[#1.Установка JDK 7](#_Toc1)

[#2. Установка eclipse 7](#_Toc2)

[#3. Установка intellij idea 7](#_Toc3)

[#4.ООП. Hello World. 7](#_Toc4)

[#5.Запуск прог.без main (у меня не работает) 8](#_Toc5)

[#6. Типы данных. 8](#_Toc6)

[#7.где можно использовать типы данных 8](#_Toc7)

[#8. Коментарии 8](#_Toc8)

[#9. Пакеты и импорты 9](#_Toc9)

[#10. Наследование. Extends 9](#_Toc10)

[#11.Полиморфизм. @Override 9](#_Toc11)

[#12.Инкапсуляция 9](#_Toc12)

[#13.Идентификаторы 10](#_Toc13)

[#14.Абстрактные и финальные классы 10](#_Toc14)

[#16.Модификаторы доступа 11](#_Toc15)

[#17.Переменные 11](#_Toc16)

[#18.Методы.разные модификаторы 11](#_Toc17)

[#19.конструктор класса 12](#_Toc18)

[#20.массивы 12](#_Toc19)

[#21.Enums 12](#_Toc20)

[#22.перегрузка методов 12](#_Toc21)

[23#конструкторы 2. This и super 12](#_Toc22)

[#24.переопределение методов 13](#_Toc23)

[#25.Static 13](#_Toc24)

[#26.блоки инициализации 14](#_Toc25)

[#27.инициализация массива 14](#_Toc26)

[#28.обертки 14](#_Toc27)

[#29.перегрузка методов 2 15](#_Toc28)

[#30.установка jdk на Линукс 15](#_Toc29)

[#31.Установка eclipse под Линукс 15](#_Toc30)

[#32.установка lntellij idea под Линукс 16](#_Toc31)

[#33.установка oracle jdk на Линукс 16](#_Toc32)

[#34. Garbage Collector (сбор мусора) 16](#_Toc33)

[#35. операторы 16](#_Toc34)

[#36. main(String[] args) 16](#_Toc35)

[#37. if else 17](#_Toc36)

[#38.swith 17](#_Toc37)

[#39. while, do while 17](#_Toc38)

[#40. For 17](#_Toc39)

[#41. Улучшенный for 17](#_Toc40)

[#42. Исключения 17](#_Toc41)

[#43.Assertions 18](#_Toc42)

[#44.StringBuilder 18](#_Toc43)

[#45.работа с файлами 18](#_Toc44)

[#46.сериализация 19](#_Toc45)

[#47.сериализация наследуемых классов 19](#_Toc46)

[#48.дата.календарь.время 20](#_Toc47)

[#49.Работа с "Map",equals и HashMap 20](#_Toc48)

[#50. Коллекции. 21](#_Toc49)

[#51.сортировка коллекции 21](#_Toc50)

[#52.ArrayList. Vector. LinkedList. 22](#_Toc51)

[#53.Queue очередь 22](#_Toc52)

[#54. Map. 23](#_Toc53)

[#55.Set 23](#_Toc54)

[#56.Generic. Обобщения 24](#_Toc55)

[#57.Inner Classes 25](#_Toc56)

[#58. Анонимные классы 25](#_Toc57)

[#59. Клонирование 26](#_Toc58)

[#60.Reflection - рефлексия 27](#_Toc59)

[#61. технологии в  програмировании 28](#_Toc60)

[#62.запуск через командную строку 29](#_Toc61)

[#63.Про собеседование 29](#_Toc62)

[#64.Формирование проектов из джар файлов с папками и пакетами 29](#_Toc63)

[#65. чтение из консоли 29](#_Toc64)

[#66.Мноогопоточность 30](#_Toc65)

[#67.Многопоточность. жизненный цикл потоков 30](#_Toc66)

[#68. Синхронизация 31](#_Toc67)

[#69.синхронизация статических полей 31](#_Toc68)

[#70.Многопоточность.volatile 32](#_Toc69)

[#71.Многопоточность. атомарные переменные 32](#_Toc70)

[#72.многопоточность.синхронизированные коллекции 32](#_Toc71)

[#73.многопоточность. Deadlock 32](#_Toc72)

[#74.многопоточность. wait и notify 33](#_Toc73)

[#75.Wait и notify продолжение поточное голодание 33](#_Toc74)

[#76. Многопоточность. ReenrantLock 34](#_Toc75)

[#76. Многопоточность. ReenrantLock 34](#_Toc76)

[#77.Многопоточность. TryLock 35](#_Toc77)

[#78.Многопоточность. Condition\пересмотреть 35](#_Toc78)

[#79.Callable and Futures//пересмотреть 36](#_Toc79)

[#80.многопоточность.Executur 37](#_Toc80)

[#81.многопоточность.newScheduledThreadPool() 40](#_Toc81)

[#82.Semafore 41](#_Toc82)

[#83.CountDownLatch 42](#_Toc83)

[#84.Exchenger.Обменник 42](#_Toc84)

[#85.Многопоточность.CyclicBarrier 43](#_Toc85)

[#86.Многопоточность. Phaser 44](#_Toc86)

[#87.многопоточность.BlockingQueue 46](#_Toc87)

[#88.многопоточность.FreadFactory 46](#_Toc88)

[#89.многопоточность.ForkJoinFramevork 46](#_Toc89)

[#90.многопоточность.TimeUnit и ThreadLokalRandom 47](#_Toc90)

[#91.Dinamic Proxy 47](#_Toc91)

[#92.литература для чтения 48](#_Toc92)

[#93.Уроки на github 48](#_Toc93)

[#94юзарплаты джавистов 48](#_Toc94)

[#95.как получить максимальную зарплату 49](#_Toc95)

[#96. Swing 49](#_Toc96)

[#97.JComponent 49](#_Toc97)

[#98.Action. Listener 51](#_Toc98)

[#99.Swing.EventHandler 52](#_Toc99)

[#100.LookandFeel 53](#_Toc100)

[#101.WindowListener 54](#_Toc101)

[#102.Actions 55](#_Toc102)

[#103.KeyStroke короткое сочетание клавиш 56](#_Toc103)

[#104.MouseListener. 57](#_Toc104)

[#105.Текстовые поля JtextFild,JPasswordFild,JTextArea,ScrollPane 59](#_Toc105)

[#106.swing. галочка выбора. Radio button,RadioGroup,ComponentBox,Border 59](#_Toc106)

[#107.swing.Sliders 60](#_Toc107)

[#108.swing.Menu 60](#_Toc108)

[#109.swing.PopupMenu 61](#_Toc109)

[#110.ToolBar 62](#_Toc110)

[#111.Layout 62](#_Toc111)

[#112.Swing.Border and Gride Layout 62](#_Toc112)

[#113.GridBagLayout 63](#_Toc113)

[#114.swing. No Layout 64](#_Toc114)

[#115.swing. Custom layout 64](#_Toc115)

[#116.swing. Focus 65](#_Toc116)

[#117.swing.dialogs 65](#_Toc117)

[118.swing. Custom dialogs 65](#_Toc118)

[#119.File Dilog 66](#_Toc119)

[#120.swing. ColorDialog 67](#_Toc120)

[121.awt. shapes 67](#_Toc121)

[#122.awt.Areas 67](#_Toc122)

[#123.awt.Stroke 68](#_Toc123)

[124.awt. Paint 68](#_Toc124)

[#125.awt. Transformation 68](#_Toc125)

[#126.awt.Clip 68](#_Toc126)

[127.awt.Transparency 68](#_Toc127)

[#128.awt.Скорость и качество отрисовки фигур 68](#_Toc128)

[#129.awt.чтение и запись изображений 68](#_Toc129)

[#130.awt.работа с изображениями 68](#_Toc130)

[#131.awt. image filter 68](#_Toc131)

[#132.awt. печать изображений. 69](#_Toc132)

[#133.awt. Печать документов 69](#_Toc133)

[#134.awt. Print PostServise 69](#_Toc134)

[#135.awt. Работа с буфером обмена, основы, копирование строк 69](#_Toc135)

[#136.awt. буфер обмена. Копирование картинок 69](#_Toc136)

[#137.awt. DragandDrop 69](#_Toc137)

[#138.awt. Splashscreen. Заставка 69](#_Toc138)

[#139.awt. Запуск других приложений,файлов и браузера 69](#_Toc139)

[#140.awt. System tray панель задач 70](#_Toc140)

[#141.установка жду на мак ос 70](#_Toc141)

[#142.установка эклипса на мак ос 70](#_Toc142)

[#143. Установка айдие на мак ос 70](#_Toc143)

[#144. Создание самозапускающегося jar 70](#_Toc144)

[#145. Аплеты  70](#_Toc145)

[#146. Java WedStart 70](#_Toc146)

[#147.Properties. 70](#_Toc147)

[#148.Preferences. 70](#_Toc148)

[#149.логирование 71](#_Toc149)

[#150. Debagging 71](#_Toc150)

[#151. Debagging swing 71](#_Toc151)

[#152. Robot 71](#_Toc152)

[#153. Как ему предложили работу в США 71](#_Toc153)

[#154. JavaDoc 71](#_Toc154)

[#155.Overlioading 71](#_Toc155)

[#156.переопределение equals 71](#_Toc156)

[#157. instanseof null 72](#_Toc157)

[#158.почему нет множественного наследования в Java 72](#_Toc158)

[#159. Статическая инициализация 72](#_Toc159)

[#160. final 72](#_Toc160)

[#161. Enums 72](#_Toc161)

[#162. Static inner classes 72](#_Toc162)

[#163.inner class 72](#_Toc163)

[#164. Локальный класс 73](#_Toc164)

[#165. Интерфейсы 73](#_Toc165)

[#166. Try с ресурсами и AutoClosable 73](#_Toc166)

[#167.отношения между классами is a и has a 73](#_Toc167)

[#168.Cohesion and Coupling 73](#_Toc168)

[#169.работа с файлами и стримы 73](#_Toc169)

[#170. path and Files 74](#_Toc170)

[#171. Scanner, PrintWriter 74](#_Toc171)

[#172.xml. DOM подход 74](#_Toc172)

[#173.xml.SAX подход 74](#_Toc173)

[#174.создание XML при помощи dom 74](#_Toc174)

[#175.создание XML при помощи SAX 74](#_Toc175)

[#176.XPath и работа с XML 74](#_Toc176)

[#177.основы сетей 74](#_Toc177)

[#178.Сокеты клиентские 74](#_Toc178)

[#179.про серверсокет 75](#_Toc179)

[#180.http и как получить html 75](#_Toc180)

[#181.отправка емайл 75](#_Toc181)

[#182.установка Интел айдия ультимате 75](#_Toc182)

[#183.JDBS 75](#_Toc183)

[#184.установка mysql на windows 10 75](#_Toc184)

[#185.установка mysql на Линукс 75](#_Toc185)

[#186. Подключение к mysql через айдию 75](#_Toc186)

[#187.подключение к базе данных на примере mysql 76](#_Toc187)

[#188.gdbc SQL запросы 76](#_Toc188)

[#189.sql injection and preparementStatement 76](#_Toc189)

[#190.BLOB Binary Large Objects 76](#_Toc190)

[#191.sql даты 76](#_Toc191)

[#192.JDBS хранимые процедуры 76](#_Toc192)

[#193.JDBS множественные результаты 76](#_Toc193)

[#194.JDBS Scrollable RowSet 76](#_Toc194)

[#195.JDBS Update ResultSet 77](#_Toc195)

[#196.JDBS CachedRowSet кеширование запросов 77](#_Toc196)

[#197.JDBS Metadata 77](#_Toc197)

[#198.JDBS transaction транзакции 77](#_Toc198)

[#199.JDBS.как накатить SQL группой SQL Batch 77](#_Toc199)

[#200.JDBS.уровни изоляции транзакций 77](#_Toc200)

[#201.JMX Java menegment extension 77](#_Toc201)

[#202. JMX. Notification. 78](#_Toc202)

[#203.JMX.на стороне клиента 78](#_Toc203)

[#204.Сравнение строк 78](#_Toc204)

[#205.Форматирование строк 78](#_Toc205)

[#206.Locale-локаль 78](#_Toc206)

[#207.локализация и интернационализация 78](#_Toc207)

[#208.Бины 78](#_Toc208)

[#209.загрузчик классов class loader 78](#_Toc209)

[#210.Byte code verifier 79](#_Toc210)

[#211.Security manager 79](#_Toc211)

[#212.Security manager аутентификация пользователей 79](#_Toc212)

[@#213.работа с шифрованием md5 и sha-1 79](#_Toc213)

[@#214.certification 79](#_Toc214)

[@#215.шифрование AES 79](#_Toc215)

[@#216.шифрование RSA 80](#_Toc216)

[@#217.запуск javascript на jave 80](#_Toc217)

[@#218.компилируем в jar в рантайме 80](#_Toc218)

[@#219. Аннотации 80](#_Toc219)

[@#220.Пример применения аннотаций 80](#_Toc220)

[#221.RMI 80](#_Toc221)

[#222.запуск "native" кода 80](#_Toc222)

[#223.JNDI 80](#_Toc223)

[@#224.JAXB 81](#_Toc224)

[#225.java FX 81](#_Toc225)

[#226.java FX packager 81](#_Toc226)

[#227. добавление фигур в java FX 81](#_Toc227)

[#228.canvas 81](#_Toc228)

[#229. java FX properties and binding 81](#_Toc229)

[#230. java FX layout 81](#_Toc230)

[#231. java FX UI controls 82](#_Toc231)

[#232. java FX кастомный контрол control 82](#_Toc232)

[#233. java FX обработка событий 82](#_Toc233)

[#234. java FX работа с изображениями 82](#_Toc234)

[#235. javaFX анимации 82](#_Toc235)

[#236. javaFX scene builder 82](#_Toc236)

[#237. javaFX CSS 82](#_Toc237)

[#238. javaFX FXML 82](#_Toc238)

[#239. javaFX медиа 82](#_Toc239)

[#240. javaFX отображение html и javascript в WebWiew 83](#_Toc240)

[#241. javaFX графики charts 83](#_Toc241)

[#242. javaFX работа с 3D 83](#_Toc242)

[#243. javaFX коллекции в фх 83](#_Toc243)

[#244. javaFX concurency 83](#_Toc244)

[#245. javaFX swing and fx 83](#_Toc245)

[Java 9 L1 установка JDC 83](#_Toc246)

[Java 9 L1 JShell 83](#_Toc247)

[Java 9 L3 ProcessHandler 83](#_Toc248)

[Java 9 L4 приватные методы в интерфейсах 84](#_Toc249)

[Java 9 L5 коллекции 84](#_Toc250)

[Java 9 L6 StackWalker 84](#_Toc251)

[Java 9 L7 modules 84](#_Toc252)

[Java 9 L8 изменения try с ресурсами 84](#_Toc253)

[Java 9 L9  85](#_Toc254)

[#246. RegEx что такое рег.выражения 85](#_Toc255)

[#247. RegEx буквальные выражения 85](#_Toc256)

[#248. RegEx наборы символов 85](#_Toc257)

[#249. RegEx точка 85](#_Toc258)

[#250. RegEx границы слов 85](#_Toc259)

[#251. RegEx начало и конец строки 85](#_Toc260)

[#252. RegEx Pipeline 86](#_Toc261)

[#254. RegEx символы звездочка и плюс 86](#_Toc262)

[#255. RegEx группы 86](#_Toc263)

[#256. RegEx именованные группы 86](#_Toc264)

[#257. RegEx unicode 86](#_Toc265)

[#258. RegEx режимы рег.выражений 86](#_Toc266)

[#259. RegEx possesive quantifier 86](#_Toc267)

[#260. RegEx atomic group 87](#_Toc268)

[#261. RegEx look around 87](#_Toc269)

[#262. RegEx множественные требования 87](#_Toc270)

[#263. RegEx if then else 87](#_Toc271)

[#264. RegEx posix simbols 87](#_Toc272)

[#265. RegEx коментарии 87](#_Toc273)

[#266. RegEx примеры 87](#_Toc274)

[#267. RegEx тренировались на сайте regexone 87](#_Toc275)

[#268. Patterns. что такое паттерны 88](#_Toc276)

[#269. Patterns. Factory 88](#_Toc277)

[#270. Patterns. Abstract Factory 88](#_Toc278)

[#271. Patterns. Factory Metod 88](#_Toc279)

[#272. Patterns. singleton 88](#_Toc280)

[#273. Patterns. Builder 89](#_Toc281)

[#274. Patterns. Prototype 90](#_Toc282)

[#275. Patterns. Object pool 90](#_Toc283)

[#276. Patterns. Adapter 91](#_Toc284)

[#277. Patterns. Bridje 92](#_Toc285)

[#278. Patterns. Filter 92](#_Toc286)

[#279. Patterns. Composite 92](#_Toc287)

[#280. Patterns. Decorator 92](#_Toc288)

[#281. Patterns. Facade 93](#_Toc289)

[#282. Patterns. FlyWeight 93](#_Toc290)

[#283. Patterns. Proxy 93](#_Toc291)

[#284. Patterns. Chain of responsibility 94](#_Toc292)

[#285. Patterns. Command 95](#_Toc293)

[#286. Patterns. Interpreter 96](#_Toc294)

[#287. Patterns. Iterator 96](#_Toc295)

[#288. Patterns. Mediator 96](#_Toc296)

[#289. Patterns. Momento 96](#_Toc297)

[#290. Patterns. Observable 97](#_Toc298)

[#291. Patterns. State 97](#_Toc299)

[#292. Patterns. Null object 97](#_Toc300)

[#293. Patterns. Strategy 97](#_Toc301)

[#294. Patterns. Template 98](#_Toc302)

[#295. Patterns. Visitir 98](#_Toc303)

[#296. Patterns. MVC 98](#_Toc304)

[#297. Patterns. Business Delegate 98](#_Toc305)

[#298. Patterns. Composite Entity 98](#_Toc306)

[#299. Patterns. DAO 99](#_Toc307)

[@#300. Patterns. Front Controller 99](#_Toc308)

[@#301. Patterns. Intercepting Filter 100](#_Toc309)

[#302. Patterns. Servise Lokator 100](#_Toc310)

[#303. Patterns. Transfer Object 100](#_Toc311)

[#304.SOLID что такое солид 100](#_Toc312)

[#305.SOLID single responcibility 100](#_Toc313)

[#306.SOLID. open code principle 100](#_Toc314)

[#307.SOLID. Liskov principle 100](#_Toc315)

[#308.SOLID. Interface sugregation 101](#_Toc316)

[#309.SOLID. Dependency Inversion 101](#_Toc317)

[#310.Generics. 101](#_Toc318)

[#311. Наследование generics 101](#_Toc319)

[#312.Generic методы 101](#_Toc320)

[#313.Bounds generic 101](#_Toc321)

[#314. Whild card 102](#_Toc322)

[#315(Whild card)перепутаны видео 102](#_Toc323)

[#316. ArrayList and LinkedList 102](#_Toc324)

[#317. Stack, Queue, Dequeue 102](#_Toc325)

[#318.BinarySearch 102](#_Toc326)

[#319. выбрасывание ошибок методами 102](#_Toc327)

[#320. эксэпшены и Pipeline operator 102](#_Toc328)

[#321.как работает AutuClosable 102](#_Toc329)

[#322.Аssertions 103](#_Toc330)

[#323. Immutable 103](#_Toc331)

[@#324.ReadWrite lock 103](#_Toc332)

[#325.что такое функциональное программирование 103](#_Toc333)

[@#326.лямбда 104](#_Toc334)

[@#327. функциональные интерфейсы 104](#_Toc335)

[#328. Streams 104](#_Toc336)

[@#329. default и static методы 105](#_Toc337)

[@#330. Optional 105](#_Toc338)

[#331.metod references 105](#_Toc339)

[@#332. collertor 106](#_Toc340)

[@#333. parallel stream 106](#_Toc341)

[@#334. Local Date 106](#_Toc342)

[@#335. как работают лямбда выражения 106](#_Toc343)

[#336. equals и HashCode 107](#_Toc344)

[#337. Clonnable 107](#_Toc345)

[#338.пулл оберток над примитивами 107](#_Toc346)

[#339. коллекции коллекции коллекции 107](#_Toc347)

[#340. List 107](#_Toc348)

[#341. set 108](#_Toc349)

[#342. Queue очередь 108](#_Toc350)

[#343. Map 108](#_Toc351)

[#344. сложность алгоритмов поиска по коллекции 109](#_Toc352)

[#345. Concurrency package 109](#_Toc353)

[#346. volatile, happens befor 109](#_Toc354)

[#347. double check 109](#_Toc355)

[#348. типы reference 109](#_Toc356)

[@#349модель памяти java и алгоритмы сборщиков мусора 110](#_Toc357)

[#350. как поместить стрингу в пул стрингов 110](#_Toc358)

[#351. размещение обьектов вне хипа 110](#_Toc359)

[#352. распределение нагрузки load ballansing 110](#_Toc360)

[#353. immutable коллекции 110](#_Toc361)

[#354. высоконагруженная архитектура 110](#_Toc362)

[#355. шардинг и репликация баз данных 111](#_Toc363)

[#356.создание индексов в базе данных 111](#_Toc364)

[#357. synchronized is bad 111](#_Toc365)

[#358. GRASP паттерны 111](#_Toc366)

[#359. CAP теорема 111](#_Toc367)

[#360. Externalizer 111](#_Toc368)

[#361. immutable классы 112](#_Toc369)

[#362. как обнаружить dead lock 112](#_Toc370)

[#363. ThreadLocal 112](#_Toc371)

[#364. изучаем intellije idiю 112](#_Toc372)

[#365. Best Practice 113](#_Toc373)

[#365. Best Practice 2 113](#_Toc374)

### #1.Установка JDK

Попытка запустить чтото из консоли ничего непонятно ничего не работает

### #2. Установка eclipse

Там 2 проги установил одну только

Создал проект FirstProgect в нем создал класс

**public** **class** HelloWorld {

**public** **static** **void** main(String[] args)

{

//Print"Hello World"to the terminal windows.

