**Import …** - importing elements

**Import …\*** - importing all elements from the package

**<T>** void **…(T** temp**){…}** – creating generic methods, constructors

**Class …<T extends … & …>** - extending generics (variable will have methods from that class)

**Public void …(… <? extends** or **super E>)** – allow all elements extended from this or its child’s

**Instanceof** – operator, which checks if this class extends another one

**Volatile …** - variable which is not saving in cash, only at main memory of cpu (for avoiding exceptions with threads, when you want to write and read from variable at one time, because read is from main memory, when write is in cash)

To create methods in interfaces and set them, write before type of function **default**

**Scanner** scanner **= new Scanner(System.*in*);** - scanner variable (console in intelIdea)

Type … = scanner**.nextLine();** - similar to cin>> in c++

**Console** console **= System.console()** – variable for input in console (don’t know how to use)

Type … = console**.readLine()** – input in console

**<T, U, …>** - set method or class as multivariable (public class … <T> {…} , static <T> T …(){…} )

Class has to implement class “comparable” and override method compare to be able to compare

**<? Super/extends …>** - when you use collections in methods, you can add this to let pc understand that you will be using parents of child of this class

If you want to get inner class from class you have to write: **inner\_class … = new class().new inner\_class();**

**Strictfp class …** - class with only float values

**Abstract class …** - abstract class

**Final class …**  - final class

**void foo(char a, int... c)** – if you want to add unlimited value

в конструкторе можно вызвать другой конструктор класса методом **this(…);**

**enum имя {… , …}**  - объединения, могут иметь конструкторы и параметры **enum имя {…(…) , …(…); имя() {…}}**

**Color. …** - colors

In parametrs add **–ea** –enable exceptions for using **assert true/fase : “…”;** - will throw exception in that line if false (include logic expression) with message

URLClassLoader loader = new URLClassLoader(new URL[]{new URL("")}); - get variable with classes   
Class cl = loader.loadClass("name"); - load class

To create inner class variable from not connected class: **FatherClass.InnerClass** name **= new FatherClass().new InnerClass();**

To create static variable in the inner class it has to be final

Implement **AutoClosable** and **override close** in class to avoid writing close at the end, just add it to the try parametrs

JavaCompiler compiler = ToolProvider.getSystemJavaCompiler(); - create compiler for java code  
int result = compiler.run(inputStream, outputStream, error, "where"); - compile code and return 1 if it is ok

**Data types and operations with them**

cover of types {**Integer Byte Short Long Boolean Character Float Double** }

**Integer.parseInt(…)** – string into the int

**Integer.valueOf(…)** – строку в число Integer

**new Intager(…) – parse int integer**

**try { }catch (){ }finally { }** – finally happens in both cases

**assert(…)** – short version of try catch, it checks if the equation(уравнение) is true and if not it breaks the program (you have to add **–ea** to the VM options for this thing to work)

**DATE**

Date is old, use **LocalDate**, it has more easier methods

**Date** *date* **= new Date()** – creating data element

**Calendar** *calendar* **= Calendar.*getInstance*();** - creating variable*calendar***.setTime(***…***);** - setting time*calendar***.add(Calendar.*DAY\_OF\_MONTH*, …);** - add date to the current (1 element is defined variable of the Calendar and second is number of *calendar***.getTime();** - getting time

**SimpleDateFormat** *simpleDateFormat* **= new SimpleDateFormat("**dd/M/yy HH:m**");** - creating variable, which will be changing format of time (“dd/M/yy HH:m” – this abbreviator you can find by googling <https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html> )*simpleDateFormat***.format(***date***);** - changing date format

**Collections (interface)**

**ArrayList/CopyOnWriteArrayList** – second for multithreads applications, it is same to common ArrayList

**List/LinkedList/ vector** – vector is the same list, but all methods are synchronized for threads

**Queue**

**PriorityQueue** – with sort

**Map/HashMap/LinkedHashMap/TreeMap**

**HashSet** – no repeated elements, random order

**TreeHashSet** – no repeat, always order

**LinkedHashSet** – no repeat, with order

Collections.sort(list); - sorting (must be before binary search)  
Int index = Collections.binarySearch(list, "salsa"); - search for value (searching index of key)

List.get(index); - get element by index

Has methods:

