**Import …** - importing elements

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

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

Static methods **static{…}** happens only once, when you create object

**....equals(…)** – идентичность по значению, == - по положению в памяти если это ссылка

**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() {…}}**

**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)

**Catch has arguments:**

**NullPointerException** – when object is incorrectly used

**ArrayIndexOutOfBoundsException** – array exception

**ООП**

**Class … extends … {…}** – наследование

If parent’s constructor needs value you can enter it into the **super(…)**, with is placed in the start of the constructor of the extended class

You can assign extended class to the parent one

**Class …{ {…} – block …(){…} – constructor }** – block apply its code before the constructor

**FILE**

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

**File … = new File(“directory or file”)** – creating file or directory

**… .isDirectory()** – checking if it is directory

**… .isFile()** – checking if it is file

**… .exists()** – checking if it exists

**… .mkdir()** – create directory

**… .createNewFile()** – creating new file in your directory

**FileWriter** filewrite **= new FileWriter(**file**);** - creating variable to write into the filefilewrite**.write("**parasha1 \n **");** - writing textfilewrite**.flush();** - pushing new text into the filefilewrite**.close();** - closing file **FileReader** fileread **= new FileReader(**file**);** - creating variable to read file **char[]** ch **= new char[…];** fileread**.read(ch);** - copying message from file to the variable  **BufferedWriter** bufwr **= new BufferedWriter(**filewrite**); -** creating clever variable to write into the filebufwr**.write("**jopka**");** - writing into the variablebufwr**.newLine();** - make new linebufwr**.flush();** - push the variable into the filebufwr**.close();** - close file **BufferedReader** bufrd **= new BufferedReader(**fileread**);** - creating clever reader  **while (**bufrd**.ready()){ System.*out*.println(**bufrd**.readLine());**  } – if file is ready read a line

**Serializing (recording objects)**

You have to add **implements Serializable** to the class if you want to be able to save it to the file

**public class … implements Serializable { … }**

Creating object *car* with parameters **FileOutputStream** *fileOutputStream* **= new FileOutputStream("**first/temp**");** -getting location **ObjectOutputStream** *objectOutputStream* **= new ObjectOutputStream(***fileOutputStream***);** - taking location to the variable of class, which can record objects

*objectOutputStream***.writeObject(***car***);** - recording object*objectOutputStream***.close();** - closing file **FileInputStream** *fileInputStream* **= new FileInputStream("**first/temp**");** - getting directory to read **ObjectInputStream** *objectInputStream* **= new ObjectInputStream(***fileInputStream***);** - taking location to the variable of class, which can read objects

*creating object … to record similar object from the file* **= (***object***)***objectInputStream***.readObject();***objectInputStream***.close();**

**DATE**

**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**

**List/LinkedList**

**Queue**

**Map/HashMap/LinkedHashMap/TreeMap**

**HashSet** – no repeated elements

Has methods:

**.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

**ARRAYS**

**Int arr = new int […]** – creating array

**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)

**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 (…) super.clone(); }**

**}**

**THREADS**

Create and extend class **Thread**, then override method **run()**, where all functions start

Create new exemplar of this class and run method **start();** to create thread

**Thread.sleap(…)** – static function to stop current thread (put in try/catch with **InterraptedExeption**)

**Thread.*yield*();** - throw current thread from the running to the runnable

**.join()** – connect this thread to the queue, that’s mean ask him to wait

**Synchronized(this** or variable**){…}** – change current function

**.wait()** – ask to stop thread in this variable (connect with function on the top)

**.notify()** – start all threads in this synchronized line (connect with function on the top)

**Synchronized** – add it before name of the function to ask the function to finish before let another thread make it

**Volatile** – ask threads to write variable into the current cash, not to create personal

There are special variables for threads begins with **Atomic…** for example **AtomicInteger** … = new **AtomicInteger(…)** – to create value and it has lots of methods for threads, without any bugs (synchronized)

**Lock** lock **= new ReentrantLock();** - create lock variable

lock**.lock();** - lock thread

lock**.unlock();** - unlock thread

lock**.tryLock()** – checks if it is possible to lock

**Condition** *condition***=** *lock***.newCondition();** - create condition variable

*condition***.await();** - pause this thread

*condition***.signal();** - continue paused thread

**ExecutorService** executorService **= Executors.*newFixedThreadPool*(…);** - create variable, which can control threads better then common caseexecutorService**.submit(…);** - start thread insede the classexecutorService**.shutdown();** - end threads after all functionsexecutorService**.shutdownNow()** – end threads now

**Synchronized(this** or variable**){…}** – change current function

**Сереализация – recording objects**

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