System.***out***.println("Hello World");

}

}

зачемто надо было удалить module\_info.java иначе ошибку выдавал

УРААА ЗАРАБОТАЛА

### #3. Установка intellij idea

Скачал установил запустил HelloWorld

### #4.ООП. Hello World.

Почемуто функ.main должна наход. Внутри класса

public class HelloWord  
{  
 int age=45;  
  
 public static void main(String[] args)  
 {  
 HelloWord hw=new HelloWord();  
//Print"Hello World"to the terminal windows.  
 System.*out*.println("Hello World");  
 hw.BuyBread();  
 hw.BuyMilk();  
 hw.Bithday();  
 }  
 void BuyBread() {  
 System.*out*.println("buy bread");  
 }  
void BuyMilk(){  
 System.*out*.println("buy milk");  
}  
void Bithday(){  
 age=age+1;  
 System.*out*.println(age);  
}  
}

### #5.Запуск прог.без main (у меня не работает)

class HelloWord  
{  
 static {  
 System.*out*.println("Hello World");  
 System.*exit*(0);  
 }  
}

### #6. Типы данных.

public class Type\_Data {  
 byte b; //1 байт от -128 до 128  
 short s;//2 байта (2в16й) от -32768 до 32767  
 int i;//4байта -2 147 483 648 до 2 147 483 647  
 long l;//8байт (2в63)  
 float f;//4байта ддробные  
 double d;//8 байт  
 boolean bl;//логическая  
 char ch;//  
 Car car=new Car();//ссылочный тип//класс должен быть заранее создан отдельно  
  
 public static void main(String[] args) {  
 Type\_Data td=new Type\_Data();  
 System.*out*.println(td.car.Spid);  
 }  
}

### #7.где можно использовать типы данных

public class Type\_Data {  
 public static void main(String[] args) {  
 Type\_Data td=new Type\_Data();  
 System.*out*.println(td.Sum(3,5.7778,new Car()));  
 }  
 int nethod(){  
 return 4;  
 }  
 Car nethod1(){  
 return new Car();  
 }  
 double Sum(int i,double d,Car car){return i+d;}  
}

### #8. Коментарии

//коментарий

/\*коментарий\*/

### #9. Пакеты и импорты

В папке src создаем Packages имя типа url com.max

Помещаем туда класс Car

В основном классе пишем com.max.Car car;

Или чтоб много раз не писать в начале кода import com.max.Car; или import com.max.\*; загрузит все классы в пакете

В одном классе может быть несколько классов но только один public

Класс в пакете имеет запись package com.max; свою принадлежность к пакету

### #10. Наследование. Extends

Делаем класс наследником

public class Toyota extends Car{}

Теперь через наследника можно вызывать методы и обьекты класса предка

System.out.println(toyota.GetMaxSpeed());

### #11.Полиморфизм. @Override

Переопределяем метод в наследнике

public class Toyota extends Car{  
 @Override  
 int GetCountSeets(){return 2;}  
}

Теперь можем подменивать метод класса предка методом класса наследника

public class Main {  
 public static void main(String[] args) {  
 Car car=new Car();  
 Car car2=new Toyota();  
 Toyota toyota=new Toyota();  
 *GetInfoCar*(car);  
 *GetInfoCar*( car2);  
 *GetInfoCar*(toyota);  
 }  
 static void GetInfoCar(Car car){  
 System.*out*.println(car.GetCountSeets());  
 }  
}

Зачемто метод должен был быть статическим

### #12.Инкапсуляция

Переменные класса обьявляем public

### #13.Идентификаторы

Правила назначения имен

### #14.Абстрактные и финальные классы

Public class может быть только один

Абстрактный класс

**public** **abstract** **class** Car {

**private** String model;

**private** String color;

**private** **int** maxSpeed;

**public** **abstract** **void** gas();

**public** **abstract** **void** brake();

**public** String getModel() {

**return** model;

}

**public** **void** setModel(String model) {

**this**.model = model;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** **int** getMaxSpeed() {

**return** maxSpeed;

}

**public** **void** setMaxSpeed(**int** maxSpeed) {

**this**.maxSpeed = maxSpeed;

}

}

Его наследник

**public** **class** Sedan **extends** Car {

@Override

**public** **void** gas() {

System.out.println("Седан газует!");

}

@Override

**public** **void** brake() {

System.out.println("Седан тормозит!");

}

}

Абстр.класс не интерфейс множ.насл.нет

Финальные классы

Final class MyFinalClass{} Этот клас нельзянаследовать и его методы прим.в основном в старых библиотеках

#15.Интерфейс.**interface. implements**

Создаем его из менюшки или сами

Имя начинаем с I большой ай

В интерфейсах запр.писать хоть какое тело

Все переменные имеют по умолчанию public static final

**public** **interface** ISwimmable {

**public** **void** swim();

}

Его наследник

**public** **class** Duck **implements** Swimmable {

**public** **void** swim() {

System.out.println("Уточка, плыви!");

}

**public** **static** **void** main(String[] args) {

Duck duck = **new** Duck();

duck.swim();

}

}

Имеет множественное наследование

### #16.Модификаторы доступа

Private, protected, defolt(по умолчанию),public

protected, defolt видны в соседних классах но не видны из других пакетов

### #17.Переменные

Volatile int a=5;позволяет нескольким потокам пользоваться ею из памяти

Transient int s=7;убирает из сериализации это поле

Final int d=7;это константа определяется сразу

Static int g=6; оно общее для всех классов

### #18.Методы.разные модификаторы

Native библиотечный метод говорит что он написал на другом я зыке более низком

Strictfp указание для формата выполнения операций с опред точностью

Sinchronised даёт к себе доступ только одному потоку

Abstract final разный подход к переопределению

Static запускается во всех классах

Передача массива в параметр

Void metod(int... j)может только в конце стоять

Вызываем метод

Class.metod(56,4,78,9,10,4)

{

For(int k:j)

{

System.out.println(k);

}

}

### #19.конструктор класса

Как и в с++

public class Run{

Run(string msg){

Sustem.out.println(msg);

//Может иметь перегрузки вт.числе и МО умолчанию

}

}

Вызовется в момент создания new

Run ran=new Run("constructor called");

### #20.массивы

Int[] array={4,7,5};

Int[][] array={{2,3},{5,7}};многомерный массив

\\\\\\\\\\из до.источн.

int[] myArray; // объявление массива

myArray = new int[10];

Нельзя менять размер

java.util.Arrays//класс для работы с массивами

void sort(int[] myArray, int fromIndex, int toIndex)//сортировка

int binarySearch(int[] myArray, int fromIndex, int toIndex, int key). Этот метод ищет элемент key в уже отсортированном массиве

Метод String toString(int[] myArray) преобразовывает массив к строке

System.out.println(Arrays.toString(array));//вывод в консоль

### #21.Enums

enum CoffeSize = {SMALL(100), MEDIUM(200), BIG(300)};

### #22.перегрузка методов

Эммммм....

### 23#конструкторы 2. This и super

переменная this ссылается на текущий экземпляр класса, в котором она используется, тогда как super — на текущий экземпляр родительского класса.

Каждый конструктор при отсутствии явных вызовов других конструкторов неявно вызывает с помощью super() конструктор без аргументов родительского класса, при этом у вас всегда остается возможность явно вызвать любой другой конструктор с помощью либо this(), либо super()

class A{

    A(){

        System.out.println("Конструктор без аргументов класса A");

    }

    A(String args){

        System.out.println("Конструктор с одним аргументом класса A");

    }

}

class B extends A{

   B(){

        this(""); // вызов конструктора с одним аргументом класса B

        System.out.println("Конструктор без аргументов класса B");

    }

   B(String args){

        super(""); // вызов конструктора с одним аргументом класса A

        System.out.println("Конструктор с одним аргументом класса B");

    }

}

// Тест-класс и вывод

public class Test {

    public static void main(String args[]) {

       B b = new B();

    }

}

Выводв консоль:

Конструктор с одним аргументом класса A

Конструктор с одним аргументом класса B

Конструктор без аргументов класса B

Super() авто вызов родительского класса без аргументов проставляется невидимо

### #24.переопределение методов

Void metod()throw Exeption{}

============

@override

Void metod()throw IQExeption{}

С разными эксепшенами будет работать

Только у наследника можно поле ресепшена оставить пустым или вызвать наследника верхнего эксепшена

### #25.Static

Static int a

Статическое поле:

static {

int a

}

Бывают статические классыОни бывают полезными для представления улучшенных связей

вложенные классы находятся в отдельном файле .class. Таким образом, если вы объявили пять вложенных классов в вашем главном классе, у вас будет 6 файлов с расширением .class.

Статические методы не переопределяются@override

### #26.блоки инициализации

Порядок запуска:

Снач запуск.статические блоки

Далее конструктор

В конструкторе выполняетсяsuper()

Запус.статические блоки Супера

Обычн.блоки Супера

И обычные блоки запускаемого класса

static{.....}//статический блок

{.....}//обычн.блок

### #27.инициализация массива

Int[] j=new int[5];

for(i=0;i<j.lenght;++i)

{System.out.println(j[i]);}

for(i=0;i<j[0].lenght;++i)//просмотр массива1го эл.двумерного массива

static void metod(int[]j){for...}//передача массива в метод

### #28.обертки

Список классов оберток:

int Integer

short Short

long Long

byte Byte

float Float

double Double

char Character

boolean Boolean

Объявление оберток:

Integer i = new Integer(682);

   Double d = new Double(2.33);

   Boolean b = new Boolean(false);

Преобр.обертки в строку:

Integer i = new Integer(432);

   String s = i.toString();

Обратный перевод:

String s = "1166628";

   Integer i = Integer.parseInt(s);

   System.out.println(i);

задача: вывести в консоль максимально возможное число int, а после — минимально возможное.

System.out.println(Integer.MAX\_VALUE); System.out.println(Integer.MIN\_VALUE);

Автоураковка,автораспаковка

int x = 7;

       Integer y = 111;

       x = y; // автораспаковка

       y = x \* 123; // автоупаковка

Работает также с различ.методами но не работает с массивами

Немного о работе с листами:

List<Integer> list=new ArrayList<>();

list.add(2);

list.add(7);

list.add(9);

for(int k:list){System.out.println(k);}

Оператор сравнения:

Integer i=4;

Integer j=8;

if(i==j)//не сработает, будет проверять по адресам памяти, хотя уже работает пофиксил за 4года

if(i.equals(j))//метод сравнения

### #29.перегрузка методов 2

public class Main {

    public static void main(String[] args) {

        byte b=5;

        Main.metod( b);//автобоксинк сам преобразует к инту

        metod((long) b);//приведение к лонгу

    }

    static void metod(int a){System.out.println("int");}

    static void metod(long a){System.out.println("long");}

    static void metod(Integer a){System.out.println("Integer");}

    static void metod(Byte a){System.out.println("Byte");}

    static void metod(Short a){System.out.println("Short");}

}

### #30.установка jdk на Линукс

Эмммм... У меня нет линукса

### #31.Установка eclipse под Линукс

У меня нет линукса...

### #32.установка lntellij idea под Линукс

.......

### #33.установка oracle jdk на Линукс

..........

### #34. Garbage Collector (сбор мусора)

//дальше какаято ерунда но оч. интересно

import java.util.Date;

public class Main {

    public static void main(String[] args) {

        metod(100\_000\_000);

    }

    private static void metod(int count){

        Runtime runtime=Runtime.getRuntime();

        System.out.println("total memory before"+runtime.totalMemory());

        System.out.println("free memory before"+runtime.freeMemory());

        for (int i = 0; i < count; i++) {

        Date date=new Date();

        date=null;

        }

        System.out.println("total memory after"+runtime.totalMemory());

        System.out.println("free memory after"+runtime.freeMemory());

        System.gc();//Garbage Collector просим систему почистить память но не заставляем потомучто она сама знает когда и что ей чистить

        System.out.println("total memory gc"+runtime.totalMemory());

        System.out.println("free memory gc"+runtime.freeMemory());

    }

    }

### #35. операторы

все работает как и в с++

Car instanceof Main//оператор проверки принадлежности

""+5+6 //преобразует в строчку

3>4 | 6<7 //проверит обе части обязательно

9<5|| 7>6 //первая часть false во вторую часть уже не пойдет

3<4 & 6<7 //проверит обе части обязательно

3<4 && 6<7 //первая часть true во вторую часть уже не пойдет

folse^true//вернет true, проверяет чтобы операторы были разные true^true //вернет false

### #36. main(String[] args)

Это параметры для запуска из консоли

### #37. if else

### #38.swith

//туда можно передавать энамы и стринги тоже

int a=3;

switch (a){

    case 1:

        System.out.println(1);

        break;

    case 2:

        System.out.println(2);

        break;

    case 3:

        System.out.println(3);

        break;

        default:

            System.out.println(break);

            break;

}

### #39. while, do while

............

### #40. For

............

### #41. Улучшенный for

int[]array={5,7,7,9};

for(int el:array){out.println(el);}

### #42. Исключения

import java.io.IOException;

public class Main {

    public static void main(String[] args) {

       new Main().someMetod();

    }

    void someMetod(){

        try {

            Object o=null;

            o.hashCode();

        }

        catch (NullPointerException e) {

            e.printStackTrace();//список вызовов

        }

        finally {//выполн.вне завис.от наличия ошибок

            //напр.если try запретил доступ отальным программам к какому либо файлу finaly должен отпустить этот файл

            System.out.println("finaly");

            System.exit(0);//прерывание программы,есть аналог вечный цикл

        }

    }

    }

    //Exception

//IOException

//NullPointerException

//RuntimeException

//trow new Error();//прерывание программы

//Эксепшена может быть несколько, но сначала идут более конкретные а потом более общие

======≠===============

void metod()throw IOExeption/\*передача проблемы наверх\*/{

throw new IOExeption();

}

### #43.Assertions

private void someMetod(int a){

assert(a>0);//проверка на а

//Do something this a

System.out.println(a);

}//Там есть проблемы с подключением ассертов

### #44.StringBuilder

Строчку невозможно изменить обычными способами

Для этого есть спец.классы

StringBuilder sb=new StringBuilder("World");

sb.append("Hello")//добавит в начало

sb.insert(5," ")//5ым символом вставит пробел

### #45.работа с файлами

6 библиотек по работе с файлами

File f=new File(temp.txt);//работает с файлами и дерикториями

if(f.isDerrictory()){};проверка не дерриктория ли это

if(f.isFile()){};//не файл ли это

if(f.exists()){};//проверка на существование

f.mkdir();//создание дерриктория

f.createNewFile();//создание файла

FileWriter fWriter=new FileWriter(f);//класс редактирования файла

fWriter.write("str1\n");

fWriter.write("str2\n");

fWriter.flush();

fWriter.close();

FileReader fReader=FileReader(f);//класс чтения считывает например в массив чаров

char[] chars=new char[20];

fReader.read(chars);

System.out.println(chars);

BufferedWriter bffWriter=new BufferedWriter(fWriter);//более удобный класс записи принимает другой класс

bffWriter.write("str3");

bffWriter.newline();

bffWriter.write("str4");

bffWriter.flush();

bffWriter.close();

BufferedReader bffReader=new BufferedReader(fReader);

while(fReader.ready()){

System.out.println(fReader.readLine());//построчно выведет

}

### #46.сериализация

//Сохранение классов на какой-то стадии выполнения программы

class User implements Serializeble{

int lifeLevel;

transient Sord sord;

}

======================

В мэйне:

User user=new User();

user.lifeLevel=55;

Сохраняем:

FileOutputStream f\_OutputStream=new FileOutputStream("tempfile");

ObjectOutputStream o\_OutputStream=new ObjectOutputStream(fOutputStream);

o\_OutputStream.writeObject(user);

o\_OutputStream.close();

Достаем обратно:

FileInputStream f\_InputStream=new FileInputStream("tempfile");

ObjectInputStream o\_InputStream=new ObjectInputStream(f\_InputStream);

User newUser=(User??) o\_InputStream.readObject();

o\_InputStream.close();

System.out.println(newUser.lifeLevel);

Сериализоваться не могут статические поля

А также те объекты классы кот.например были созданы другими людьми применяемые в твоём классе должны унаследовать implements Serializeble

Или помечены как transient

Например:

transient Sord sord;//прямо в классе

Они не будут сериализованы

### #47.сериализация наследуемых классов

class UserChild extends User implements Serialisable{

int childLevel;

//Теперь вместо User из прошлого урока впихивает везде UserChild

//Родительский коас можно не привязывать к Serialisable но тогда он будет транзиентным

}

### #48.дата.календарь.время

Data data=new Data();//считает время в миллисекундах с 1янв1970г создание линукса и юникса

System.out.println(data);//вернёт 1янв1970

Calendar calendar=Calendar.getInstanse();//к самому календарю доступа нет он приватный

calendar.setData(data);

System.out.println(calendar);выведет слишком много информации

System.out.println(calendar.getTime());//выведет норм значение

calendar.add(Calendar.WEEK\_OF\_MENTH,1);//прибавит неделю

System.out.println(calendar.getTime());

DateFormat dateFormat=DateFormat.getDateInstanse(

DateFormat.SHORT);

System.out.println(dateFormat.format(calendar.getTime()));//выберет формат даты покороче

SimpleDateFormat sdf=SimpleDateFormat("dd/MM/yyyy HH:mm");//пользовательский настраиваемый формат подробности можно загуглить

System.out.println(sdf.format(calendar.getTime()));

Date newDate=sdf.parse("14/12/1985 05:22");//опратное действие

System.out.println(newDate);

### #49.Работа с "Map",equals и HashMap

class Demo

{

class Book{

String autor;

String name;

}

public static void main (String[] args) throws java.lang.Exception

{

Book book=new Book();

book.autor="Karnegi";

book.name="How to get friends";

Map<String,Book> library=new HashMap<>();

library.put(book.name,book);//первый это ключ к таблице по которой можно найти обьект,2й это сам обьект

Book karnegiBook=library.get(book.name);//создали новый обьект и данные загрузили из library

System.out.println(karnegiBook.name);

System.out.println(book.hashCode());//вернет номер ячейки где лежит book

}

}

===========================

Equals это метод сравнения обьектов, переопределяется под конкретный объект. Он более подробно сравнивает по сравнению с hashCode()

Пример переопределения equals:

@Override

public boolean equals(Object o) {//передаем объект с которым будем сравнивать

   if (this == o) return true;

//сравниваем адреса памяти this с объектом o

   if (o == null || getClass() != o.getClass()) return false;

//проверка на null и чтобы имя типа класса совпадало

   Man man = (Man) o;

//временный обьект

   return dnaCode == man.dnaCode;

//сравнение полей и возвращение буля

}

=====================

hashCode() тоже переопределяемый метод сравнения но более быстрый, сравнивает по int значению

@Override

public int hashCode() {

   return dnaCode;//сравнивают по коду днк

}

там всего много потом ещё почитать

### #50. Коллекции.

Потерялась инфа:(

### #51.сортировка коллекции

class Person imlement Comparable<Person>{

int age;

public Person(int age){

this.age=age;

@Override

public int compareTo(Person p//перед.обьект сравниваемый с текущим об.){

return this.age-p.age;//если вернёт отриц.числ, значит this меньше

}//Теперь можно сравнивать и сортировать список об.Person

}

public String toString(){

return "Person(age="+age")";//чтобы в консоль можно было вывести

}

}

Set set=new TreeSet();

set.add(new Person(6));

set.add(new Person(2));

set.add(new Person(5));

set.add(new Person(7));

for(Object o:set){

System.out.println(o);//выведет уже отсортированный список

}

======================

Если нет доступа к классу Person то сделаем это спомощью интерфейса Comparate:

class ComparePerson implement Comparator<Person>{

@Override

public int compare(Person p1,Person p2){

return p1.age-p2.age;

