылками на разные типы, потому что типы их параметров отличаются.

Работают только с объектами, т.е. параметризовать его примитивными типами нельзя.

3. Exceptions and Errors

1. Исключительная ситуация — Совокупность определенных условий, возникновение которых приводит к нарушению предусмотренной последовательности выполнения в программе. Обработка ошибок в java производится при помощи ключевых слов try – определяет блок кода, в котором может произойти исключение; catch – определяет блок кода, в котором происходит обработка исключения; finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try. throw – используется для возбуждения исключения; throws – используется в сигнатуре методов для предупреждения, о том, что метод может выбросить исключение.

2. Все исключения в Java являются объектами. Поэтому они могут порождаться не только автоматически при возникновении исключительной ситуации, но и создаваться самим разработчиком. Исключение может выбрасываться вручную, например, тогда, когда наша логика программы не позволяет делать данное действие, а для VM все нормально, например, наша программа должна в графу цена записать стоимость покупки, а вместо этого она записала ее в графу телефонный номер покупателя, VM ошибки не увидит, а вот для нас это ошибка, в следствие чего, нужно остановить работу программы и сообщить об ошибке.

3. Исключения делятся на несколько классов, но все они имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error. Так же в java есть такие типы исключений как Checked исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. Unchecked могут не обрабатываться и не быть описанными. Unchecked исключения в Java — наследованные от RuntimeException, checked — от Exception (не включая unchecked). Checked исключения отличаются от Unchecked исключения в Java, тем что: Наличие\обработка Checked исключения проверяются на этапе компиляции. Наличие\обработка Unchecked исключения происходит на этапе выполнения. Исключения, выбрасываемые виртуальной машиной это классические исключения отслеживание, которых было написано разработчиками JVM.

4. Данный оператор состоит из нескольких частей:

try – определяет блок кода, в котором может произойти исключение; catch – определяет блок кода, в котором происходит обработка исключения;

finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try. Данный оператор используется, когда метод, который мы вызываем может вызвать исключение и в таком случае работа программы не будет остановлена о ошибка будет обработана и работа продолжится.

Catch в блоке try, может быть, сколько угодно, но если одно из них сработает, то остальные уже нет (возможно). Блоки наиболее специализированных классов исключений должны идти первыми, поскольку ни один подкласс не будет достигнут, если поставить его после суперкласса. Операторы try можно вкладывать друг в друга. Если у оператора try низкого уровня нет раздела catch, соответствующего возбужденному исключению, стек будет развернут на одну ступень выше, и в поисках подходящего обработчика будут проверены разделы catch внешнего оператора try. «try с ресурсами» это оператор try, в котором объявляются один или несколько ресурсов. Ресурс — это объект, который должен быть закрыт после того, как программа закончит с ним работу. «try с ресурсами» берет всю работу по закрытию ресурсов на себя ресурсы будут закрываться снизу-вверх автоматически после завершения работы блока try. При использовании оператора try с ресурсами, совершенно не обязательно использовать блоки catch и finally, они являются опциональными.

5. Блоки наиболее специализированных классов исключений должны идти первыми, поскольку ни один подкласс не будет достигнут, если поставить его после суперкласса. Если возникает необходимость снова сгенерировать исключения из блока, который обрабатывает исключения, можно сделать это путем вызова throw без указания исключения. В результате текущее исключение будет передано во внешнюю последовательность try/catch обработки исключений. Причиной для этого может послужить желание обрабатывать исключения несколькими обработчиками. Например, один обработчик может заниматься одним аспектом исключения, а второй обработчик — другим. Исключение может быть снова сгенерировано или изнутри блока catch, или из функции, вызванной в этом блоке. Когда повторно генерируется исключение, оно не будет перехвачено той же самой

инструкцией catch. Оно будет распространяться до следующей внешней инструкции catch.

6. Finally будет вызываться после выполнения блоков кода try или catch. Единственные времена finally будут называться:

1. Если вы вызываете System.exit();
2. Если JVM падает в первую очередь;
3. Если JVM достигает бесконечного цикла (или некоторого другого непрерываемого, не заканчивающегося оператора) в блоке try catch;
4. Если ОС принудительно завершает процесс JVM; например, "убить -9" в UNIX.
5. Если хост-система умирает; например, сбой питания, аппаратная ошибка, паника ОС и так далее.
6. Если блок finally будет выполняться потоком демона и все остальные потоки, не являющиеся демонами, завершат работу до вызова метода finally. Да, finally может выбрасывать исключение (например, если мы в блоке finally вызываем метод, который вызовет исключительную ситуацию). Finally срабатывает только 1 раз (не уверен не нашел).