List<...> list = List.of(..., …); - fast set

Iterator<…> … = ….iterator() – get iterator first, then use .hasNext() in loop

**….get(…)** – get by index (ArrayList)

**.add(…)** – add element

**.remove(…)** – remove element by id

**.get(…)** – get element by id

**.size()** - size

**PriorityQueue**

**.poll()** – get first element and delete it from array

**.peek()** – get first element

**Hashmap**

**Map … = new HashMap();** - creating variable

**.entrySet()** – swap into set massive

**.keySet()** – swap keys into Set massive

**STRING**

**String … = new String(“…”);** - add a variable ( if you want to concat another str, it will create new variable in RAM) **don’t use if you are going to change it many times**

**StringBuilder … = new StringBuilder(“…”)** – add changeable variable( if you want to concat another str, it wont create new variable in RAM)

**Clever srtings**

String str = "… {0} … {1, date, short}…{2, choice, 0# no one|1# one pizduk|2# {2} or more pizducs}"; - create str with required variable instead of meanings in {}  
String res = MessageFormat.format(str, "vasok", new GregorianCalendar(2018, 10, 15).getTime(), 2); - set values by order 1, 2, 3 … (special one for time)

**Create two pointers on one heap plase**

**String a = “abc”**

**String b = new String(“abc”).intern** – look for the same

**Functions**

**System.gc();** - garbage collector, thing to clean useless rubbish but this function is dangerous to use

you can override method called “finalize”, which turns on, when garbage collector ends its work

**System.exit(0)** – end program

**Cloning**

If you want to clone this class you have to implement it from the **Cloneable** and override method:

**public class … implements Cloneable{  
 @Override  
 protected … clone() throws CloneNotSupportedException{ return (**name of class**) super.clone(); }**

**}**

**Create multivariable with many classes inside**

Create class (for example MyProxy), implement **InvocationHandler** and override **public Object invoke (Object proxy, Method method, Object[] args)** – cod there will happen before any classes methd

**InvocationHandler** handler **= new** MyProxy**();** - creating proxy

**Class[]** classes **= new Class[]{**Comparable**.class,** Callable**.class};** - creating array with classes **Object** proxy **= Proxy.*newProxyInstance*(null,** classes**,** handler**);** - creating variable, which includes those classes

splashScreen.update(); - update paint  
}

**Creating .jar programs**

-> Open console

-> go to the needed folder

-> compile main class by “ **javac … .java** “

->create jar file by“ **jar cvfe 1.jar 2 \*.class** “ – 1 – name of the new program, 2 – name of the main class

to open by using web browser : https://www.youtube.com/watch?v=o8zsGW36XJQ&index=148&list=PL786bPIlqEjRDXpAKYbzpdTaOYsWyjtCX

**Loggers**

public static final Logger logger = Logger.getGlobal(); - creating logger with name “global”  
public static final Logger logger2 = Logger.getLogger("personal\_name"); - creating logger with personal name

Handler fileHandler = new FileHandler(); - creating filehandler which will record as a file

Handler fileHandler2 = new ConsoleHandler(); - creating consolehandler which just show message at the console

fileHandler.setFilter(new MyFilter()); - creating filter, where you override method and sort what to add and what not  
logger.setUseParentHandlers(false); - stop showing at the console

logger.addHandler(fileHandler); - set filehandler or consolehandler  
logger.info("…"); - add or show info  
logger.log(Level…., "message"); - add or show message with personal level

class MyFilter implements Filter{  
 @Override  
 public boolean isLoggable(LogRecord record) {  
 return record.getMessage().endsWith("…");  
 }  
}

**Sorting strings**

List<String> list = new ArrayList<>(); - create array  
Collator collator = Collator.getInstance(); - create variable with rule  
collator.setStrength(Collator.TERTIARY); - set rule  
Collections.sort(list, collator); - sort

Adaptive to different countries

Locale locale = new Locale("ua", "UA"); - get variable with local settings of specific region  
System.out.println(Locale.getDefault()); - get your region  
System.out.println(NumberFormat.getCurrencyInstance(locale).format(100)); - format variable (the same with date)

Swap in different countries

(create lots of files with the same start name and format .properties

for example: res. Properties, res\_en. Properties, res\_ua. Properties)

in file values records like: val=1 val2=2 …

ResourceBundle bundledef = ResourceBundle.getBundle("resources", new Locale("","")); - get variable from default file (file without ending \_en ), files have to be .property   
ResourceBundle bundled1 = ResourceBundle.getBundle("resources", new Locale("en", "EN")); ("","")); - get variable from file (file with ending \_en ), files have to be .property   
  