}

}

ComparePerson comparePerson=new ComparePerson();

Set set=new TreeSet(comparePerson);

set.add(new Person(6));

set.add(new Person(2));

set.add(new Person(5));

set.add(new Person(7));

for(Object o:set){

System.out.println(o);

}//Должно получиться тоже самое

### #52.ArrayList. Vector. LinkedList.

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import java.util.Vector;

public class Main {

    public static void main(String[] args) {

        List list=new ArrayList();//работает с эллементами на основе индексов. у листа есть капассити

        list.add(2);

        list.add(4);

        list.add(56);

        list.add(7);

        list.add(1);

        for (int i = 0; i < list.size(); i++) {

            System.out.println(list.get(i));

        }

        List list2=new Vector();//тот же Array list но все методы синхронизированы

        List list3=new LinkedList<>();//очереди. двунаправленный список

    }

    }

### #53.Queue очередь

import java.util.LinkedList;

import java.util.PriorityQueue;

import java.util.Queue;

public class Main {

public static void main(String[] args) {

Queue queue=new LinkedList();

Queue queue1=new PriorityQueue();//отсортированная очередь

queue1.add(12);

queue1.add(4);

queue1.add(11);

queue1.add(45);

queue1.add(56);

while (queue1.size()>0){

System.out.println(queue1.poll());//выведет и удалит

}

}

}

### #54. Map.

import java.util.\*;

public class Main {

   public static void main(String[] args) {

       Map map=new HashMap();//мапы хранит ключики и их значения в качестве ключиков могут быть обьекты с переопределенным хэшкодом

       //так же важно переопределять equals

       //хешкод может совпадать тогда 2й обьект с одинак.хешкодом попадет в next чтобы их различить нужен equals

       Map map2=new Hashtable();//тотже хешмап но все методы синхронизированы

       Map map3=new LinkedHashMap();//выводит в том же порядке в котором и ложатся

       Map map4=new TreeMap();//отсортированный список строится наподобие бинарного дерева

       map.put("3","one");

       map.put("2","three");

       map.put("1","two");

       System.out.println(map.get("3"));

       Set set=map.entrySet();

       for (Object o:set) {

           System.out.println(o);

       }

       }

   }

### #55.Set

import java.util.HashSet;

import java.util.LinkedHashSet;

import java.util.Set;

import java.util.TreeSet;

public class Main {

    public static void main(String[] args) {

        Set set=new HashSet();//хранит неповторяющиеся значения.работает на основе мапа на основе хешкода

        Set se2=new LinkedHashSet();//хранит в том порятке в котором добавлялся

        Set set3=new TreeSet();//отсортированный set

        set.add("1");

        set.add("1");

        set.add("2");

        set.add("3");

        set.add("2");

        set.add("4");

        for (Object o:set){

            System.out.println(o);

        }

    }

    }

### #56.Generic. Обобщения

import java.io.File;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

public class Main<T extends Main/\*класс идет вперед интерфейсов\*/ & Comparable & Serializable>{//сделаем класс шаблонным. Теперь "T" наследуется от comparable.и классы передаваемые шаблоном должны тоже наследоваться от...

    T var;//приймет тип заданный при создании класса

    public static void main(String[] args) {

        Main main=new Main();

        main.<Integer>metod(4565);

        main.metod(4565);

        main.metod(new Main());

        /\*Main<String> gc=new Main<>();

        gc.var="Hello";//переменная тоже станет стринг

        System.out.println(gc.var);\*/

        Main<Car> car=new Main<>();

        car.var=new Car();

        car.someMetod();//теперь кару доступны методы унаслед.классов

        List<String> list=new ArrayList<>();//параметризованный лист.в этот лист пойдут только стринги

        List list1=new ArrayList();//сюда пойдет все без разбора.лучше такие листы не использовать

        list.add("string");

        list1.add(25);

        list1.add(new Object());

        List<File> flist=new ArrayList<>();

    }

    <T> T metod(T type) {

        System.out.println(type);

        return type;

    }

    void someMetod(){

        System.out.println("some metod");

    }

    <T,U,K> void/\*здесь возвращаемый тип кот.будет в return\*/ metod(T type,U u,K k){

    }

    void metod(List<File> list){

    }

    void metod1(List<? extends Parent> list/\*так.зап.можно передать наследников Parenta и \*/){    }

    void metod2(List<? super Parent> list/\*можно добавить тех кто по иерархии выше Parenta\*/){    }

}

class Parent{}

class MyClass <T>{T t;    T metod(T t){return t;}}//эта Т работает только до компиляции потом после компиляции все меняется на Object. потомучто когда делали дженерики не хотели поломать старый код

//у дженериков есь проблемы с наследованиями созданием массивов и невозможность создавать статические поля

### #57.Inner Classes

public class Main{

    private  int i=5;

    public static void main(String[] args) {

        Main main=new Main();

       //в статическом классе не получится создать вложенный клас для этого уже нужен созданный обьект внешнего класса

        /\*вот так\*/main.metod2();

        /\*или так\*/Inner inner=main.new Inner();

    }

    void  metod2(){

        Inner inner=new Inner();

        inner.metod();

        System.out.println(inner.k);

    }

    class  Inner{

        //тесно взаимосвязан с внешним классом видят все приватные поля друг друга

        //в основном используется в програмировании свифтов

        int k=6;

        void metod(){

            System.out.println(i);

        }

    }

    static class StaticClass{}//статик класс не внутренний класс и к приватным полям доступа не имеет кроме статических полей внешнего класса

    //так же бывает класс внутри метода но он очень сложен в использовании

    }

### #58. Анонимные классы

class  Popcorn{//были проблемы создать эксперементальный класс внутри осн.класса но для этого нужен экземпляр класса чтоб не заморачиваться вынес его

    void  doSomthing(){

        System.out.println("popcorn");

    }

    void SecondMetod(){

        System.out.println("second metod");

    }

}

public class Main{

    public static void main(String[] args) {

        Popcorn popcorn=new Popcorn(){

            @Override

            void doSomthing(){

                System.out.println("anonimny class");

            }

        };

        popcorn.doSomthing();

        popcorn.SecondMetod();

        //пример переопределения интерфейса

        /\*Comparable comparable1=new Comparable() {//переопределение интерфейса

            @Override

            public int compareTo(Object o) {

                return 0;

            }

        };

        comparable1.compareTo(new Object());\*/

        //пример переопределения при вызове метода в данном случае статического

        staticMetod(new Popcorn(){@Override void SecondMetod(){System.out.println("переопределили SecondMetod");}});

    }

    static  void staticMetod(Popcorn popcorn){

        popcorn.SecondMetod();

    }

    }

### #59. Клонирование

//такое гг что лучше конструктором копирования пользоваться или рой заново инет по клонированию а так чтото и получается но если дело касается вложенных переложенных классов то лучше вернуться на завод

public class Main{

    public static void main(String[] args) {

        MyObject myObject=new MyObject();

        myObject.g=1;

MyObject myObject1=null;

        try {

//создаем новый обьект null снаружи а клонируем в него старый обьект внутри trycatch

            myObject1=myObject.clone();

        }

        catch (CloneNotSupportedException e){

            e.printStackTrace();

        }

            myObject1.g=2;

System.out.println(myObject.g);

        NewObject newObject=new NewObject();

        newObject.j=3;

        myObject.newObject=newObject;

        myObject.newObject.j=4;

        System.out.println(myObject.newObject.j);

}

}

class MyObject implements Cloneable{

    int g;

    NewObject newObject;

    @Override//прав.клавишей generate->override->clone

    protected MyObject clone() throws CloneNotSupportedException {

        MyObject myObject=(MyObject) super.clone();

        myObject.newObject=newObject.clone();

        return myObject;

    }

}

class NewObject implements Cloneable {

    int j;

    @Override

    protected NewObject clone() throws CloneNotSupportedException {

        return (NewObject) super.clone();

    }

}

### #60.Reflection - рефлексия

//используется в основном для написания библиотек

//списывал с достаточно старого видео и там не хватает кучу эксепшенов будут ошибки

import java.lang.reflect.\*;

public class Main {

    public static void main(String[] args) {

        SomeClass someClass=new SomeClass();//рассматриваем основные способы получить доступ к чужому классу из библиотеки

        Class clss=someClass.getClass();

        Class clss2=SomeClass.class;

        //Class clss3=Class.forName("NewPackage");

        //Возможности:

        SomeClass someClass1=(SomeClass)clss.newInstance();

        Constructor[] constructors=clss.getDeclaredConstructors();

        for (Constructor constructor:constructors){

            System.out.println(constructor.getName());

            Parameter[] parameters=constructor.getParameters();

            for (Parameter parameter:parameters){

                System.out.println(parameter.getName());

                System.out.println(parameter.getType().getName());

            }

        }

        System.out.println("");

        System.out.println("Metods:");

        Method[] methods=clss.getDeclaredMethods();

        for (Method method:methods){

            System.out.println(method.getName());

            Parameter[] parameters=method.getParameters();

            for (Parameter parameter:parameters){

                System.out.println(parameter.getName());

                System.out.println(parameter.getType().getName());

            }

            System.out.println(Modifier.toString(method.getModifiers()/\*в чистом виде выдаст int\*/));

            method.setAccessible(true);//ломаем приватность метода

            method.invoke(someClass,"test");//запуск метода с передачей туда параметра

        }

        System.out.println("");

        System.out.println("filds:");//филды это имена пременных "i" "s"

        Field[] fields=clss.getDeclaredFields();

        for (Field field:fields){

            System.out.println(field.getName());

            System.out.println(field.getType().getName());

            System.out.println(Modifier.toString(field.getModifiers()));

            //пытаемся изменить приватное поле

            field.setAccessible(true);//убрали приватность поля

            System.out.println(field.getInt(someClass));

            field.setInt(someClass,5);

            System.out.println(field.getInt(someClass));

        }

    }

}

class SomeClass{

    private int i;

    public SomeClass(){}

    SomeClass(String s){    }

    private String SomeMetod(String s){

        System.out.println("this is "+s);

        return s;

    }

}

### #61. технологии в  програмировании

//java 8 лямбда выражения оч.часто встречается

//три направления swing редко используется

//android

//web обязательный работа с базами данных

//git система контроля версии

//maven консольный сборщик приложения

//юнит тесты JUnit

//Hibernate для работы с sql

//spring набор библиотек

//sql

//html

//css

//javascript

//

### #62.запуск через командную строку

//помещаем нужный класс Main.java в проект эклипса

//открываем эклипс

//открываем проект и класс

//file->Export->java->Runnable JAR File-> тапаем некст

//выбираем проект и выбирам папку для будущего jar файла из короткой менюшки проверить кружочек в первом выборе ->Finish

//создался Main.jar

//копируем путь к папке с файлом бежим в консоль

//там после C:\Users\222>  пишем "cd "  (пробел обязательно без ковычек) вставляем путь до папки C:\Users\222\Desktop\MyProject

//тапаем Enter

//появляется C:\Users\222\Desktop\MyProject>  поменяли дерикторию

//запускаем jar C:\Users\222\Desktop\MyProject>java -jar Main.jar (там два пробела)

//появляется Hello World

//рядом с файлом создаем текстовый файл записываем в него строку java -jar Main.jar файл>>сохранить как>>Main.bat (простое переименование не прокатит)

//создался системный файлик который автоматически вызовет консоль и jar файл

УРАААААА

### #63.Про собеседование

.....................

### #64.Формирование проектов из джар файлов с папками и пакетами

ничего не понятно

### #65. чтение из консоли

import java.io.Console;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        /\*Console console=System.console();

        String s="";

        while (!s.equals("exit")){

            s=console.readLine();

            System.out.println(s);

        }\*/ //с этой проблемы лучше 2й вариант

        Scanner scanner=new Scanner(System.in);

                String s="";

        while (!s.equals("exit")){

            s=scanner.nextLine();

            System.out.println(s);

        }

    }

}

### #66.Мноогопоточность

public class Main {

    public static void main(String[] args) {

        System.out.println(Thread.currentThread().getName()+": "+Thread.currentThread().threadId()+" поток Main");

        MyThread myThread=new MyThread();

        myThread.start();//этод метод вызовет метод run()

        //=========способ с Runnable============//

        MyRunnable myRunnable=new MyRunnable();

        Thread thread2=new Thread(myRunnable);

        thread2.start();

        }

}

class  MyThread extends Thread{

    @Override

    public void run() {

        System.out.println("this is my new thread");

        System.out.println(Thread.currentThread().getName()+": "+Thread.currentThread().threadId()+" поток из MyThread");

    }

    /\*private void someMethod{

        throw new RuntimeException();

    }\*/

}

class  MyRunnable implements Runnable{

    @Override

    public void run() {

        System.out.println(Thread.currentThread().getName()+": "+Thread.currentThread().threadId()+" поток из Runnable");

    }

}

### #67.Многопоточность. жизненный цикл потоков

public class Main {

    public static void main(String[] args) {

        MyThread myThread = new MyThread();

        myThread.start();

        myThread.interrupt();

        myThread.setPriority(10);//выставляет приоритет от 0 до максимального10

        Thread.yield();//заставляет процессор переключиться на обработку других, менее приоритетных потоков,но это не точно, подумает и снова начнет выполнять этот поток

myThread.join();//подождёт завершения потока myThread

        try {Thread.sleep(5000);/\*уснет на 1 секунду\*/}catch (InterruptedException e){e.printStackTrace();/\*если вызвать myThread.interrupt(); до трай кэча то sleep не выполнится но у меня не заработало\*/}

        System.out.println(Thread.currentThread().getName() + ": " + Thread.currentThread().threadId() + " поток Main");

    }

}

class  MyThread extends Thread{

    @Override

    public void run() {

        System.out.println("this is my new thread");

        System.out.println(Thread.currentThread().getName()+": "+Thread.currentThread().threadId()+" поток из MyThread");

    }

}

### #68. Синхронизация

Защита разделяемых ресурсов

Есть методы и поля synchronized

public synchronized void chengeI(){

public void chengeI(){

synchronized(this){

......./\*синхронизированное поле\*/

......

}

}

### #69.синхронизация статических полей

class MyThread exstends Thread{

Resurse resurse;

@Override

public void run(){

Resurse.changeStaticI();//Статик синхр метод

new Resurse().changeI();//нестатик синхр метод будут выполняться одновременно

}

}

class Resurse{

static int i;

public synchronized void changeStaticI(){

}

public void changeStaticI(){

synchronized(Resurse.class){

/\*статический блок синхронизации

Синхронизируется на уровне класса

Можете выполняться одновременно с нестатическим синхронизированный блоком\*/

}

}

}

### #70.Многопоточность.volatile

Если мы объявляем в нашей программе какую-то переменную, со словом volatilе это означает, что:

1.Она всегда будет атомарно читаться и записываться. Даже если это 64-битные double или long.

2.Java-машина не будет помещать ее в кэш. Так что ситуация, когда 10 потоков работают со своими локальными копиями исключена.

Пример:

public class Main {

   public volatile long x = 2222222222222222222L;

   public static void main(String[] args) {

   }

}

Потоки кешируют переменную в свою память и там проводят с ней операции в этот момент другой поток может из основного main потока считывать уже неактуально переменную

А с переменной volatile поток будет проводить манипул.прямо в памяти main

### #71.Многопоточность. атомарные переменные

Такие операции как I++ например не атомарны, и при одновременном использовании на одной переменной разными потоками будут неточности

Т.к. i++ это не одна операция а 3 за раз

Можно использовать sinchronized и volatile но долго по времени работает

Есть CAS но я не знаю что это такое

AtomicBoolean, AtomicInteger, AtomicLong, AtomicIntegerArray, AtomicLongArray —классы в которых операции атомарны.лучший вариант

Там есть методы геттеры сеттеры инкремента декременты

### #72.многопоточность.синхронизированные коллекции

Java concurrency package искать на oracle набор плюшек оберток для работы с потоками

List list=collection.sinchronizedList(new ArrayList());

Короче ерунда. Все равно лучше просто синхронизировать методы

Единственное поможет при возвращении листа

public void getList(){return list;} и то лучше по возможности клонировать а не возвращать

### #73.многопоточность. Deadlock

Взаимная блокировка потоков

При вызове друг у друга залоченных ресурсов

### #74.многопоточность. wait и notify

Командой weit мы просим один поток подождать другой когда он выполнит код и дойдет до notify

Оба метода должны вызываться из synchronized

import java.util.\*;

import java.lang.\*;

import java.io.\*;

/\* Name of the class has to be "Main" only if the class is public. \*/

class Demo

{

public static void main (String[] args) throws java.lang.Exception

{

MyThread myThread= new MyThread ();

myThread.start();

synchronized (myThread){myThread.wait();}//просим подождать поток myThread

System.out.println(myThread.total);

}

}

class MyThread extends Thread{

int total;

@Override

public void run(){

synchronized(this){

for(int i=0;i<100;++i){

total+=i;

//try{sleep(500)}catch(InterruptedExeption e){}

}

notify();//поток говорит что закончил работу

}

}

}

### #75.Wait и notify продолжение поточное голодание

Что такое поточное голодание???

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Main {

    static List<String> strings=new ArrayList<>();

    public static void main(String[] args) {

        new Operator().start();

        new Machine().start();

      }

      static  class Operator extends Thread{

        @Override

          public void run() {

            Scanner scanner = new Scanner(System.in);

            while (true) {

                synchronized (strings) {

                strings.add(scanner.nextLine());

                strings.notify();

                }

                try {Thread.sleep(1000);}catch (InterruptedException e){e.printStackTrace();}

            }

        }

      }

    static  class Machine extends Thread{

        @Override

        public void run(){

                while (strings.isEmpty()) {

                    synchronized (strings) {

                    try {System.out.println("поток ждет вашего сообщения");

                        strings.wait();} catch (InterruptedException e) { e.printStackTrace();}

                    System.out.println("ваше сообщение: "+strings.remove(0));

                }

            }

        }

    }

}

### #76. Многопоточность. ReenrantLock

Аналог synchronized но только более гибкий может например в одном методе вызывать блокировку а в другом снимать

class Test{

Lock lk=new ReetrantLock();

public void metod(){

lk.lock();//заблокировали

/\*Какой-то

код\*/

metod2();

}

public void metod2(){

/\*Какой-то

код\*/

lk.unlock();//разблокировали в другом методе

}

### #76. Многопоточность. ReenrantLock

Аналог synchronized но только более гибкий может например в одном методе вызывать блокировку а в другом снимать

class Test{

Lock lk=new ReetrantLock();

public void metod(){

lk.lock();//заблокировали

/\*Какой-то

код\*/

metod2();

}

public void metod2(){

/\*Какой-то

код\*/

lk.unlock();//разблокировали в другом методе

}

### #77.Многопоточность. TryLock

//Тот же Лок но только попытка взять ограничивается по времени

// класс для работы с Lock API. Переписан с приведенной выше программы,

// но уже без использования ключевого слова synchronized

public class ConcurrencyLockExample implements Runnable{

    private Resource resource;

    private Lock lock;

    public ConcurrencyLockExample(Resource r){

        this.resource = r;

        this.lock = new ReentrantLock();

    }

    @Override

    public void run() {

        try {

            // пытаемся взять лок в течении 10 секунд

            if(lock.tryLock(10, TimeUnit.SECONDS)){

            resource.doSomething();

            }

        } catch (InterruptedException e) {

            e.printStackTrace();

        }finally{

            //убираем лок

            lock.unlock();

        }

        // Для логгирования не требуется потокобезопасность

        resource.doLogging();

    }

}

### #78.Многопоточность. Condition\пересмотреть

С sinchronized мы используем wait и notify а с Lock используем Condition

Очень похоже на wait и notify

Есть 3 команды await/signal/signalAll

import java.lang.\*;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

class Demo {

    static Lock lk = new ReentrantLock();

    static Condition cdt = lk.newCondition();

    static int account = 0;

    public static void main(String[] args) throws java.lang.Exception {

        new AccountMinus().start();

        new AccountPlus().start();

    }

    static class AccountPlus extends Thread {

        @Override

        public void run() {

            lk.lock();

            account += 10;

            cdt.signal();

            lk.unlock();

        }

    }

    static class AccountMinus extends Thread {

        @Override

        public void run() {

            lk.lock();

            System.out.println("account= "+account);

            if (account <= 10) {try {cdt.await();} catch (InterruptedException e) {throw new RuntimeException(e);}}

            System.out.println("account= "+account);

            account -= 10;

            lk.unlock();

        }

    }

}

### #79.Callable and Futures//пересмотреть

Аналоги Runnable

Используются чтобы вернуть из потока какое либо значение "return"

Callable очень похож на runnable имеет метод call вместо run возвращает значение "return" используем дженерики

Futures вспомогательный класс

Помогает достать из потока результат методом get и проверить наличие ошибок

Методы интерфейса Future

Метод Описание

cancel (boolean mayInterruptIfRunning) попытка завершения задачи

V get() ожидание (при необходимости) завершения задачи, после чего можно будет получить результат

V get(long timeout, TimeUnit unit) ожидание (при необходимости) завершения задачи в течение определенного времени, после чего можно будет получить результат

isCancelled() вернет true, если выполнение задачи будет прервано прежде завершения

isDone() вернет true, если задача завершена

### #80.многопоточность.Executur

Фреймворк по управлению пулом потоков

три интерфейса: Executor, ExecutorService и ScheduledExecutorService.