System.out.println(bundledef.getString("val")); - get val

**Set security properties**

System.setProperty("java.security.policy", "src/MyApp.policy"); - set security properties from your own file  
System.setSecurityManager(new SecurityManager()); - set security manager  
System.getProperty("os.name"); - for example what you can allow not to do

grant{ - in file what you had to create  
 permission java.util.PropertyPermission "os.name", "read,write"; - allow to use this command in command line  
 permission java.net.SocketPermission "ip", "connect"; - allow to connect to this socket  
 permission java.io.FilePermission "/tmp/\*", "read,write"; - allow everything in this directory   
};

**MD5 and SHA-1**

String str = "…"; - get str  
MessageDigest shal = MessageDigest.getInstance("SHA-1"); - create hash format  
byte[] bytes = shal.digest(str.getBytes()); - get bytes from hashed str

**Subscribe your .jar file by hash**

[**https://www.youtube.com/watch?v=3EE-8nKaMCQ&index=224&list=PL786bPIlqEjRDXpAKYbzpdTaOYsWyjtCX**](https://www.youtube.com/watch?v=3EE-8nKaMCQ&index=224&list=PL786bPIlqEjRDXpAKYbzpdTaOYsWyjtCX)

**Crypt and decrypt messages**

**AES**

String str = "…"; - create str

Create key by 2 ways

KeyGenerator generator = KeyGenerator.getInstance("AES"); - create generator  
generator.init(128); - size of key  
SecretKey key = generator.generateKey(); - create key  
SecretKey key1 = new SecretKeySpec("13567890".getBytes(), "AES"); - create own key

encrypting

Cipher cipher = Cipher.getInstance("AES"); - create variable with cryptography   
cipher.init(Cipher.ENCRYPT\_MODE, key); - set mode and key  
byte[] bytes = cipher.doFinal(str.getBytes()); - get bytes of encrypted str  
  
cipher.init(Cipher.DECRYPT\_MODE, key); - set decrypting mode and key  
byte[] bytes1 = cipher.doFinal(bytes); - get encrypted bytes

**RSA**

The same as AES, but generate 2 keys (public and private) and encrypt using public? When decrypt is with private

KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");  
KeyPair keyPair = generator.generateKeyPair();  
Key publicKey = keyPair.getPublic();  
Key privateKey = keyPair.getPrivate();

***Insert JavaScript***

ScriptEngineManager manager = new ScriptEngineManager(); - create manager  
ScriptEngine engine = manager.getEngineByName("JavaScript"); - create variable with language  
  
engine.put("a" , 3); - set variable in “code”  
Object o = engine.eval(“code”); - code something and return object with “returned” variable from code  
Object o2 = ((Invocable)engine).invokeFunction("f","1"); - start function  
engine.get("a"); - get variable

**Link common intagers**

IntegerProperty a = new SimpleIntegerProperty(…); - create dynamic integer, variable of what consist of link, which you can share to another method, common integer won’t change if you change its copy in another method

a.setValue(…); - set

a.get(); - get

**Patterns**

Pattern pattern = Pattern.compile("...",Pattern.MULTILINE); - create expression (second value is for multiple or single line, if in single line you use logic with ending of line it won’t see \n, only end of stroke, multiple is inversion)  
Matcher matcher = pattern.matcher("..."); - get result from regular expression  
while (matcher.find()){ - get results  
 matcher…  
}

“…”.matches([\\b...\\b](file:///\\b...\\b)) or String a.matches([\\b...\\b](file:///\\b...\\b)) – regular pattern

“\\Q\* ... \\E+” - everything between those symbols is read as a common string

**(…)** – group, which has link (1,2,3,…)

**(?:…)** – group without link

**(?>…|…)** – atomic group (if it find first, it won’t go next) (example: a(?>bc|c)c abc- false abcc- true)

**“…”.replaceAll(“(…)…”, “$1 …”)** – replacing arguments with link to group (only there used $), **$0** – take all regular

In regular expressions if you want to use made group it looks like [\\1](file:///\\1) (for example if you look for html teg, it has to be ended with the same word <div>…</div>)

“(?...) … (?-…)…” – modifiers for example global, register sensitive…

\x{…} – direct chars

\u… - search by Unicode

**\p{**country**}** – symbols of these country

a**(?=**b**)** – look for a, what is coming before b, but don’t include b in results

a**(?!**b**)** – look for a, what isn’t coming before b

**(?<=**b**)**a– look for a, what is coming after b, but don’t include b in results

**(?<!**b**)**a– look for a, what isn’t coming after b

**Get back deleted variable**

Object o = new Object();

0 = null;

**SoftReference<**Object**> sr = new SoftReference<>(**o**)** – this will not be deleted after first use

**WeakReference<**Object**> wr = new WeakReference <>(**o**)** – this will be deleted after first garbage collecting

**PhantomReference<**Object**> pr = new PhantomReference <>(**o, wr**)**; - variable with function of checking the deletion

**Pr.remove()** – will stop thread until this variable won’t be deleted by garbage collector

**Types of classes methods**

Factory – creating common classes in one class

AbstractFactory – creating factories in one factory

FactoryMethod – create your own class

Singelton – create private elements and methods, which return link to them, you always change the same variables

Builder – hide constructors in integrated class, less code

Prototype – create variables and method, which return copy of those elements

ObjectPool – create 2 lists: first - free items and second - used items (you can always get access to the created item)

Adapter – adapting one class to another

Bridge – reduces levels of classes, lets you create less classes with integrated classes, what you left

Filter – gets in constructor array of classes and sort them before take back

Composite – tree structure, like folders

Decorator – let you to add your code to existing program

Façade – hide large code into another class to hide amount of code

FlyWeight – if example of class exists – don’t create another, use existing one

Proxy – add your class between another ( create class and implement or extend class, which is extended/implemented by usual class, than create example of usual class, override needed method, do what you want and ran this method from the created variable to continue cod as it was)

Chain Of Responsibility – actions happen one after another

Interpreter – change mathematical operations between classes by another class a + b == sum(a,b)

Momento - create class, which will record state and use it as a checkpoint

Observable – if you change one exemplar of class, it runs method connected with all (create static list of examples of this class and in constructor add new object to list, than you can create method on change and go throw all elements)

State – change behavior in way of changing states (create different classes and one combine them to inside one class, than in constructor set what class you want to use)

Null Object – your class don’t have to return null when you create it, return another class, which implements the same interface and it has to be empty

Strategy – don’t make lots of if conditions if you can move them to the another class and make core more readable

Template – create abstract class, which has abstract method and it runs them in constructor, than implement this class, override methods and they will start one after another

Front Controller – create class with switch check, where you start different classes

Low coupling – always use interfaces for classes for them not to know anything about each other

Stream

**(a , b , c, …) -> {…}** – lambda

List<String> list = new ArrayList<>();  
list.add("ft");  
list.add("sc");  
list.add("th");  
Stream<String> stream = list.stream(); - get stream

Stream<String> stream = list.stream().paralel(); - get parallel stream, break in threads and use it faster, but be careful  
list = stream.map(x -> x.toUpperCase()).collect(Collectors.toList()); - set all variables to upper case   
stream = list.stream(); - reload stream, because it closes after each using  
stream.forEach(x -> System.out.println(x)); - show all elements

Examples

Stream.of(1,2,3,0).collect(Collectors.partitioningBy(x -> x > 2)).forEach((a,b) -> {…}); - separate group in true and false by statement  
Stream.of(1,2,3,0).collect(Collectors.groupingBy(x -> x)).forEach((a,b) -> {System.out.println(…}); - group by statement  
Stream.of(1,2,3,1).collect(Collectors.groupingBy(x -> x, counting())).forEach((a, b) -> {System.out.println(…}); - group by statement and count

**Find deadlock**

ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();  
long[] dedloks = threadMXBean.findDeadlockedThreads();  
if (dedloks != null){  
 ThreadInfo[] threadInfos = threadMXBean.getThreadInfo(dedloks);  
 for (ThreadInfo info : threadInfos){  
 System.out.println(info);  
 }  
}

**Side effect** – in RX when you use variable, which can be changed in rx process by another method

**Сереализация** – recording objects

**Deadlock** – зависание программы, мол вызов одной функции, которая в ходе своей работы вызывает пред ведущую и замыкает цикл рекурсии

**127.0.0.1** – localhost

**Native** class – class, which use function from another language for example C++

JNDI – way to record something with key

RMI – create server from your machine and create there class, which will make add requests distantly

JAXB – connect xml with class

System.getenv().forEach((k, v) -> { - get all info about pc  
 System.out.println(k + ":" + v);   
});

Create variables in ram without garbage collector connection

https://www.youtube.com/watch?v=fFN\_wIKGjtk&index=361&list=PL786bPIlqEjRDXpAKYbzpdTaOYsWyjtCX