Executor — простой интерфейс, содержит метод execute() для запуска задачи, заданной запускаемым объектом Runnable.

ExecutorService суб-интерфейс Executor,  метод submit()аналогичен  execute(), но более универсален. Перегруженные версии метода submit() могут принимать как (Runnable), так и (Callable), также как callable может возвращать значение. Поэтому, если мы передаем объект Callable методу submit(), он возвращает объект Futurе.

ScheduledExecutorService, это суб-интерфейс ExecutorService. Он добавляет функциональность, планирует выполнение задач в коде.

класс Executors, который по умолчанию включает в себя методы для создания различных типов служб-исполнителей. С помощью этого класса и интерфейсов можно создавать пулы потоков.

В классе Executors есть четыре основных метода, которые используются для создания пулов потоков.

Вот они:

->newSingleThreadExecutor(), будет создан только один новый поток и одновременно будет выполняться только одна задача.//shutdown() ожидает завершения выполнения задач, в настоящий момент переданных исполнителю, чтобы завершить его работу. Однако, если вам хочется завершить работу исполнителя без ожидания, используйте shutdownNow().//

->newFixedThreadPool() фиксированное количество потоков. в коде будет создано три новых потока три задачи. Остальные две задачи находятся в очереди ожидания.

->newCachedThreadPool() максимальный размер пула устанавливается на максимальное целочисленное значение в Java.Если поток бездействии более минуты, метод устраняет его.

NOP::Nuances of Programming

Published in

NOP::Nuances of Programming

Рудольф Коршун

Рудольф Коршун

Follow

Sep 15, 2020

·

6 min read

Save

Фреймворк Executor в Java

Вплоть до Java 5 создавать потоки и управлять ими было возможно только на уровне приложения. Объекты Thread требуют значительного объема памяти. Таким образом, если постоянно создавать много таких объектов в крупномасштабном приложении, это приведет к существенным затратам памяти. Поэтому создание потоков и управление ими лучше отделить от остальной части приложения.

В качестве решения Java предоставляет фреймворк Executor. Он содержит множество функций для эффективного управления несколькими потоками. Не нужно каждый раз создавать новые потоки с помощью Executor — он позволяет использовать уже созданные потоки, когда вам это необходимо. В результате экономится как память вашего Java-приложения, так и ваше драгоценное время.

Из этой статьи вы узнаете о фреймворке Executor, пуле потоков и различных методах их создания, а также о том, как с их помощью управлять потоками. Давайте начнем.

Интерфейсы и классы Executor

Во фреймворке Executor присутствуют три интерфейса: Executor, ExecutorService и ScheduledExecutorService.

Executor — простой интерфейс, содержащий метод execute() для запуска задачи, заданной запускаемым объектом Runnable.

ExecutorService представляет собой суб-интерфейс Executor, который добавляет функциональность для управления жизненным циклом потоков. Он также включает в себя метод submit(), который аналогичен методу execute(), но более универсален. Перегруженные версии метода submit() могут принимать как выполняемый (Runnable), так и вызываемый (Callable) объект. Вызываемые объекты аналогичны выполняемым, за тем исключением, что задача, определенная вызываемым объектом, также может возвращать значение. Поэтому, если мы передаем объект Callable методу submit(), он возвращает объект Future. Этот объект можно использовать для получения возвращаемого значения Callable и управления статусом как Callable, так и Runnable задач.

ScheduledExecutorService, в свою очередь, — это суб-интерфейс ExecutorService. Он добавляет функциональность, которая позволяет планировать выполнение задач в коде.

Помимо трех вышеупомянутых интерфейсов, Executor Framework также содержит класс Executors, который по умолчанию включает в себя методы для создания различных типов служб-исполнителей. С помощью этого класса и интерфейсов можно создавать пулы потоков. Что же это такое?

Пулы потоков

Пул потоков — это набор объектов Runnable и постоянно работающих потоков. Коллекция объектов Runnable называется рабочей очередью. Постоянно запущенные потоки проверяют рабочий запрос на наличие новой работы, и если новая работа должна быть выполнена, то из рабочей очереди будет запущен объект Runnable. Чтобы использовать фреймворк Executor, нам нужно создать пул потоков и отправить туда задачу для выполнения. В классе Executors есть четыре основных метода, которые используются для создания пулов потоков. Рассмотрим каждый из них на примере.

newSingleThreadExecutor()

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class MyRunnable implements Runnable {

    private final String task;

    MyRunnable(String task) {

        this.task = task;

    }

    @Override

    public void run() {

        for (int i = 0; i < 3; i++) {

           System.out.println("Executing "+ task +" with "+Thread.currentThread().getName());

        }

        System.out.println();

    }

}

public class Exec1 {

    public static void main(String[] args) {

        ExecutorService executor = Executors.newSingleThreadExecutor();

        for (int i = 1; i <= 5; i++) {

            Runnable worker = new MyRunnable("Task" + i);

            executor.execute(worker);

        }

        executor.shutdown();    /\* После этого исполнитель перестанет принимать какие-либо новые потоки и завершит все существующие в очереди \*/

    }

}

В данном примере мы отправляем на исполнение пять задач. Но так как применяется метод newSingleThreadExecutor(), будет создан только один новый поток и одновременно будет выполняться только одна задача. Остальные четыре задачи находятся в очереди ожидания. Как только задача выполнится потоком, этот поток тут же выберет и выполнит следующую. Метод shutdown() ожидает завершения выполнения задач, в настоящий момент переданных исполнителю, чтобы завершить его работу. Однако, если вам хочется завершить работу исполнителя без ожидания, используйте вместо этого метод shutdownNow().

newFixedThreadPool()

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class MyRunnable implements Runnable {

    private final String task;

    MyRunnable(String task) {

        this.task = task;

    }

    @Override

    public void run() {

        for (int i = 0; i < 3; i++) {

           System.out.println("Executing "+ task +" with "+Thread.currentThread().getName());

        }

        System.out.println();

    }

}

public class Exec2 {

    public static void main(String[] args) {

        ExecutorService executor = Executors.newFixedThreadPool(3);

        for (int i = 1; i <= 5; i++) {

            Runnable worker = new MyRunnable("Task" + i);

            executor.execute(worker);

        }

        executor.shutdown();

    }

}

Использован тот же пример, что и в предыдущем случае, только на этот раз — с методом newFixedThreadPool(). Этот метод позволяет создать пул с фиксированным количеством потоков. Таким образом, когда мы отправим пять задач, в коде будет создано три новых потока и будут выполнены три задачи. Остальные две задачи находятся в очереди ожидания. Как только какая-либо задача выполнится потоком, этим же потоком будет выбрана и выполнена следующая задача.

newCachedThreadPool()

Когда мы создаем пул потоков с помощью этого метода, максимальный размер пула потоков устанавливается на максимальное целочисленное значение в Java. Этот метод создает новые потоки по запросу и разрушает потоки, которые простаивают больше минуты, если запрос отсутствует.

ПримерCachedThreadPool():

class MyRunnable implements Runnable {

    private String task;

    MyRunnable(String task){this.task=task;}

    @Override

    public void run() {

        for (int i = 0; i < 3; i++) {

           System.out.println("Executing "+ task +" with "+Thread.currentThread().getName());

        }}}

public class Exec {

    public static void main(String[] args) {

        ExecutorService executor = Executors.newCachedThreadPool();

        for (int i = 1; i <= 5; i++) {

            Runnable worker = new MyRunnable("Task" + i);

            executor.execute(worker);

        }

        executor.shutdown();

    }}

### #81.многопоточность.newScheduledThreadPool()

создает пул потоков, который может планировать выполнение задач после заданной задержки или через регулярные промежутки времени. Этот метод возвращает ScheduledExecutorService.

Метод scheduled()

ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool();

scheduledThreadPool.schedule(task,3, TimeUnit.SECONDS); //принимаемая задача, задержка запуска,тип единиц времени

scheduledThreadPool.shutdown();//закрываем

Там ещё есть методы

### #82.Semafore

Чаще всего, семафоры необходимы, когда нужно ограничить доступ к некоторому общему ресурсу.

class Demo

{

public static void main (String[] args) throws java.lang.Exception

{

Semaphore table=new Semaphore(2);//сидеть за столиком могут не больше 2х

Person person1=new Person();

Person person2=new Person();

Person person3=new Person();

Person person4=new Person();

Person person5=new Person();

Person person6=new Person();

Person person7=new Person();

person1.table=table;

person2.table=table;

person3.table=table;

person4.table=table;

person5.table=table;

person6.table=table;

person7.table=table;

person1.start();

person2.start();

person3.start();

person4.start();

person5.start();

person6.start();

person7.start();

}

}

class Person extends Tread{

Semaphore table;

@Override

public void run(){

System.out.println(this.getName()+" ждет столик");

try{

table.acquire();//перев.приобретать

System.out.println(this.getName()+" кушает");

this.sleep(1000);

System.out.println(this.getName()+" освобождает столик");

table.release();//перев.освобождает

}catch(интераптед е){}

}

}

### #83.CountDownLatch

примером CountDownLatch из жизни может служить сбор экскурсионной группы: пока не наберется определенное количество человек, экскурсия не начнется

import java.util.concurrent.CountDownLatch;

public class Main {

    public static void main(String[] args)throws InterruptedException{

        CountDownLatch countDownLatch = new CountDownLatch(3);

        Thread.sleep(2000);

        Work work1=new Work(countDownLatch);

        Work work2=new Work(countDownLatch);

        Work work3=new Work(countDownLatch);

        work3.join();

        work2.join();

        work1.join();

        System.out.println("all job done");

    }

    }

    class Work extends Thread {

    CountDownLatch countDownLatch;

    public Work(CountDownLatch countDownLatch) {

        this.countDownLatch = countDownLatch;

        start();

    }

    @Override

    public void run() {

        try {

            sleep(3000);

            System.out.println("work is ready");

            countDownLatch.countDown();

            sleep((int) (Math.random() \* 5000 + 50));

            countDownLatch.await();

        } catch (InterruptedException e) {

        }

        System.out.println("work is finished");

    }

}

### #84.Exchenger.Обменник

Обмен информацией между потоками например строками

используется один метод exchange()

import java.util.concurrent.Exchanger;

public class Main{

    public static void main(String[] args) throws InterruptedException{

        Exchanger<String> exchanger=new Exchanger<>();

        new Mike(exchanger);

        new Anket(exchanger);

    }

    static class Mike extends Thread {

        Exchanger<String> exchanger;

        public Mike(Exchanger<String> exchanger) {

            this.exchanger=exchanger;

            start();

        }

        @Override

        public void run() {

            try {

                exchanger.exchange("My name is Mike");

                sleep(2000);

                exchanger.exchange("I'm 20 years old");

            }catch (InterruptedException e){e.printStackTrace();}

        }

    }

    static class Anket extends Thread{

        Exchanger<String> exchanger;

        public Anket(Exchanger<String> exchanger) {

            this.exchanger=exchanger;

            start();

        }

        @Override

        public void run() {

            try {

                System.out.println(exchanger.exchange(null));

                System.out.println(exchanger.exchange(null));

            }catch (InterruptedException e){e.printStackTrace();}

        }

    }

}

### #85.Многопоточность.CyclicBarrier

Усовершеннствованная версия CoundDownLatch

Неограниченное количеству потоков будет накапливаться у барьера berrier.await(); и отпускаться скажем по 3 шт

 и потоки продолжат выполнять свой код дальше

когда набралась партия перед отпуском есть возможность выполнить какое либо действие например сообщение об отправке

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.BrokenBarrierException;

public class Main {

    public static void main(String[] args)throws InterruptedException{

        Runnable action = () -> System.out.println("На старт!");

        CyclicBarrier berrier = new CyclicBarrier(3, action);//action необязательное действие, но оно показывает на готовность потоков к старту

        //3 это количество максимальное потоков на старте

        Runnable task = () -> {

            try{

            berrier.await();

            System.out.println("Finished");

            }

            catch(BrokenBarrierException | InterruptedException e){e.printStackTrace();}

        };

        System.out.println("Limit: " + berrier.getParties());

        for (int i = 0; i < 19; i++) {new Thread(task).start();}

    }

}

### #86.Многопоточность. Phaser

Делает то что предыдущие но ..

Есть возможность изменить число встречающих

Регистрировать новые потоки

И удалите отработавшие циклы

import java.util.ArrayList;

import java.util.concurrent.Phaser;

public class Bus {

    private static final Phaser PHASER = new Phaser(1);//Сразу регистрируем главный поток

    //Фазы 0 и 6 - это автобусный парк, 1 - 5 остановки

    public static void main(String[] args) throws InterruptedException {

        ArrayList<Passenger> passengers = new ArrayList<>();

        for (int i = 1; i < 5; i++) {           //Сгенерируем пассажиров на остановках

            if ((int) (Math.random() \* 2) > 0)

                passengers.add(new Passenger(i, i + 1));//Этот пассажир выходит на следующей

            if ((int) (Math.random() \* 2) > 0)

                passengers.add(new Passenger(i, 5));    //Этот пассажир выходит на конечной

        }

        for (int i = 0; i < 7; i++) {

            switch (i) {

                case 0:

                    System.out.println("Автобус выехал из парка.");

                    PHASER.arrive();//В фазе 0 всего 1 участник - автобус

                    break;

                case 6:

                    System.out.println("Автобус уехал в парк.");

                    PHASER.arriveAndDeregister();//Снимаем главный поток, ломаем барьер

                    break;

                default:

                    int currentBusStop = PHASER.getPhase();

                    System.out.println("Остановка № " + currentBusStop);

                    for (Passenger p : passengers)          //Проверяем, есть ли пассажиры на остановке

                        if (p.departure == currentBusStop) {

                            PHASER.register();//Регистрируем поток, который будет участвовать в фазах

                            p.start();        // и запускаем

                        }

                    PHASER.arriveAndAwaitAdvance();//Сообщаем о своей готовности

            }

        }

    }

    public static class Passenger extends Thread {

        private int departure;

        private int destination;

        public Passenger(int departure, int destination) {

            this.departure = departure;

            this.destination = destination;

            System.out.println(this + " ждёт на остановке № " + this.departure);

        }

        @Override

        public void run() {

            try {

                System.out.println(this + " сел в автобус.");

                while (PHASER.getPhase() < destination) //Пока автобус не приедет на нужную остановку(фазу)

                    PHASER.arriveAndAwaitAdvance();     //заявляем в каждой фазе о готовности и ждем

                Thread.sleep(1);

                System.out.println(this + " покинул автобус.");

                PHASER.arriveAndDeregister();   //Отменяем регистрацию на нужной фазе

            } catch (InterruptedException e) {

            }

        }

        @Override

        public String toString() {

            return "Пассажир{" + departure + " -> " + destination + '}';

        }

    }

}

int register() — регистрирует нового участника, который выполняет фазы. Возвращает номер текущей фазы;

int getPhase() — возвращает номер текущей фазы;

int arriveAndAwaitAdvance() — указывает что поток завершил выполнение фазы. Поток приостанавливается до момента, пока все остальные стороны не закончат выполнять данную фазу. Точный аналог CyclicBarrier.await(). Возвращает номер текущей фазы;

int arrive() — сообщает, что сторона завершила фазу, и возвращает номер фазы. При вызове данного метода поток не приостанавливается, а продолжает выполнятся;

int arriveAndDeregister() — сообщает о завершении всех фаз стороной и снимает ее с регистрации. Возвращает номер текущей фазы;

int awaitAdvance(int phase) — если phase равно номеру текущей фазы, приостанавливает вызвавший его поток до её окончания. В противном случае сразу возвращает аргумент.

### #87.многопоточность.BlockingQueue

Queue<String>queue=new PriorityQueue(); не безопасно для потоков

BlockingQueue<String>queue=new PriorityBlockingQueue(); безопасен для потоков

обеспечивает блокировку потока в двух случаях :

при попытке получения элемента из пустой очереди;

при попытке размещения элемента в полной очереди.

Так же содержит кое какие методы

### #88.многопоточность.FreadFactory

С помощью фабрики создания потоков ThreadFactory можно делать разные настройки создаваемым потокам, например, – выставить приоритеты, использовать подклассы потоков, добавить поток

Мало инфы

ExecutorService executorService = Executors.newSingleThreadExecutor(new ThreadFactory() {

            @Override

            public Thread newThread(Runnable r) {

                Thread thread = new Thread(r, "MyThread");

                thread.setPriority(Thread.MAX\_PRIORITY);

                return thread;

            }

        });Переопределили метод создания нового потока, передав в конструктор имя.Поменяли имя и приоритет для создаваемого потока.

Из видео Ютуба:

ThreadFactory threadFactory=new ThreadFactory(){

            @Override

            public Thread newThread(Runnable r) {

                Thread thread = new Thread(r); thread.setPriority(Thread.MAX\_PRIORITY);

                return thread;

            }

threadFactory.newThread(new MyClass).start();создали новый поток с максимальным приоритетом

### #89.многопоточность.ForkJoinFramevork

/\* package whatever; // don't place package name! \*/

import java.util.\*;

import java.lang.\*;

import java.io.\*;

/\* Name of the class has to be "Main" only if the class is public. \*/

class Demo

{

public static void main (String[] args) throws java.lang.Exception

{

static long numOfOperations=10\_000\_000\_000L;

System.out.println(new Date);

ForkJoinPool pool=new ForkJoinPool(numOfThreads);

System.out.println(pool.invoke(new MyFork(0,numOfOperations)));

System.out.println(new Date);

}

}

static class MyFork extends RecursiveTask<long>{

long from,to;

public MyFork(long from,long to){

this from=from;

this to=to;

}

@Override

protected long compute(){

if((to-from)<=numOfOperations/numOfThreads){

long j=0;

for(long i=from;i<to;i++){

j+=i;

}

return j;

}

else{

long middle=(to+from)/2;

MyFork firstHalf=new MyFork(from,middle);

firsHalf.fork();

MyFork secondHalf=new MyFork(middle+1,to);

long secondValue=secondHalf.comute();

return firstHalf.join()+secondValue;

}

}

}

Fork делит длинную операцию на несколько потоков рекурсивно ориентируясь на количество ядер машины

### #90.многопоточность.TimeUnit и ThreadLokalRandom

Thread.sleep(TimeUnit.DAYS.toSecond(14));//покажет сколько миллисекунд в 14 днях

System.out.println(Math.random());

System.out.println(ThreadLokalRandom.current().nextInt());для многопоточки лучше использовать этот метод.там есть методы для других типов переменных

### #91.Dinamic Proxy

//Динамический прокси нужен если заранее не знаешь, какие интерфейсы будет использовать программа и не знаешь что имплементировать

InvocationHandler handler=new MyProxy(new Integer(5));

//создаём массив классов

Class[] classes=new Class[]{Compareble.class,Calleble.class};

//Создаём новый обьек прокси

Object proxy=Proxy.newProxyInstanse(null,classes,handler);//1й парам это Class loader?? 3йэто тот объект кот.будет реализовать наш новый интерфейс

proxy.getClass();//теперь прокси имеет доступ ко всем методам

((Compare)proxy).compareTo(5);//приведение прокси к определенному интерфейсу

System.out.println((Compare)proxy).compareTo(3);//сравнит переданный в аргумент интеджер 5ку с 3.

proxy.toString()

}

Static class MyProxy implements invocationHandler{

Object target;

public MyProxy(Object target){

this.target=target;

}

@Override

Public Object invoike(Object proxy,Metod metod,Object[]args)throws Throw able{

System.out.println(args);//просто выведет аргументы

//!!!при вызове метода invoke() в программе сначала выполнится наш добавочный код напр.какаянибудь проверка

return metod.invoce(target, args);//а потом в return уже вызовется оригинальный метод invoke

}

}

}

Динамические прокси это путь к аспектно ориентированному программированию

Например когда у нас при вызове различных простых методов типа toString() и другие постоянно приходится делать различные проверки типа забронирован пользователь или нет а потом уже выполнять код метода

Чтобы код был чище мы вызываем эти методы через динамический прокси

Цель (target) — объект, к которому будут применяться советы.

### #92.литература для чтения

........

### #93.Уроки на github

<https://github.com/hacker85/JavaLessons/tree/master/src>

### #94юзарплаты джавистов

<https://career.habr.com/salaries>

### #95.как получить максимальную зарплату

..........

### #96. Swing

java1.0 была библиотека awt натимные кнопки возникали проблемы между разными платформами

java 1.1 вышла swing использовала не натимные а уже сама все прорисовывала

в swing  ко всем похожим функциям подобавляли букву J    например Button and JButton

import javax.swing.\*;

import java.awt.\*;

public class MySwing {

    public static void main(String[] args) {

        JFrame jFrame=new JFrame(){};//создали анонимный класс

        jFrame.setVisible(true);//запускаем

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);//закрывающая функция

//JFrame.setSize(500,300);//задаем размер без нее она будет очень маленькая

//JFrame.setLocation(500,400);//место появления чтобы не в углу появлялась

//JFrame.setBounds(500,500,500,500);//заменяет предыдущие две команды.Первые 2 параметра это локация 2е это размер окна

        Toolkit toolKit=Toolkit.*getDefaultToolkit*();//набор инструментов для работы с  нашими фреймами

        Dimension dimension=toolKit.getScreenSize();//получаем размер экрана//dimension переводится -измерение

        jFrame.setBounds(dimension.width/2-250,dimension.height/2-250,500,500);//автоматическая ориентация

        jFrame.setTitle("this my add");//cообщение вверху окна

//jFrame.setIconImage();//добавит картинку

    }

}

### #97.JComponent

import javax.swing.\*;

import java.awt.\*;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.net.URL;

public class JComponentLesson {

    public static void main(String[] args) {

        JFrame jFrame = *getFrame*();

        jFrame.add(new MyComponent());

    }

    static class MyComponent extends JComponent {

        @Override

        public void paint(Graphics g) {

            Graphics2D g2 = (Graphics2D) g;//преобразование вместо устаревшего родственного класса в параметрах"Graphics g"

            Font font = new Font("Times New Roman", Font.*BOLD*/\*толщина\*/, 20/\*размер\*/);//отвечает за шрифт

            g2.setFont(font);//устанавливаем настройки шрифта в g2

            g2.drawString("hello world", 50, 50);

            Point2D p1 = new Point2D.Double(70, 70);

            Point2D p2 = new Point2D.Double(170, 170);

            Line2D line2D = new Line2D.Double(p1, p2);//рисуем линию можно вместо этих 2х параметров вбить 4 параметра с координатами

            g2.draw(line2D);

            Ellipse2D el = new Ellipse2D.Double(70, 70, 170, 170);//последние 2 парам это не координады а ширина и высота

            el.setFrameFromDiagonal(p1, p2);//самонастройка фигуры по точкам работает и для квадрата и для линии

            g2.setPaint(Color.*red*);

            g2.setPaint(new Color(40, 215, 74));//настройка цвета по градиентам

            g2.draw(el);

            g2.fill(el);//закрасит территорию

            Rectangle2D rectangle2D = new Rectangle2D.Double(70, 70, 170, 170);

            rectangle2D.setFrameFromDiagonal(p1, p2);

            g2.draw(rectangle2D);

            URL url = null;

            try {

                url = new URL("https://crosti.ru/patterns/00/1d/a7/b6\_preview\_a5ce310f.jpg");//картинка оказалась в слишком большом расширении

            } catch (java.net.MalformedURLException ignored) {

            }

            Image image = new ImageIcon(url).getImage();//создаем картинку по URL

            Image image2 = new ImageIcon("C:/Users/222/Desktop/MyProject/korabl-s-podnyatymi-parusami-v-more.png").getImage();//рисуем картинку с компа

            g2.drawImage(image, 220, 20, null);

            g2.drawImage(image2, 220, 20, null);

        }

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame();

        jFrame.setVisible(true);

        jFrame.setTitle("this my add");

        jFrame.setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);

        Toolkit toolkit = Toolkit.*getDefaultToolkit*();

        Dimension dimension = toolkit.getScreenSize();

        jFrame.setBounds(dimension.width / 2 - 250, dimension.height / 2 - 150, 500, 300);

        return jFrame;

    }

    void PrintEnvironment()/\*метод выведет все доступные шрифты в консоль\*/ {

        String[] fonts = GraphicsEnvironment.*getLocalGraphicsEnvironment*().getAvailableFontFamilyNames();

        for (String s : fonts) {

            System.*out*.println(s);

        }

    }

}

### #98.Action. Listener

Добавляем кнопочку и учим ее чтото делать

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

/\*\*

 \* Created by husiv on 9/13/16.

 \*/

public class ActionListernerLesson {

static JFrame jFrame = getFrame();

public static void main(String[] args) {

 JPanel jPanel = new JPanel() {

 };

 jFrame.add(jPanel);

 JButton button = new JButton("submit");

 jPanel.add(button);

 button.addActionListener(new ActionListener() {

 @Override

 public void actionPerformed(ActionEvent e) {

 jPanel.setBackground(Color.blue);

 }

 });

 }

static JFrame getFrame() {

 JFrame jFrame = new JFrame() {};

 jFrame.setVisible(true);

 jFrame.setBounds(750, 250, 500 ,500);

 jFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

 return jFrame;

 }

}

### #99.Swing.EventHandler

//эта штука сокращает длину кода информации более подробной мало на русском

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.beans.EventHandler;

*/\*\**

*\* Created by husiv on 9/13/16.*

*\*/*

public class ActionListernerLesson {

    public static void main(String[] args) {

        JFrame jFrame = *getFrame*();

        JPanel jPanel=new JPanel();

        jFrame.add(jPanel);

        JButton jButton=new JButton("submit");

        jPanel.add(jButton);

//        jButton.addActionListener(new ActionListener() {

//            @Override

//            public void actionPerformed(ActionEvent e) {

//                //переведет текст с кнопки на title

//                jFrame.setTitle(((JButton/\*приведение параметра\*/)e.getSource()).getText());

//            }

//        });

        //заменяет предыдущие строчки кода

        jButton.addActionListener(EventHandler.*create*(ActionListener.class,jFrame,"title","source.text"));

        //EventHandler конечно топчик но не понятно откуда он команды взял

        //короче он опирается на прокси класс

        // а текст в скобочках это реальные команды какогото класса которые раньше исползовались как функции

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #100.LookandFeel

//изменение стилей

//не факт что поменяет

//они как то все похожи там один только отличается

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LookAndFeel {

    public static void main(String[] args){

        JFrame jFrame = *getFrame*();

        JPanel jPanel=new JPanel();

        jFrame.add(jPanel);

        //как посмотреть доступные стили

//        UIManager.LookAndFeelInfo[] lookAndFeelInfos=UIManager.getInstalledLookAndFeels();

//        for (UIManager.LookAndFeelInfo l:lookAndFeelInfos) {

//            System.out.println(l.getName());

//            System.out.println(l.getClassName());

//        }

        JButton jButton=new JButton("Metal");

        JButton jButton2=new JButton("Nimbus");

        JButton jButton3=new JButton("CDE/Motif");

        JButton jButton4=new JButton("Windows");

        JButton jButton5=new JButton("Windows Classic");

        {

            jPanel.add(jButton);

            jPanel.add(jButton2);

            jPanel.add(jButton3);

            jPanel.add(jButton4);

            jPanel.add(jButton5);

        }//добавление в панель

        jButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    UIManager.*setLookAndFeel*("javax.swing.plaf.metal.MetalLookAndFeel");

                }catch (Exception ignore){}

            }

        });

        jButton2.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    UIManager.*setLookAndFeel*("javax.swing.plaf.nimbus.NimbusLookAndFeel");

                }catch (Exception ignore){}

            }

        });

        jButton3.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

                }catch (Exception ignore){}

            }

        });

        jButton4.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

                }catch (Exception ignore){}

            }

        });

        jButton5.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel");

                }catch (Exception ignore){}

            }

        });

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #101.WindowListener

//позволяет переопределить некоторые методы Window при работе с окном

//всего 7 методов

//например добавить диал.окно с предупреждением о сохранении данных при нажатии кнопки закрыть рабочее окно

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

*/\*\**

*\* Created by husiv on 9/13/16.*

*\*/*

public class WindowListenerLesson {

    static JFrame *jFrame* = *getFrame*();

    public static void main(String[] args) {

        JPanel jPanel = new JPanel() {};

*jFrame*.add(jPanel);

*jFrame*.addWindowListener(new WindowAdapter() {

            /\*там 7 методов чтобы не переопределять их все вызовем WindowAdapter\*/

            @Override

            public void windowOpened(WindowEvent e) {

                super.windowOpened(e);

                jPanel.setBackground(Color.*blue*);

            }

            @Override

            public void windowClosing(WindowEvent e) {

                super.windowClosing(e);

                jPanel.setBackground(Color.*orange*);//не сработало

                try {Thread.*sleep*(20000); }catch (Exception ignore){}//а пауза сработала

                System.*exit*(0);

            }

        });

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #102.Actions

//небольшие подсказки при наведении мышью

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

*/\*\**

*\* Created by husiv on 9/13/16.*

*\*/*

public class ActionsLesson {

    static JFrame *jFrame* = *getFrame*();

    static JPanel *jPanel* = new JPanel() {};

    public static void main(String[] args) {

*jFrame*.add(*jPanel*);

        JButton button = new JButton(new ColorAction());

        button.setText("submit");

*jPanel*.add(button);

    }

    static class ColorAction extends AbstractAction {

        public ColorAction() {

            //при наведении мышью на кнопку будет выводиться небольшая подсказка "change color"

            putValue(Action.*SHORT\_DESCRIPTION*, "change color");}

/\*можно нагуглить некоторые свойства абстракт акшиона

<https://spec-zone.ru/RU/Java/Docs/8/api/javax/swing/Action.html>

поискал в оракле\*/

        @Override

        public void actionPerformed(ActionEvent e) {

*jPanel*.setBackground(Color.*blue*);

        }

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #103.KeyStroke короткое сочетание клавиш

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

*/\*\**

*\* Created by husiv on 9/13/16.*

*\*/*

public class KeyStrokeLesson {

    static JFrame *jFrame* = *getFrame*();

    static JPanel *jPanel* = new JPanel() {};

    public static void main(String[] args) {

*jFrame*.add(*jPanel*);

        Action action = new ColorAction();

        JButton button = new JButton(action);

        button.setText("submit");

*jPanel*.add(button);

        //делаем кейстрок

        KeyStroke keyStroke = KeyStroke.*getKeyStroke*("ctrl B");

        //мапа сочетание ключика и значения

        InputMap inputMap = *jPanel*.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*);//для виндовса другая команда,определяем при каком фокусе,окошке и положении мыши быдет срабатывать мап

        inputMap.put(keyStroke, "changeColor");

        ActionMap actionMap = *jPanel*.getActionMap();

        actionMap.put("changeColor", action);

    }

    static class ColorAction extends AbstractAction {

        public ColorAction() {

            putValue(Action.*SHORT\_DESCRIPTION*, "change color");

        }

        @Override

        public void actionPerformed(ActionEvent e) {

*jPanel*.setBackground(Color.*blue*);

        }

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #104.MouseListener.

//прослушивание действий мыши

//есть проблемы с одновременным выведением нескольких компонентов

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

*/\*\**

*\* Created by husiv on 9/13/16.*

*\*/*

public class MouseLesson {

    static JFrame *jFrame* = *getFrame*();

    public static void main(String[] args) {

        JPanel jPanel = new JPanel(){};

*jFrame*.add(jPanel);

        JComponent jComponent = new MyPanel();

*jFrame*.add(jComponent);

        //интерфейс обработки событий мыши

        jPanel.addMouseListener(new MouseAdapter() {

            //если адаптер не добавлять там будет 5 методов по нажатию отпусканию входу выходу мыши

            @Override

            public void mouseClicked(MouseEvent e) {

                super.mouseClicked(e);

                jPanel.setBackground(Color.*blue*);

            }

        });

*jFrame*.addMouseMotionListener(new MouseAdapter() {

            @Override

            public void mouseMoved(MouseEvent e) {

                super.mouseMoved(e);

                MyPanel.*xCoord* = e.getX();

                MyPanel.*yCoord* = e.getY();

                try {Thread.*sleep*(1000);}catch (Exception ignore){}//делает задержку в цикле

                jComponent.repaint();//обьект перекрашивается,это цикл

            }

        });

    }

    static class MyPanel extends JComponent {

        public static int *xCoord*;

        public static int *yCoord*;

        @Override

        protected void paintComponent(Graphics g) {

            super.paintComponent(g);

            ((Graphics2D)g).drawString("Coordinates x: " + *xCoord* + " Y: " + *yCoord*, 50, 50);

        }

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #105.Текстовые поля JtextFild,JPasswordFild,JTextArea,ScrollPane

import javax.swing.\*;

*/\*\**

*\* Created by husiv on 9/15/16.*

*\*/*

public class TextInputLesson {

    public static void main(String[] args) throws Exception {

        JFrame jFrame = *getFrame*();

        JPanel jPanel = new JPanel();

        jFrame.add(jPanel);

        //это надпись для какого нибудь поля

        jPanel.add(new JLabel("<html><b>this is label</b></html>"));

        //самое простое добавляем текст

        jPanel.add(new JTextField("text", 10));

        JTextField jTextField=new JTextField();

        jTextField.getText();//можно получить текст там дофига геттеров и сеттеров

        //будет прятаться за звездочки

        jPanel.add(new JPasswordField(10));

        //большое поле в несколько строк саморасширяющееся

        JTextArea jTextArea = new JTextArea(8, 20);

        jTextArea.setLineWrap(true);//не дает расширятся строке переводит на новую строчку расширяется вниз

//        jPanel.add(jTextArea);

        //добавит ползунок если TextArea переполнится ScrollBar добавит сразу не дожидаясь переполнения

        JScrollPane jScrollPane = new JScrollPane(jTextArea);

        jPanel.add(jScrollPane);

        jPanel.revalidate();//походу цикл как и repaint

    }

    static JFrame getFrame() {

        JFrame jFrame = new JFrame() {};

        jFrame.setVisible(true);

        jFrame.setBounds(750, 250, 500 ,500);

        jFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        return jFrame;

    }

}

### #106.swing. галочка выбора. Radio button,RadioGroup,ComponentBox,Border

jPanel.add(new JCheckBox("check box label"));//добавит квадратик для галочки с тестом информации

//Кружочки обязательного выбора какого-то одного варианта

JRadioButton jrb=new JRadioButton("one");

JRadioButton jrb2=new JRadioButton("one");

//Выставим выбор по умолчанию

jrb2.setSelected(true);//обратно получаем геттерами

//Сгруппируем 2 радиобаттона

ButtonGroup bg=new ButtonGroup();

bg.add(jrb);

bg.add(jrb2);

jPanel.add(jrb);

jPanel.add(jrb2);//появятся 2 кружочка с предв выбором

//Это такое поле может с рамочкой в которое можно запихнуть различные кнопки

Border border=new BorderFactory.createEtchedBorder();

Border border=new BorderFactory.createTitleBorder(border,"title");//чтобы было видно

//jPanel.setBorder(border2);//передвину вниз а то комбо Бордер не видно будет

JComboBox<String>  jComboBox=new JComboBox();//небольшой открывающийся список для настройки

jComboBox.addItem("one");

jComboBox.addItem("two");

jComboBox.addItem("three");

jPanel.add(jComboBox);

jPanel.setBorder(border2);

### #107.swing.Sliders

//Типа можно регулировать громкость допустим

JSlider jSlider=new JSlider(JSlider.HORIZONTAL,0,100,20);//горизонтальный,мин,мах,и по умолчанию

jPanel.add(jSlider);

jSlider.setMinorTickSpaicing(10);//разметка маленькие чёрточки каждые 10

jSlider.setMajorTickSpaicing(20);//больш.деления

jSlider.setPainTicks(true);

jSlider.setSnapToTicks(true);//ползунок магнитится к делениям

jSlider.setPainLebels(true);//добавит циферки к делениям

### #108.swing.Menu

JFrame jFrame = getFrame();//не в панели

        JMenuBar jMenuBar = new JMenuBar();

        JMenu fileMenu = new JMenu("File");

        JMenu editMenu = new JMenu("Edit");

        jMenuBar.add(fileMenu);

        jMenuBar.add(editMenu);

//Добавляем след уровень

        fileMenu.add(new JMenuItem("Open",new ImageIcon(""), 'O'));//можно добавить иконку,последнее это мнемоник для нажатия ctrl+alt+O

        fileMenu.add(new JMenuItem("Open As")).setEnabled(false);//делает поля недоступными, серыми

        fileMenu.add(new JPopupMenu.Separator());//разделительная полосочка

        fileMenu.add(new JMenuItem("Close"));

//Добавляем след уровень

        editMenu.add(new JMenuItem("Cut"));

        editMenu.add("copy");

//Создаём 3й уровень с переходом по стрелочке

        JMenu options = new JMenu("Options");

        options.add(new JMenuItem("one"));

        options.add(new JMenuItem("two"));

        editMenu.add(options);

//Добавляем во 2й уровень поле для галочки

        editMenu.add(new JCheckBoxMenuItem("check box"));

//Добавляем с галочками кружочками

        ButtonGroup buttonGroup = new ButtonGroup();

        JRadioButtonMenuItem jRadioButtonMenuItem1 = new JRadioButtonMenuItem("one");

        JRadioButtonMenuItem jRadioButtonMenuItem2 = new JRadioButtonMenuItem("two");

        buttonGroup.add(jRadioButtonMenuItem1);

        buttonGroup.add(jRadioButtonMenuItem2);

        editMenu.add(jRadioButtonMenuItem1);

        editMenu.add(jRadioButtonMenuItem2);

        JMenuItem exit = fileMenu.add(new JMenuItem("exit"));

        exit.setAccelerator(KeyStroke.getKeyStroke("ctrl Q"));//добавление Кей команды для exit

//Пишем код для закрытия окна при нажатии на exit

        exit.addActionListener(new AbstractAction() {

            @Override

            public void actionPerformed(ActionEvent e) {

                System.exit(0);

            }

        });

//Добавляем все во фрейм

        jFrame.setJMenuBar(jMenuBar);

        jFrame.revalidate();//какой-то цикл

### #109.swing.PopupMenu

//Всплывает по клику прав.клавишей мыши

jFrame.add(jPanel);

        JPopupMenu jPopupMenu = new JPopupMenu();

        jPopupMenu.add(new JMenuItem("one"));

        jPopupMenu.add(new JMenuItem("two"));

JMenuItem("one").addActionLisenner(new ActionLissener(){

@Override

public void actionPerformed(new ActionEven e){

}

});

}

        jPanel.setComponentPopupMenu(jPopupMenu);

### #110.ToolBar

//Инструментальное поле у края окна

JToolBar jToolBar = new JToolBar("title");//текст title появиться при перемещении на новое место

        JButton one = new JButton("one");

        one.setToolTipText("first");//добавим подсказочный всплывающий текст

        jToolBar.add(one);

        jToolBar.add(new Button("two"));

        jToolBar.addSeparator();//небольшой разграничитель

        jToolBar.add(new Button("three"));

        jPanel.add(jToolBar);

### #111.Layout

//Инструмен дизайна круче чем в с++

//Поля сами настраиваются по размеру в зависимости от .....

Crs->swing->пр.кл.мыши->new->GUI Form->//задать имя->выбрать один из 6 layout->//накидываем элементов

Докачка ресурсов

File->setting->поиск: plugins->второй поиск: swing->UI Designer:нужно установить

Проверяем//по сбор.полю прав кл.мыши->previev//все соберётся и покажет некое собранное окошко

//Чтобы настроить все надо пересмотреть

Там что-то докачивать надо

### #112.Swing.Border and Gride Layout

//BorderLayout: размещение каких либо полей по частям света северюг западвосток центр

        jFrame.add(new Button("one"), BorderLayout.NORTH);

        jFrame.add(new Button("two"), BorderLayout.WEST);

        jFrame.add(new Button("three"), BorderLayout.CENTER);

        jFrame.add(new Button("four"), BorderLayout.EAST);

        jFrame.add(new Button("five"), BorderLayout.SOUTH);

        JPanel jPanel = new JPanel();

        jFrame.add(jPanel);

        //GridLayout: размещение лейаутов в таблице по колонкам и строкам

//        jPanel.setLayout(new GridLayout(2,2));

//        jPanel.add(new Button("one"));

//        jPanel.add(new Button("two"));

//        jPanel.add(new Button("three"));

//        jPanel.add(new Button("four"));

### #113.GridBagLayout

jFrame.add(jPanel);

//распределяет элементы по клеточкам как в екселе

GridBagLayout gridBagLayout = new GridBagLayout();

jPanel.setLayout(gridBagLayout);

GridBagConstraints constraints1 = new GridBagConstraints();//будет определять количество занимаемых клеток

constraints1.weightx = 0;//указываем нули чтобы он сам растягивался

constraints1.weighty = 0;

//указываем отправные клеточки в данном случае занимаем от верхней левой

constraints1.gridx = 0;

constraints1.gridy = 0;

//указываем крайнюю правую и нижнюю

constraints1.gridheight = 2;

constraints1.gridwidth = 2;

//добавляем туда текстовое поле

jPanel.add(new JTextArea(4, 20), constraints1);

GridBagConstraints constraints2 = new GridBagConstraints();

constraints2.weightx = 0;

constraints2.weighty = 0;

constraints2.gridx = 2;//на самом деле 3я потомучто отсчет с 0

constraints2.gridy = 0;

constraints2.gridheight = 1;

constraints2.gridwidth = 2;

jPanel.add(new JButton("submit2submit2asdf"), constraints2);

GridBagConstraints constraints3 = new GridBagConstraints();

constraints3.weightx = 0;

constraints3.weighty = 0;

constraints3.gridx = 2;

constraints3.gridy = 1;

constraints3.gridheight = 1;

constraints3.gridwidth = 1;

jPanel.add(new JButton("submit3"), constraints3);

GridBagConstraints constraints4 = new GridBagConstraints();

constraints4.weightx = 0;

constraints4.weighty = 0;

constraints4.gridx = 3;

constraints4.gridy = 1;

constraints4.gridheight = 1;

constraints4.gridwidth = 1;

jPanel.add(new JButton("submit2"), constraints4);

jPanel.revalidate();

### #114.swing. No Layout

//отсутствие лэйоута не лучшее решение для дизайна

jFrame.add(jPanel);

jPanel.setLayout(null);

JButton jButton = new JButton("bubmit");

jPanel.add(jButton);

jButton.setBounds(50, 50, 150, 30);

### #115.swing. Custom layout

//Попытка автоматизировать процесс распределения кнопок на панели

jFrame.add(jPanel);

        jPanel.setLayout(new MyLayoutMangager());

        jPanel.add(new JButton("submit"));

        jPanel.add(new JButton("submit"));

        jPanel.add(new JButton("submit"));

    }

//Создаём свой пользовательский лэйаут

    static class MyLayoutMangager implements LayoutManager {

        @Override

        public void addLayoutComponent(String name, Component comp) {}

        @Override

        public void removeLayoutComponent(Component comp) {}

        @Override

        public Dimension preferredLayoutSize(Container parent) {

            return null;

        }

        @Override

        public Dimension minimumLayoutSize(Container parent) {

            return null;

        }

        @Override

        public void layoutContainer(Container parent) {

            int n = parent.getComponentCount();

            for (int i = 0; i < n; i++) {

                Component component = parent.getComponent(i);

                component.setBounds(i \* 100, i \* 40, 100, 40);//распределяем компоненты по диагонали

            }

        }

    }

### #116.swing. Focus

jFrame.add(jPanel);

        jPanel.add(new JButton("submit"));

        jPanel.add(new JButton("submit2")).setFocusable(false);//при переключении фокуса клавишами он не будет подать на эту кнопку

        jPanel.add(new JButton("submit3")).requestFocusInWindow();//ставит фокус по умолчанию сюда

### #117.swing.dialogs

//Диалоговое окно при появлении блокирует основной экран

jFrame.add(jPanel);

        JButton jButton = new JButton("show dialog");

        jPanel.add(jButton);

//При нажатии кнопки вызовется диалоговое окно

//Печатаем для кнопки акшион лисенер

        jButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

//Просто выведет сообщение JOptionPane.showMessageDialog(jPanel, "this is message", "title", JOptionPane.ERROR\_MESSAGE);//задаем текст сообщения, титл окошка, иконку перед сообщением

//Там есть заготовки окон

//Может возвращать значение инт нажатой кнопки по ее порядку

                int confirmValue = JOptionPane.showConfirmDialog(jPanel, new JButton("bla"));//там есть больше параметров

                System.out.println(confirmValue);

int optionValue = JOptionPane.showOptionDialog(jPanel, "this is option dialog", "title3", JOptionPane.DEFAULT\_OPTION, JOptionPane.ERROR\_MESSAGE, null, new String[]{"one", "two"}, "two");//передаем туда массив кнопок и по умолчанию two, и ещё сообщение иконку и ещё что-то

                System.out.println(optionValue);

//Вводим текст в окно

String inputvalue = JOptionPane.showInputDialog(jPanel, "this is input dialog");

                System.out.println(inputvalue);//вернёт введеный текст

            }

        });

### 118.swing. Custom dialogs

Самостоятельно настроим диал.окно

jFrame.add(jPanel);

        JButton jButton = new JButton("show dialog");//кнопка для вызова окна

        jPanel.add(jButton);

        jButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                MyDialog myDialog = new MyDialog(jFrame);

                myDialog.setVisible(true);//вызываем

            }

        });

    }

//Реализуем окно

    static class MyDialog extends JDialog {

        MyDialog(JFrame owner) {

            super(owner, "title", true);//вместо первого параметра он выставил jFrame//2й титл//3й модальность, включ блокировку основного окна

            add(new JLabel("<html><h1><i>hello</i></h1></html>"), BorderLayout.NORTH);//это просто текст лэбл по размещению север

            add(new JButton("test"),BorderLayout.SOUTH);//а это просто кнопка

            setBounds(50, 50, 250, 150);задаем координаты окна и размер

        }

    }

### #119.File Dilog

//Диалог показывает файлы и дериктории с некоторыми примитивными кнопочками типа сохранить и открыть

//Инфы много надо видео пересматривать

jFrame.add(jPanel);

//Кодируем кнопку

        JButton jButton = new JButton("submit");

        jPanel.add(jButton);

        jButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

//Тот самый обьек

                JFileChooser jFileChooser = new JFileChooser();

                jFileChooser.setCurrentDirectory(new File("."));//точка это текущая директория. вместо точки можно c:/win...

                jFileChooser.setSelectedFile(new File("."));//заранее выберет конкретные файлы

                jFileChooser.setMultiSelectionEnabled(true);//для выбора/выделения сразу нескольких файлов при помощи ctrl и др.

                jFileChooser.setFileSelectionMode(JFileChooser.FILES\_AND\_DIRECTORIES);//выберет только директории

                //jFileChooser.showOpenDialog(jFrame);//два метода различаются

                //jFileChooser.showSaveDialog(jFrame);//одной кнопочкой открыть или сохранить

//.showDialog(jFrame,"save as naprimer")пользовательская настройка

//Настраиваем фильтр по разрешению файла

                FileFilter fileFilter = new FileFilter() {

                    @Override

                    public boolean accept(File f) {

                        if(f.getName().endsWith("gif"))

                            return true;//отобразит файл если его разрешение gif

                        else

                            return false;

                    }

                    @Override

                    public String getDescription() {

                        return "Only gif";//текст на выдвиг.кнопочке меню которая запускает фильтр

                    }

                };

                jFileChooser.setFileFilter(fileFilter);

                jFileChooser.setFileView(new FileView() {});

//Просто воткнули текст.поле в диалог

                JTextField jTextField = new JTextField("bla", 50);

                jFileChooser.add(jTextField, BorderLayout.EAST);

                jFileChooser.addPropertyChangeListener(new PropertyChangeListener() {

                    @Override

                    public void propertyChange(PropertyChangeEvent evt) {

                        jTextField.setText(evt.getNewValue().toString());

                        //JOptionPane.showMessageDialog(jPanel, "this is message", "title", JOptionPane.ERROR\_MESSAGE);

                    }

                });

                int i = jFileChooser.showDialog(jFrame, "Select");

                System.out.println(i);

                File file = jFileChooser.getSelectedFile();

                System.out.println(file.getName());

            }

        });

### #120.swing. ColorDialog

Кнопка батон вызывает диалоговое окно с палитрой

//Остальное смотреть в видеоуроке

### 121.awt. shapes

//Учимся рисовать фигуры в awt

### #122.awt.Areas

//Класс позволяет работать с несколькими фигурами

Складывать их в одну

Вычитать одну из другой и другое

### #123.awt.Stroke

//Настраивали линию, строку разными режимами

### 124.awt. Paint

//Изменение цвета

//Есть интересные вещи например перетекание цвета из одного в другой

### #125.awt. Transformation

//Манипуляции с фигурами

Поворот перемещение сдвиги всякие

### #126.awt.Clip

//Обрезание фигур

### 127.awt.Transparency

//Регулировка прозрачности

//Способы наложения полупрозрачных фигур друг на друга

### #128.awt.Скорость и качество отрисовки фигур

### #129.awt.чтение и запись изображений

//Перезапись изображения со сменой формата

//Получение циклом списка доступных форматов

//При загрузки картинки могут возникнуть проблемы с форматом, поэтому..... Там куча кода

//Сабнейлс это маленькая картинка

### #130.awt.работа с изображениями

//Разбираем картинку попиксельно

//Немного о стенографии

### #131.awt. image filter

//Несколько классов фильтров

//Позволяют поворачивать, уменьшать и др.

### #132.awt. печать изображений.

Работа с принтером

### #133.awt. Печать документов

Работа с принтером

### #134.awt. Print PostServise

//Специальный формат для переноса для печати с одного компа на другой

PostServise.odd

### #135.awt. Работа с буфером обмена, основы, копирование строк

//Оч интересно

//Использует класс Clipboard

### #136.awt. буфер обмена. Копирование картинок

//Помещаем в Clipboard image

//Присутствует проверка flavor на поддерживаемость формата буфером

### #137.awt. DragandDrop

//Перенос копирование перетаскивание мышью в том числе из одного окна в другое

//Все просто

### #138.awt. Splashscreen. Заставка

//Заставка перед запуском приложения пока данные подгружаются

//Progress bar полоса загрузки

### #139.awt. Запуск других приложений,файлов и браузера

//Написанное приложение может при помощи класса Desktop запускать другие программы с ПК открывать файлы и открывать браузер с переходом по конкретной ссылке

### #140.awt. System tray панель задач

//Добавление иконки приложения в панель задач

//Можно добавить некую функциональность типа менюшки

### #141.установка жду на мак ос

//......

### #142.установка эклипса на мак ос

//......

### #143. Установка айдие на мак ос

//.....

### #144. Создание самозапускающегося jar

//Не рассказал про bat

### #145. Аплеты

//Устаревшая функция добавления jar файла в http в тело <body>

### #146. Java WedStart

//Как бы с интернета загружаем и запускаем файл свинговое окошечко

//У него будет закрыт доступ к файловой системе компьютера

### #147.Properties.

//Класс наподобие мапы

Хранит некие объекты по ключикам

//Может создавать файл my.properties

Используемый для хранения паролей Логинов других соединений

Чтобы можно было перезаливать приложение без потери данных

### #148.Preferences.

Создаёт файл в одной из корневых папок операционной системы с настройками программы

Программа выгружает в него данные при закрытии и загружает при запуске

### #149.логирование

Существуют если программа начинает валиться

Данные логов могут отправляться на сервер или храниться на компе

Сложная тема

### #150. Debagging

Очень широкая тема по нахождению багов с помощью логов, отладчика, сторонних программ отслеживания состояния процессора

### #151. Debagging swing

Короткое видео по нахождению ошибок в окошке swing если вдруг что-то не так прорисовывается

### #152. Robot

Сам клиент по кнопочкам переводит мыш на конкретные координаты

Фотографирует

Актуально при создании вирусов

### #153. Как ему предложили работу в США

//Короче его используют как дешёвую раб.силу но ему по кайфу

### #154. JavaDoc

//Подготовка тех.документации на свою прогу

//Комментарии /\*\*  \*/ и много чего ещё

### #155.Overlioading

Некоторые тонкости из переопределения

### #156.переопределение equals

 //Принципы сравнения обьектов

Рефлексивность

Симметрия

Транзитивность

Консистентност

Нот нулл

### #157. instanseof null

Маленькая фишка из equals типа проверки на нулл и не требуется потому что проверка instanseof null все равно вернёт false

### #158.почему нет множественного наследования в Java

// Можно заимплементить несколько интерфейсоф но нельзя заэкстэндить несколько классов потому что при вызове метода родительского класса джава не будет знать в из какого именно парента вызывать метод, а в интерфейсе все равно реализации методов никакой нет

### #159. Статическая инициализация

//Подробнее о статиках

### #160. final

//Бывают поля final

//При применении final к переменной мы ей оставляем возможность инициализировать мы только один раз!

//Методы final не наследуются

### #161. Enums

//Спал.. когда смотрел

Лень пересматривать

### #162. Static inner classes

//Имеет доступ только к статические переменным парент класса

Могут создаваться экземпляры

Может вызываться в других классах но с особенностями синтаксиса

Иногда нарушает инкапсуляцию

### #163.inner class

Не может быть создан без экземпляра внешнего класса

Имеет полный доступ к полям внешнего класса

Не может создавать своих статиков в любом виде кроме final static и констант

### #164. Локальный класс

//Создаётся внутри метода

На него распространяются все ограничения внутреннего класса

Живёт только внутри метода

### #165. Интерфейсы

Могут содержать в себе переменные классы и другие интерфейсы

### #166. Try с ресурсами и AutoClosable

При открывании ресурсов каких либо требуется try catch потом нужно закрыть ресурсы для этого тоже нужен трай кэтч

Чтобы код не громоздить придумали AutoClosable который унаследован от closable

Должно получиться 2 строчки

Подробности синтаксиса в видео

### #167.отношения между классами is a и has a

Типа классы не могут наследоваться от чего попало даже если это компилируется

Особенности архитектуры

Типа машина и радио не имеют ничего общего чем его наследовать лучше внутри машины создать экземпляр радио

Ну и вообще выражение is a не позволяет наследовать разные объекты даже если они очень похожи

### #168.Cohesion and Coupling

Cohesion - сплочённость.. философия о том что класс должен быть сосредоточен на какой-то одной цели и не брать на себя много обязанностей

Coupling - связанность

Типа отсутствие геттеров и сеттеров это плохая связанность

### #169.работа с файлами и стримы

//Для работы с определенными расширениями файлов вроде только с текстом

Reader

Writer

//Для работы с байтами и потоками (там картинки и другое)

InputStream

OutputStream

//В видео много информации по работе с этими классами

### #170. path and Files

//Классы с более расширенным набором методов по работе с файлами и файловым деревом

//Оч.интересно

//Надо пересматривать

### #171. Scanner, PrintWriter

Более удобные классы по записи и чтению данных

Про сканер не сказал что он может из консоли читать

### #172.xml. DOM подход

Работа с файлом XML при помощи класса dom

Много кода

### #173.xml.SAX подход

Вывод элементов XML при помощи класса SAX

### #174.создание XML при помощи dom

.....

//Надо пересмотреть.много кода

### #175.создание XML при помощи SAX

...

SAX более удобен

### #176.XPath и работа с XML

Работает тоже с xml

Даёт доступ к конкретному полю XML

Короче хороший класс

### #177.основы сетей

Про айпи про уровни которые проходит запрос и др.

### #178.Сокеты клиентские

Отправляли запрос в Перу узнать время

... Там есть код

### #179.про серверсокет

Достаточно инфы

### #180.http и как получить html

Дело в том что обычным сокетом html не получить так как он зашифрован

Есть специальный класс URL для работы с html

Подробнее в видео

### #181.отправка емайл

Проблемы будут в том что Гугл не даст отправить емайл без ввода пароля и другого связано это со спамом

### #182.установка Интел айдия ультимате

......... Ultimate окончательный

## #183.JDBS

Интерфейс для работы с различными базами данных

Основные

Mssql,mysql,sqllite

## #184.установка mysql на windows 10

Короче он ее загрузил и работает с ней через консоль

Показал не ного первых запросов

### #185.установка mysql на Линукс

......

## #186. Подключение к mysql через айдию

Нужна платная версия айдии

## #187.подключение к базе данных на примере mysql

Надо драйвер скачивать

И подключаться

Надо будет пересмотреть

## #188.gdbc SQL запросы

Показал пару примеров запросов как положить данные создать таблицу и вытащить данные

## #189.sql injection and preparementStatement

Не понял ничего

Инъекции и класс который защищает от неправильных инъекций что-то с защитой связано

Короче инъекциями только хакеры пользуются

А PreparamentStatement это защита от них ну и заодно им легче писать

## #190.BLOB Binary Large Objects

Для записи больших файлов например картинок

Короче блоб тоже в отдельную переменную записывается

## #191.sql даты

Просто отправляем и записываем даты в миллисекундах от 1970го

## #192.JDBS хранимые процедуры

Очень интересно

Вызываем заранее подготовленные запросы

## #193.JDBS множественные результаты

Можно запрашивать сразу несколько запросов из таблиц при помощи хранимых процедур

## #194.JDBS Scrollable RowSet

Делаем запросы бегая по строкам вверх вниз а также перемещаясь влево вправо по колонкам

## #195.JDBS Update ResultSet

Добавляем удаляем выводим

## #196.JDBS CachedRowSet кеширование запросов

Чтобы постоянно не поддерживать соединение с базой и давать другим в нее залезть мы кешируем ее проводим операции и изменения загружаем обратно в базу

## #197.JDBS Metadata

Получаем метаданные таблицы

Имена таблиц количество колонок строк их имена

## #198.JDBS transaction транзакции

Целостные операции

Если один запрос не выполнился то связанные с ним запросы отменятся

## #199.JDBS.как накатить SQL группой SQL Batch

Объединяет несколько запросов в один

Продолжение транзакций

Если объединить несколько десятков запросов то он съэкономит время

## #200.JDBS.уровни изоляции транзакций

Советуют не парится по этому поводу но на собеседованиях спрашивают

Там 4 уровня изоляции

Если в базе роются одновременно несколько потоков то могут возникать ошибки

Естественно по умолчанию во многих функциях стоят локи

## #201.JMX Java menegment extension

Посылать запросы на приложение работающее в бэкграунде и узнавать его состояние .. памяти как-то влиять на него управлять

Quartz job sheduler позволяет запускать код по расписанию например каждый час запускать программу

Создаём обьек Bean это простое приложение .java и вешаем его на сервере

И при помощи клиентского jmx подключаемая к нему и управляем его методами вызываем его методы

Следим по графику за его активностью

## #202. JMX. Notification.

Нотификация внедряется в после основного кода

И когда мы удаленно запускаем методы нотификация возвращает тип время какие-то ещё переменные типа что все прошло хорошо

## #203.JMX.на стороне клиента

На нашем компьютере мы создаём прокси который шлёт запросы на сервер на объект который там находится и получает от него ответы о ходе его работы

### #204.Сравнение строк

Сравниваем при помощи класса Collator

В том числе есть возможности сравнивать с символами из других языков

## #205.Форматирование строк

Оч интересно

Программирование внутри строки

В зависимости от ситуации она будет менять свое содержание

## #206.Locale-локаль

Для интернационализации приложения

Если используются разные языки в приложении

## #207.локализация и интернационализация

Продолжаем работать с файлами и проперти файлами и другими ресурсами все интересно

### #208.Бины

Бины это простые классы частопереиспользуемые с приватными полями и ТД.

Чаще употребляются в GUI формах это кнопки баттоны различные элементы

Есть возможность создания своих бинов

### #209.загрузчик классов class loader

это чтото на оч сложном низкоуровневом про байты про с++

есть готовые библиотеки зачемто ему понадобилось создавать свой класслоадер

### #210.Byte code verifier

это связано с предыдущей темой

типа после упаковки jarника перед запуском верифайер еще раз проверяет на наличие ошибок в коде особенно на переполнение буфера

хакеры уже готовый джарник берут и на уровне 16ричного кода подменивают данные так что переполняется буфер и через открытые порты туда можно проникнуть и что нибудь изменить

в видео немножко показана работа с 16ричным кодом

### #211.Security manager

запрещает проверяет использование в коде различных запросов типа system exit 0 или os.name или к другим пропертям каковых большое количество

в видео показана возможность включения выключения в секьюрити этих запретов а также создание своих собственных

### #212.Security manager аутентификация пользователей

делаем команды которые запрещены кроме конкретного пользователя

## @#213.работа с шифрованием md5 и sha-1

используется для хранения паролей в базе данных

впринципе хешкод расшифровать нереально если не использовать для пароля примитивные слова и комбинации

## @#214.certification

ну типа jarник подписывается определенным ключем ключ переводится sha-1 и с подписью отправляется другому человеку а он при получении проверяет подпись через другой распространяемый ключ

это все сложно

формируется два ключа публичный и приватный зашифровать бубличным ключем сможет кто угодно а расшифровать можно только математически связанным приватным ключем

про формулу эйлера и другое здесь:

https://habr.com/ru/post/534014/

ссылка на плейлист по криптографии:

https://youtube.com/playlist?list=PLxGo9dxQkqWAOEUOMOo9Da0l-8qKndNTP

## @#215.шифрование AES

шифрует типа одним ключем

простое шифрование для дешифровки нужен тотже ключ что и при шифровании

## @#216.шифрование RSA

тотже класс что и при шифровании аес но с получением приватного и бубличного ключа

## @#217.запуск javascript на jave

ну ты понял

есть небольшие заморочки с совмещением windowsa и кода скрипта так как он на браузер ориентирован

# @#218.компилируем в jar в рантайме

прям без всяких консолей и других танцев прямо классом компилируем класс

## @#219. Аннотации

типа @Override но только создаем свои

применяем их к классам методам полям переменным ограничиваем в применении к определенным полям передаем в аннотации параметры для переменных и другие разные штуки

## @#220.Пример применения аннотаций

привел небольшой пример типа аннотацией пометили одну из переменных в классе и циклом вывел только те которые были помечены аннотацией

интересно и сложно связано с рефлексией

### #221.RMI

еще один способ запускать методы обьекта сервера удаленно

подходит например если оба приложения написаны на одном языке

//надо пересмотреть будет

//там присутсвует небольшой код который узнает какие приложения слушают порт

### #222.запуск "native" кода

Native это классы написанные на другом языке на с++ наверное только

В видео показан пример самостоятельного создания такого класса

### #223.JNDI

Java Naming and Directory Interface

Служба имён и каталогов

Связь имён и объектов

Например доменное имя и IP

В основном применяется в базах данных но так ещё есть другие возможности например для файлов

Короче служба которая связывает привычное нам c:\window\user\text.txt с тем что возможно для понимания машины типа 233.566.00.1234

Аналог этой системы DNS связывает домены и IP

### @#224.JAXB

типа переводим класс с переменными в xml файл а потом достаем обратно из хмл в клас джавы

в работе используются аннотации

### #225.java FX

в отличии от свинга может использовать элементы написанные на html javascript и др. подходит для мобильной разработки

### #226.java FX packager

типа делаем под виндовсом пакуем открываем под линуксом

небольшой самоисполняемый jar архивчик

### #227. добавление фигур в java FX

..........

### #228.canvas

специальная штука для отрисовки сложных фигур

### #229. java FX properties and binding

некие обертки для переменных в фх с некоей возможностью управлять ими

### #230. java FX layout

бонус есть инфа по добавлению неких отступов между элементами

### #231. java FX UI controls

коротка рассказывает о большинстве элементов

### #232. java FX кастомный контрол control

чтото на узбекском

### #233. java FX обработка событий

учим батон реагировать на события

большое кол.методов взаимодействия с мышью

### #234. java FX работа с изображениями

ну тут понятно все

### #235. javaFX анимации

оч интересно

программировали анимацию фигурок

### #236. javaFX scene builder

ну типа гуи форма но только для фх

### #237. javaFX CSS

стили для фх

стили написаны на scc

пытался написать свой стиль

### #238. javaFX FXML

продолжение предыдущей темы

данные формы FX хранятся в прототипе xml так более читабельно

добавляем обработчики событий

### #239. javaFX медиа

запуск видео и аудио

### #240. javaFX отображение html и javascript в WebWiew

ну понятно немного интересно

### #241. javaFX графики charts

рисуем круговые и линейные графики

### #242. javaFX работа с 3D

пытались цилиндр двигать по экрану

### #243. javaFX коллекции в фх

у него есть свои лист и мапа которые фиксируют какието изменения???

могут скопировать обычный лист

### #244. javaFX concurency

у фх есть свои потоки которые позволяют взаимодействовать с приложением фх

### #245. javaFX swing and fx

скрещиваем свинг и фх

### Java 9 L1 установка JDC

............

### Java 9 L1 JShell

Типа консолька прям в которой код писать можно

Совсем не интересно

### Java 9 L3 ProcessHandler

Непонятная ерунда

Обработчик процессов

Типа если надо процесс усыпить на 1час или ещё что-нибудь

### Java 9 L4 приватные методы в интерфейсах

Появилась возможность добавлять приватные методы

### Java 9 L5 коллекции

Появилась возможность добавлять элементы в коллекцию через параметры прям там их через запятую забивать

List<Integer> list=new List.of(1,2,3,4,5,6,7);

Map<String,string> map=new Map.of("key1","value1","key2","value2");

### Java 9 L6 StackWalker

Позволяет проходить стек вызовов и типа можно вывести их на экран

Типа из майна в класс из класса в метод...

### Java 9 L7 modules

Непонятная ерунда рядом с файлом .java modul\_info.java

Пытались создать таких несколько штук

Еле получилось

### Java 9 L8 изменения try с ресурсами

OutputStream outputStream=new FileOutputStream(temp.txt);

try(outputStream){

outputStream.write("a");}catch(Exeption ignore){}

### Java 9 L9

### #246. RegEx что такое рег.выражения

могут например проверять введенный емэйл на соответствие синтаксиса

помогает при поиске в тексте

### #247. RegEx буквальные выражения

ищет текст введеный буква в букву

напр.1+1=2

+ он посчитает за спец символ его надо заэкранировать

\\+ так экранируют в джаве

\+ а так везде

### #248. RegEx наборы символов

[a-c]узкий диапазон [a-z] широкий диапазон

[0-9]

паттерн [0-9a-zA-U%\_#] найдет все подряд

g[^u] за буквой g должен идти любой символ кроме u крышка может стоять только в начале

[////x] пытаемся найти слэш и x

короткие выражения "\\d" найдет все цифры

"\\w" все буквы маленькие и большие

"\\s" табуляцию и пробел новая строка

[^w\] отрицание регулярного выражения найдет все кроме букв

### #249. RegEx точка

точка отвечает за любой символ даже отсутствие символа кроме конца строки

### #250. RegEx границы слов

"\bbird\b найдет отдельное слово а не вырвет из контекста

### #251. RegEx начало и конец строки

^a найдет а вначале строки это не отрицание без [] скобочек

^\\d\*$ звездочка подраз.любое кол.сим.даже 0 а + хотябы один символ

^\\d+$ найдет строку где в строке одни цифры

$конец строки

\\Z \\A тоже начало и конец строки

### #252. RegEx Pipeline

"cat|dog" найдет кота или кошку

#253. RegEx optional items знак вопроса

"Nov(vember)?" последних 6 букв может быть а может не быть

"Nov(vember)??" отменяет жадность оператора найдя первые 3 буквы он не будет сверять дальше

### #254. RegEx символы звездочка и плюс

+ от1 до бесконечности

\* от 0 до беск

{2,4} от 2 до 4 по количеству

{,4} от 0 до 4

{2,} от 2 до беск

{4} строго 4 раза

### #255. RegEx группы

(......) группа выражений

(?:.....) больше не будем ссылаться к этой группе

\\1 обращаемся к первой группе

\\0 обращаемся ко всем группам

$1 заменяем на первую группу

### #256. RegEx именованные группы

(?<group>a) дали имя группе которая ищет "а"

${group} обратились к этой группе

### #257. RegEx unicode

system.out.println("\u0061\u0300");уникоды буквы а и ударения

system.out.println("\u00E0");уникод французской а с ударением сразу

там разное в виде в том числе програмируемый поиск тайландских символов

### #258. RegEx режимы рег.выражений

можно прям в коде включать и отключать разные режимы например чтение всяких коментариев

### #259. RegEx possesive quantifier

так же как и  greedy не останавливается на найденном а идет дальше но дойдя до конца строки не возвращается а сваливается иногда экономит время

### #260. RegEx atomic group

(?>integer|insert|in)\\b найдя первое выражение он уже не будет проверять остальные на "или" и пойдет дальше

### #261. RegEx look around

lookhead ищет то что впереди

"q(?=u)" найдет те q перед которыми стоит u

"q(?!u)"наоборот чтобы не стояла u

"(?<=u)q" lookbehind

"(?<!u)q"

### #262. RegEx множественные требования

ну там типа найти слово перед которым чтото было или после

один из примеров

"\\b(?=\\w{6,12}\\b)\\w{0,9}(cat|dog|mause)\\w\*"

### #263. RegEx if then else

"(a)?b(?(1)c|d)" если а была найдена будет сверять с с есле нет то с d

для джавы доп ресурсы надо качать

### #264. RegEx posix simbols

примеры

\p{Digit} вместо [0-9]

\p{Punct} вместо &-%:!'(ґ\*%@/):?\%

список выражений есть в видео

### #265. RegEx коментарии

#comments,Pattern.COMMENTS потдерж в джаве и идет до конца строки

(?x)a#comments

(?x) непон штука вкл комментарии и начин игнорировать пробелы в тексте

(?#comments) прийдется докачивать библиотечку

### #266. RegEx примеры

разные примерчики

### #267. RegEx тренировались на сайте regexone

....

### #268. Patterns. что такое паттерны

.... всего около 24 паттерна появляются новые

### #269. Patterns. Factory

просто класс который создает новые классы return new EnyClass()

### #270. Patterns. Abstract Factory

фабрика фабрик классов

все просто

но только интерфейс

### #271. Patterns. Factory Metod

метод создает собственный класс

return new самсебя()

### #272. Patterns. singleton

package patterns.creational;

public class SingeltonLesson {

    public static void main(String[] args) {

        Singelton singelton = Singelton.getInstance();

    }

}

class Singelton {

    static Singelton singelton = new Singelton();

    private Singelton() {}

    public static Singelton getInstance() {

        return singelton;

    }

}

//более правильная реализация

public class Singleton {

    private static Singleton INSTANCE;

  private Singleton() {

  }

    public static Singleton getInstance() {

        if (INSTANCE == null) {

            synchronized (Singleton.class) {

                if (INSTANCE == null) {

                    INSTANCE = new Singleton();

                }

            }

        }

        return INSTANCE;

    }

}

Плюсы:

Ленивая инициализация.(создастся при необходимости)

Потокобезопасность

Высокая производительность в многопоточной среде(потомучто синхр.не весь метод а только часть его)

В целом синглтон делает именно то, что от него ждут:

Дает гарантию, что у класса будет всего один экземпляр класса.

Предоставляет глобальную точку доступа к экземпляру данного класса.

### #273. Patterns. Builder

это статический класс который наполняет переменные других классов смыслом при их создании типа забивает туда кучу параметров вместо нас но это не точно

и походу он находится внутри того класса который обслуживает

package patterns.creational;

public class BuilderLesson {

    public static void main(String[] args) {

        SportCar sportCar = new SportCar.Builder("Audi").setColor("green").setMaxSpeed(300).build();      System.out.println(sportCar.getName()); System.out.println(sportCar.getColor()); System.out.println(sportCar.getMaxSpeed());

    }

}

class SportCar {

    private String name;

    private String color;

    private int maxSpeed;

    public String getName() {

        return name;

    }

    public String getColor() {

        return color;

    }

    public int getMaxSpeed() {

        return maxSpeed;

    }

    private SportCar(Builder builder) {

        this.name = builder.name;

        this.color = builder.color;

        this.maxSpeed = builder.maxSpeed;

    }

    static class Builder {

        private String name;

        private String color;

        private int maxSpeed;

        public Builder(String name) {

            this.name = name;

        }

        public Builder setColor(String color) {

            this.color = color;

            return this;

        }

        public Builder setMaxSpeed(int maxSpeed) {

            this.maxSpeed = maxSpeed;

            return this;

        }

        public SportCar build() {

            return new SportCar(this);

        }

    }

}

### #274. Patterns. Prototype

это класс который умеет сам себя клонировать обычно используется в базах данных чтобы данные закэшировать и потом их уже дергать так быстрее

package patterns.creational;

public class PrototypeLesson {

    public static void main(String[] args) throws CloneNotSupportedException {

        Cash cash = new Cash();

        cash.setStudent(new Student());

        Student student = cash.getStudent();

    }

}

class Student implements Cloneable {

    @Override

    public Student clone() throws CloneNotSupportedException {

        return (Student)super.clone();

    }

}

class Cash {

    private Student student;

    public Student getStudent() throws CloneNotSupportedException {

        return student.clone();

    }

    public void setStudent(Student student) {

        this.student = student;

    }

}

### #275. Patterns. Object pool

короче один класс занимается обустройством в коллекциях мапах листов эррэев других классов

но пример туговат для восприятия

package patterns.creational;

import java.util.ArrayList;

import java.util.List;

public class ObjectPoolLesson {

    public static void main(String[] args) {

        ObjectPool objectPool = new ObjectPool();

        PooledObject pooledObject = objectPool.getObject(); objectPool.releaseObject(pooledObject);

    }

}

class PooledObject {}

class ObjectPool {

    List<PooledObject> free = new ArrayList<>();

    List<PooledObject> used = new ArrayList<>();

    public PooledObject getObject() {

        if(free.isEmpty()) {

            PooledObject pooledObject = new PooledObject();

            free.add(pooledObject);

            return pooledObject;

        } else {

            PooledObject pooledObject = free.get(0);

            used.add(pooledObject);

            free.remove(pooledObject);

            return pooledObject;

        }

    }

    public void releaseObject(PooledObject pooledObject) {

        used.remove(pooledObject);

        free.add(pooledObject);

    }

}

### #276. Patterns. Adapter

Это обертка

если есть классы с немного отличающейся реализацией то можно один из классов поместить в обертку которая сделает его похожими на другой класс

package patterns.structural;

public class AdapterLesson {

    public static void main(String[] args) {

        CarWash carWash = new CarWash();

        carWash.washCar(new TrackWrap(new MyTrack()));

    }

}

class TrackWrap implements Car {

    Track track;

    public TrackWrap(Track track) {

        this.track = track;

    }

    @Override

    public void wash() {

        track.clean();

    }

}

interface Track {

    void clean();

}

class MyTrack implements Track {

    @Override

    public void clean() {

        System.out.println("Track is clean");

    }

}

interface Car {

    void wash();

}

class Audi implements Car {

    @Override

    public void wash() {

        System.out.println("wash car");

    }

}

class CarWash {

    public void washCar(Car car) {

        car.wash();

    }

}

### #277. Patterns. Bridje

короче это то о чем мой первый учитель говорил главное преимущество полиморфизма

Есть две большие группы "фигуры" и "цвета"

фигуры экстендим от "абстрактного класса"

цвета имплементим от "интерфейса"

в абстр.кл добавляем:

protected Color color;

   public Shape(Color color) {

this.color = color;  }

а в его наследники

public Rectangle(Color color)

{ super(color);}

в интерфейсе не забудем обозначить методы его наследников

### #278. Patterns. Filter

просто фильтрует листы циклом и записывает в новый лист

оч много вспомогательных обьектов

### #279. Patterns. Composite

композитная (древовидная) структура наподобие реализации дерикторий когда папки содержат папки или файлы

реализовать можно например списками чтобы они содержали в себе еще списки

### #280. Patterns. Decorator

ну типа обертка чтоли класс улучшает общение с другим классом

BufferedInputStream декоратор класса InputStream

и старый класс передается параметром в класс декоратор

### #281. Patterns. Facade

просто кучу кода запихиваем в один обьект чтоб .... ну... понял да?

все просто!!!

### #282. Patterns. FlyWeight

типа если хотим создать новый обьект но точно такойже уже был то вернем тот что уже был и не будем создавать новый сэкономим место

package patterns.structural;

import java.util.HashMap;

import java.util.Map;

public class FlyWeightLesson {

    public static void main(String[] args) {

        FlyWieghtGarage flyWieghtGarage = new FlyWieghtGarage();

        Mercedes mercedes = flyWieghtGarage.getMercedes("green");

        Mercedes mercedes2 = flyWieghtGarage.getMercedes("green");

        System.out.println(mercedes == mercedes2);

    }

}

class Mercedes {

    String color;

}

class FlyWieghtGarage {

    Map<String, Mercedes> map = new HashMap<>();

    Mercedes getMercedes(String color) {

        Mercedes mercedes = map.get(color);

        if(mercedes == null) {

            mercedes = new Mercedes();

            mercedes.color = color;

            map.put(color, mercedes);

        }

        return mercedes;

    }

}

### #283. Patterns. Proxy

некий промежуточный этап

в данном примере класс проху вклинивает свой код перед кодом другого класса

public class ProxyLesson {

    public static void main(String[] args) {

        CarI car = new CarProxy();

        car.drive();

    }

}

interface CarI {

    void drive();

}

class CarProxy implements CarI {

    CarI car = new Reno();

    @Override

    public void drive() {

        System.out.println("proxy code");

        car.drive();

    }

}

class Reno implements CarI {

    @Override

    public void drive() {

        System.out.println("drive reno");

    }

}

Для каких задач лучше использовать Proxy:

Вот несколько ситуаций, в которых тебе точно пригодится этот паттерн:

Кэширование.

Отложенная реализация, также известная как ленивая. Зачем загружать объект сразу, если можно загрузить его по мере необходимости?

Логирование запросов.

Промежуточные проверки данных и доступа.

Запуск параллельных потоков обработки.

Запись или подсчет истории обращения.

Есть и другие сценарии использования.

На первый взгляд, Заместитель делает то же, что и Фасад, но это не так. У Заместителя есть тот же интерфейс, что и у сервисного объекта.

Также не нужно путать паттерн с Декоратором или Адаптером. Декоратор предоставляет расширенный интерфейс, а Адаптер — альтернативный.

### #284. Patterns. Chain of responsibility

цепочка ответственности

один обьект берет запрос и передает его другому запросу

например если один не смог обработать запрос то передаст другому и так по цепочке

ну в примере не ножко другое:

public class ChainOfResponsibilityLesson {

    public static void main(String[] args) {

        MessagePrinter messagePrinter = new ConsoleMessagePrinter();

        FileMessagePrinter fileMessagePrinter = new FileMessagePrinter(); messagePrinter.setNexMessagePrinter(fileMessagePrinter); fileMessagePrinter.setNexMessagePrinter(new DbMessagePrinter());

        messagePrinter.print("hello");

    }

}

abstract class MessagePrinter {

    MessagePrinter nexMessagePrinter;

    void setNexMessagePrinter(MessagePrinter messagePrinter) {

        nexMessagePrinter = messagePrinter;

    }

    void print(String message) {

        printMessage(message);

        if(nexMessagePrinter != null) {

            nexMessagePrinter.print(message);

        }

    }

    abstract void printMessage(String message);

}

class ConsoleMessagePrinter extends MessagePrinter {

    @Override

    void printMessage(String message) {

        System.out.println("print to console: " + message);

    }

}

class FileMessagePrinter extends MessagePrinter {

    @Override

    void printMessage(String message) {

        System.out.println("print to file: " + message);

    }

}

class DbMessagePrinter extends MessagePrinter {

    @Override

    void printMessage(String message) {

        System.out.println("print to db: " + message);

    }

}

### #285. Patterns. Command

Где это применяется?

По цели ясно, что и для чего это нужно, а именно: в ситуации, когда нужно отделить конкретное исполнение, это очень удобно. Чтоб использование каких-то функций не зависело от конкретной реализации и ее можно было изменять без ущерба для системы.

import java.util.ArrayList;

import java.util.List;

public class CommandLesson {

    public static void main(String[] args) {

        Receiver receiver = new Receiver();

        receiver.addCommand(new MouseClick());

        receiver.addCommand(new MousePress());

        receiver.RunCommand();

    }

}

interface Command {

    void execute();

    void revert();

}

class MouseClick implements Command {

    @Override

    public void execute() {

        System.out.println("click command");

    }

    @Override

    public void revert() {

        System.out.println("revert");

    }

}

class MousePress implements Command {

    @Override

    public void execute() {

        System.out.println("click mouse pressed");

    }

    @Override

    public void revert() {

        System.out.println("revert");

    }

}

class Receiver {

    List<Command> commands = new ArrayList<>();

    void addCommand(Command command) {

        commands.add(command);

    }

    void RunCommand() {

        try {

            for (Command command : commands) {

                command.execute();

            }

        } catch (Exception e) {

            for (Command command : commands) {

                command.revert();

            }

        }

    }

}

### #286. Patterns. Interpreter

разбивали стрингу выражение на цифры и символы а потом считали все это

логика наворочена

короче перегоняли один формат текста в другой

### #287. Patterns. Iterator

тот самый который по массивам ходит

и перебирает

### #288. Patterns. Mediator

обьяснил на основе radiobatton класс который следит чтобы только один был в состоянии true

также пример что если надо разослать сообщения конкретным пользователям программы а не всем

### #289. Patterns. Momento

типа сохранение данных обьекта и потом их восстановление

### #290. Patterns. Observable

идея в том что при изменении статуса одного обьекта он уведомляет об этом остальные обьекты

### #291. Patterns. State

обьект в зависимости от статуса записанного в переменную меняет свое поведение

### #292. Patterns. Null object

вместо:

return null;

чтобы не вылетало NullPointerExeption делаем пустой класс:

return NoAnimal();

в котором все методы с пустой реализацией

### #293. Patterns. Strategy

на примере магазина например есть день скидок есть обычный день и еще чегото там

и создается класс в который передается опр.стратегия параметрои и оно уже там само

public class StrategyLesson {

    public static void main(String[] args) {

        double startPrice = 100;

        ContextStrategy context = new ContextStrategy(new HalfPrice());

        double price = context.getPrice(startPrice);

        System.out.println(price);

    }

}

interface Strategy {

    double getPrice(double price);

}

class FullPrice implements Strategy {

    @Override

    public double getPrice(double price) {

        return price;

    }

}

class HalfPrice implements Strategy {

    @Override

    public double getPrice(double price) {

        return price \* 0.5;

    }

}

class ContextStrategy {

    Strategy strategy;

    public ContextStrategy(Strategy strategy) {

        this.strategy = strategy;

    }

    public double getPrice(double price) {

        return strategy.getPrice(price);

    }

}

### #294. Patterns. Template

шаблонный метод

идея настолько тупая и часто используемая что понять ее не возможно

### #295. Patterns. Visitir

допустим у нас уже есть устоявшаяся структура "классов" и изменять их чтото делать с ними добавлять новые методы никто не хочет изза стабильности этой конструкции

тогда создаетсядругой класс и в его те самые нехватающие методы "классы" передаются в качестве параметров

визитор - посетитель

### #296. Patterns. MVC

Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных.

Model - часть содержащая в себе функциональную бизнес-логику приложения.

Представление (View) типа UI интерфейс к примеру

Сontroller -

### #297. Patterns. Business Delegate

некое кэширование данных и использование промежуточных методов для меньшей нагрузки на интернет сеть

Преимущества :

Бизнес-делегирование уменьшает связь между клиентами уровня презентации и бизнес-службами.

Бизнес-делегат скрывает базовые детали реализации бизнес-службы.

Недостатки :

Обслуживание из-за дополнительного уровня, который увеличивает количество классов в приложении.

### #298. Patterns. Composite Entity

"составной обьект"

когда в сети есть несколько взаимосвязанных обьектов чтобы их от туда не дергать по одиночке существует отдельный класс который обьеденит их и отправит одним обьектом

### #299. Patterns. DAO

если например раньше систепа работала с базой данных а потом решили работать с файловой системой то чтобы не переписывать код во всем проекте делают взаимозаменяемый код имлеменченный от одного интерфейса с предыдущим классом

=========================

Шаблон объекта доступа к данным (DAO)

Чтобы модель предметной области приложения полностью не зависела от базы данных

## @#300. Patterns. Front Controller

типа команды присылаются в виде строки слова DispetcherServlet строку определяет и челез свитч находит нужный метод

В видео есть хороший код

import java.util.Scanner;

public class FrontControllerLesson {

    public static void main(String[] args) {

        try(Scanner scanner = new Scanner(System.in)){

            while (scanner.hasNext()) {

                String nextLine = scanner.nextLine();

                new Thread() {

                    @Override

                    public void run() {

                        new DispatcherServlet().process(nextLine);

                    }

                }.start();

            }

        }

    }

}

class DispatcherServlet {

    void process(String path) {

        switch (path) {

            case "home" : new HomeController().show(); break;

            case "user" : new UserController().show(); break;

            default: new DefaultController().show();

        }

    }

}

class HomeController {

    void show() {

        System.out.println("home");

    }

}

class UserController {

    void show() {

        System.out.println("user");

    }

}

class DefaultController {

    void show() {

        System.out.println("error");

    }

}

### @#301. Patterns. Intercepting Filter

все команды приходящие из сети перед запуском проходят фильтр

в видео есть хороший код

### #302. Patterns. Servise Lokator

типа клиент вызывает сервис какойто

и локатор проверяет из кеша не вызывался ли он раньше если вызывался то выдаст из кеша если нет то создаст новый

ктото из ... посчитал его антипаттерном

и говорит избегать его так как имеются ошибки с временными зависимостями

## #303. Patterns. Transfer Object

это самый простой класс с примитивными переменными

чтобы их не дергать по одиночке передают по сети сразу весь обьект

## #304.SOLID что такое солид

....

## #305.SOLID single responcibility

единственная ответсвенность

класс должен иметь только одну причину чтобы чтото поменять в нем

иначе разбибаем класс на подсемейство одного интерфейса

## #306.SOLID. open code principle

обьекты открыты для расширения и закрыты для модификации

если например захотелось добавить больше классов то пожалуйста

если изменить код майна или какой функции то нельзя

## #307.SOLID. Liskov principle

при наследовании не должно изменяться поведение парента

## #308.SOLID. Interface sugregation

разделение интерфейса

если класс не собирается использовать все методы интерфейса то у него будет пустой переопределенный метод

в таком случае из одного лучше сделать 2 интерфейса

## #309.SOLID. Dependency Inversion

инверсия зависимости

интерфейсы интерфейсы интерфейсы

они переворачивают зависимость

например если две массивных структуры вдруг вздумали пользоваться одним классом то в случае возникновения багов они распространятся на всю систему

тогда лучше этот класс оградить интерфейсами двумя по одному с каждой стороны и ремонтировать уже прийдется только тот класс а не всю систему

Или например в процессе написания кода выяснилось что забыли какойто метод учесть при разработке

имплементим наш участок кода от интерфейса и он нам сам напомнит о том что забыли

и продолжаем писать код дальше

### #310.Generics.

немного о дженериках

приведению типов и шаблонных типов

### #311. Наследование generics

балуемся с наследованием разных классов с <T,Y,X>

короче он шаблонные классы дженериками называет

### #312.Generic методы

<T> T metod(T t){return t}

public <T> void metod(T t){}

### #313.Bounds generic

ограничения дженериков

В конструкторе класса дженериков особо нельзя какието методы вызвать кроме самых простых

Тогда класс имплементим от какогонибудь интерфейса и тогда он сможет вызывать его методы

и вообще можно расширить функционал поунаследовав его от разных классов

### #314. Whild card

о том что не следует передавать дженерик код в недженерик метод

потомучто там может быть старый код который написан до джавы1.5

### #315(Whild card)перепутаны видео

"дикая карта" пришло из покера

Cell<? super Toiota> cell=new Cell<Car>;

Cell<? extends Toiota> cell=new Cell<Car>;

будет разное поведение к геттерам сеттерам наследникам или парент классам

Полезно например при работе со списками в которых лежат разнородные обьекты

### #316. ArrayList and LinkedList

linkedList используется больше для очередей

### #317. Stack, Queue, Dequeue

стэк первый зашел последний вышел

Stack<string> stack=new Stack<>();

Queue с одной стороны ложим с другой достаем

Deque = new ArrayDeque() or LinckedList()

двунаправленная очередь

### #318.BinarySearch

некий метод позволяет искать элемент в коллекции возвращает индекс

коллекция должна быть предварительно отсортирована

Collections.sort(list);

Integer index=Collections.binarySearch(list,5);

### #319. выбрасывание ошибок методами

все виды эксепшенов имеют иерархию и выбрасываются в методах наследников и парентов с какойто закономерностью

### #320. эксэпшены и Pipeline operator

catch(IOExseption|SQLExseption)

### #321.как работает AutuClosable

надо пересмотреть все что связано с эксэпшенами но только в гугле

### #322.Аssertions

проверка на ошибки

по умолчанию отключена

надо включать

прямо в коде пишем assert x<0

и если да то вывалится булеан false

вывалится ошибка

можно прикрепить коментарий

### #323. Immutable

неизменяемые обьекты

string нельзя изменить только заменить на новое значение

Datе ссылочное поле его только клонируем

классы с такими полями лучше определять как final чтобы наследники не могли чего переопределить в геттерах

и да сеттеров у таких полей нет

### @#324.ReadWrite lock

ReadWriteLock lock = new ReentrantReadWriteLock();

lock.writelock.lock();

buyTime++;//блок на запись

lock.writelock.unlock();

lock.readlock.lock();

sout(getBuyTime());

lock.readlock.unlock();

//блок на чтение будет открыт пока к ресурсу подходят для чтения но если хоть один откроет лок на запись остальным будет заблочено

хорошая статья про синхронизацию:

https://tproger.ru/translations/java8-concurrency-tutorial-2/

### #325.что такое функциональное программирование

говорили о том чтобы не было скрытых параметров

типа если в параметры мы передаем 2 значения кроме этого внутри функции нельзя например вызвать (new Date())

потому как он будет скрыт от тестировщиков и они не смогут его замокать

и функциональное программирование призвано бороться с такими случаями

## @#326.лямбда

()->System.out.println("hello");

(x)->System.out.println("hello");//с параметром, без указания типа потому как сама по себе лямбда не живет

x->System.out.println("hello");//если параметр один

если мы создаем класс чтобы переопределить только один метод :

Runnable runnable = ()->System.out.println("hello");

мы переопределили метод run()

ActionListener acls= (event)->System.out.println("event");

переопределили метод actionPerformed

java сама догадается какой метод мы переопределили и что параметром передали ActionEvent

()->{};

String s="hello";//указывать что это final переменная необязательно но это так

задается один раз и переопределить здесь или в функции дальше нельзя

ActionListener acls= (event)->System.out.println(s);

## @#327. функциональные интерфейсы

возможность использовать лямбда выражение несколько раз

Predicate<Integer> predicate=x->x>5;

System.out.println(predicate.test(10));//вернет true

почти тоже самое что мы и с runnable в прошлом уроке делали

типа можно и по старинке с @Override написать тоже самое

<Integer> это входной параметр в функцию

Кроме предиката есть еще штук 5 частоупотребляемых разница в них только во входных параметрах и выходных а так их оч много

показан способ создать свой собственный интерфейс

### #328. Streams

это введений 8й джавы

у каждого листа мапы и аррэя есть .... .stream(); функция

Stream<String> stringStream = list.stream();

        stringStream.forEach(x -> System.out.println(x));

stream испульзуется только один раз на второй раз вылетает эксепшен

можно все забить в одну строчку:

list.stream().forEach(x -> System.out.println(x));

фильтр:

list.filtr(x->x.equals("one"));//это lazy метод выполнится после команды .count()

превращение в коллекцию:

Set<String> set=list.collect(Collectors(toSet());//и там дофига методов в любые коллекции

list.map(.....); трансформирует обьекты напримел из интов в стринги из маленьких букв в литеры

list.flatMap(....); //обьединяет несколько коллекций в одну

есть min и max но синтаксис пересмотреть надо

reduse() сложит все числа

Streamof(1,2,3,4,5)создаем поток по ходу без использования коллекций

sorted()

limit(100) например из базы хотим только сотню значений

disting() выводит уникальные значения повторяющиеся делает как 1

skip(50) первых 50 значений пропустит

mapToInt превращает все в инты для экономии памяти

peek преобразовывает в мапу

все интересно надо пересматривать

## @#329. default и static методы

до джавы восьмой не было возможности писать реализацию кода в интерфейсах

и чтобы не поламать прги написанные на ранних версиях такую возможность добавили но только перед методом писали default или static но первое нас интересует больше

его можно как переопределять так и использовать в исходном виде

будут проблемы с множественным наследованием если у разных интерфейсов будут одинаковые классы

## @#330. Optional

введена чтобы вместо return null; писать return Optional.empty();

борется с NullPointerExseption

Optional<String> optional = getO();

        if(optional.isPresent()) {

            System.out.println(optional.get());

        }

    static Optional<String> getO() {

        if(false){

            return Optional.of("Hello");

        }

        return Optional.empty();

    }

### #331.metod references

ссылки на методы

типа на пару символов писать покороче

Stream.of("one", "two").map(x -> x.toUpperCase());

        Stream.of("one", "two").map(String::toUpperCase);

        Stream.of(new Student("Max"), new Student("Mike")).map(Student::getName).forEach(System.out::println);

        Stream.of("Jhon", "Kent").map(Student::new).forEach(x -> System.out.println(x.getName()));

## @#332. collertor

может не только превращать поток в листы но и например делить коллекцию на мапу на чет и нечет изаписывать состояние в ключи

получить мах мин число

например сгрупировать если встречаются одинаковые числа

например если надо завернуть числа в скобочки {} или[] и разделить запятой

если надо их например в файл запаковать

например сгруппировать одинаковые и посчитать количество таких вхождений

## @#333. parallel stream

stream() это в один поток

parallelStream() разобьется на несколько ядер не на конкурентные потоки

применять в зависимости от ситуации

есть лэйзи методы например фильтр мап и др.

при разнопоточном стриме будет бардак в потоке данных

взависимости от ситуации

если просто например сумму посчитать то и такое сойдет

## @#334. Local Date

пришел на смену старым Date и Calendar

## @#335. как работают лямбда выражения

сам по себе стреам не запускается для этого нужен терминальный оператор

forEach,collected,рекьюз

supplier можно использовать для многократного использования стрима

forkJoinPool используется для оптимального подбора количества потоков под количество ядер он вшит в параллельный стрим и оставляет одно ядро про запас

сортировка никогда не смешивается с рстальными операциями

### #336. equals и HashCode

о том какие поля можно сравнивать

можно сравнивать только константные поля

например машину могут перекрасить след.по цвету сравнивать нельзя а по номеру можно

### #337. Clonnable

что если не переопределять метод clone?

имплементить класс клонабл обязательно а во переопределять метод clone по мере необходимости если нужно глубокое клонирование в классе есть другие классы и др.

### #338.пулл оберток над примитивами

(Integer) a==b славнит по ссылкам по идее должен вернуть false но до 127 все рано true потому что до этого количества числа будут лежать в пуле

и да при каждом изменении числа будет создаваться новый обьект новая ссылка

правильное сравнение по значению:

a.equals(b)

### #339. коллекции коллекции коллекции

set уникальные значения

queue очереди

map понятно

оказывается можно положить значение по ключу null

### #340. List

ArrayList просто массив

LincedList итерируемся по ссылкам единственное преимущество он имплементирует очереди

Vector тотже LincedList только все методы синхронизированы

sinchronizedList синхронизированная обертка для наверное только ArrayLista

Stack ну понятно тоже синхронизирован

фориачем изменять коллекции нельзя будут проблемы с итератором

только обычным фором

при сортировке коллекций обьекты должны имплементить Comparebel и переопределять его метод

бинарный поиск ищет при помощи бинарного дерева перед(делит массив на половины) употреблением обязательно отсортировать

Collections.binarySearch(arrayList, "b")

 линкед лист можно перегонять в аррэй лист с использованием обычного массива(подробности в видео)

массив можно просто забить в sout по умолчанию выполниться toString и выведется весь массив

можно сравнивать массивы

arrayLict.equals(StringList)

list можно запихнуть в мапу но могут быть ошибки

### #341. set

неповторяющиеся значения

hashSet хранит неупорядоченно не в том порядке в каком поступило(на основе мапы)

linckedHashSet хранит в том порядке в котором поступило

treeSet отсортировывает по алфавиту или по числам(на основе навигейбалмапы)при поиске и сравнении нужно переопределять компаир а не иквалс и хэшкод.

### #342. Queue очередь

dequeue двунаправленная очередь

строится чаще из линкедлиста но может и из других

построенный из аррэйлиста работает быстрее линкед листа и стэка

priorityQueue сортирует элементы

blockingQueue блокирует если очередь забита до определенного числа используется в многопоточном

### #343. Map

для хранения обьектов нужно переопределять equals и HashCode а для древовидной ComparebleTo

LinckedHashMap запоминает порядок складывания в нее

HashTable все методы синхронизированы

synchronizedMap обертка для любой мапы

loadFactor это об коллизиях

по умолчанию там стоит 0.75 когда таблица забьется на 75% мапа перехешируется чтобы не было коллизий

navigeibl map навигация по мапе мозно выбирать из конкретного отрезка значений а не всу мапу можно задать значение и посмотреть какой элемент ближайший до или после этого значения

hashMap занимает больше места изза loadFacktora изза неполностью забитой таблицы

### #344. сложность алгоритмов поиска по коллекции

//0(1) мы знаем ячнйку где оно лежит и достаем

//0 (lon(n)) логарифмическая типа делим пополам и нужную область еще раз пополам

// 0(n) линейная просто итерируемся по всей коллекции

// 0 (n\*log(n))сначала отсортируем потом искать (звазилинейный поиск)

### #345. Concurrency package

добавили кучу целую новых коллекций построенных не на синхронизации метода а на локах этотвсе будет быстрее

CopyWriteArrayList блокирует при записи

использует transient(переменная пропускается при сериализации) volatile(переменная выполняется атомарно и не кешируется)

CopyWriteArraySet тоже самое

### #346. volatile, happens befor

volatile дает указание процессору не кешировать переменную а писать напрямую в память + атомарность операции

model mamory ???

happens befor запрещает процессору менять операции местами

дело в том что устройство памяти такое что иногда при обьявлении переменных он меняет операции местами

### #347. double check

возникает при одновременном создании двумя потоками статического класса

синхронайзед не помагает

помагает volatile

### #348. типы reference

strong referens обвчная ссылка

далее ссылки ссылающиеся на null помещаются в массивы

SoftReferense массив почистиит ссылки если не будет хватать места в памяти

WeekReferense будет почищен при первом запуске гарбадж коллектора

FantomReferense даст информацию была ли почищена ссылка

### @#349модель памяти java и алгоритмы сборщиков мусора

инфы много...

память делится на стек хип ииии.. пересматривай

### #350. как поместить стрингу в пул стрингов

это наподобие с интами до 127

сравниваем только .equals()

но применим .intern() она поместится в пул стрингов и можно сравнивать по ==

### #351. размещение обьектов вне хипа

ByteBuffer позволяет залезть в нативную память считать данные с мышки выделить память под наши обьекты и как в старых добрых с++  .clear()

Unsafe та же история превращаем джаву в с++ а байты почти на канкуляторе считаем

это все для высоконагруженных приложений

### #352. распределение нагрузки load ballansing

сервер делится на разные машины и одна машина распределяет таски другим машинам

есть несколько подходов распределения о них в видео...

### #353. immutable коллекции

unmodifiebleList

unmodifiebleSet

unmodifiebleMap

никаких операций по удалению и добавлению проводить нельзя

создаются через другие обычные листы

но в них все равно можно залезть если осталась ссылка на старый лист:)

### #354. высоконагруженная архитектура

сначала можем все на один сервер запихнуть

если количество запросов растет

1 сначала выносим sql на отдельный сервер

2 разделяем php и подключаем load ballanser

3 делаем кэш базы данных

4 dns баллансировка

5 подгрузку тяжелых файлов например картинок выносим на отдельный сервер

6 все непонятные тяжелые операции

выносим

### #355. шардинг и репликация баз данных

1 можно сделать две бд одина мастер на нее приходит запись

мастер копируется в слэйв с него идет чтение

или несколько слейвов

2 можно например разного рода данные хранить в разных местах например имена пользователей в одном месте их другие данные в другом

3 можно поделить на две базы в одной четные данные в другой нечетные

### #356.создание индексов в базе данных

типа надо отсортировать таблицу по конкретному столбцу чтобы применять логарифмический поиск

но как??

CREATE INDEX person\_name\_idx

ON person(name)

ускоряет чтение

существенно замедляет запись новых данных т.к все индексы прийдется переписать

### #357. synchronized is bad

volatyle и обычная синхронизация увеличивает время работы кода от 100 до 400 раз при миллиарде раз использований

### #358. GRASP паттерны

чтото типа SOLID но только довольно туповатый набор

единственное полезное это hitgh cohesion and low coupling

### #359. CAP теорема

о связи двух серверов

работает только один из двух вариантов

либа один работает на запись другой на чтение и между ними есть связь

или между ними нет связи и оба работают на запись и на чтение

### #360. Externalizer

наподобие Serialisedle

сохраняет обьекты в файл и достает обратно

более точная настройка

быстрее работает

минус больше кода

### #361. immutable классы

безопасны в многопоточной среде

не должен иметь сеттеров

простые переменные просто задаются через конструктор один раз

другие классы клонируются

листы перезаписываются в новый и делаются unmodifiableList

в геттерах мы их просто возвращаем, но класс возвращаем .clone()

рроблема с Оdject он не клонируется над ним можно создать обертку но лучше его просто не добавлять туда

### #362. как обнаружить dead lock

при помощи java VirtualVM идет в комплекте с jdk

можно ручками класс ThreadMXBean подробности на прим 7 стр.кода в видео

### #363. ThreadLocal

ThreadLocal<String>threadLocal=new ThreadLocal<>();

поток{

threadLocal.set("hello");

}

поток2{

threadLocal.set("helloсроипап");

}

типа какаято локальная переменная которая будет использоваться в конкретном потоке

### #364. изучаем intellije idiю

help->keyMap...

там будет список всех быстрых команд

Productivity Guade скажет сколько раз ты  контрл v использовал за жизнь или другие команды

контрл альт в допишет левую часть перед new:)

шифт ф6 можно переименовать везде

!!!!!!!!!!!контрл+альт+шифт+U построит диаргамму классов и связей

контрл шифт поиск файлов по всему проекту

в этом продуктивити гайде можно самому посмотреть там визуальное представление есть и как и зачем обьясняют

### #365. Best Practice

вводный урок

### #365. Best Practice 2