**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();**

**mostly used because of easy to record**

this class records objects in the file, which is very easy to change, also you can easily read them

**Properties** def **= new Properties();** creating default (works like main) **def.setProperty("**key**","**value**");** - set value with key **Properties** properties **= new Properties(**def**);** - create variable with default values (works like map)properties**.setProperty("**key**","**value**");  
FileOutputStream** outputStream **= new FileOutputStream("**…**");** - new variable, where you can record object, it will record with name … **properties.store(**outputStream**,"comments");** - record object **FileInputStream** inputStream **= new FileInputStream("**…**");** - read file …properties**.load(**inputStream**);** - load file into properties properties**.getProperty("**key**");** - get value

**Preferences**

**Preferences** preferences **= Preferences.userRoot();** - creating variable connected with user folder in disk C **Preferences** preferences **= Preferences.systemRoot();** - creating variable connected with system folder in disk C **Preferences** node **=** preferences**.node("**name**");** - creating new folder in preference preferences**.put("**key**", "**value**");** - setting key and valuepreferences**.get("**key**","**default**");** - getting value by key or getting default  **Preferences** mypackage **= Preferences.userNodeForPackage(**car**.class);** - creating package preferences**.exportSubtree(new FileOutputStream("… .xml"));** - recording filepreferences**.importPreferences(new FileInputStream("… .xml"));** - getting file

**Reader**

**Reader** reader **= new FileReader("**where**");**

**Int** i **=** reader**.read()** – read file by char numbers, by the end of the stroke returns **-1**

BufferedReader

**Reader** reader **= new BufferedReader(new FileReader("**where**"));** - create reader, which works much faster

reader**.read(…)** – reads 1 line and asks char array variable inside, where will be recorded line

**writer**

**FileWriter** writer = **new FileWriter("**where**")**  - slow filewriter

**BufferedWriter** writer **= new BufferedWriter(new FileWriter("**where**"));** - faster than filewriter with additional functionswriter**.write("…\n");** - write textwriter**.newLine();** - new line ( available only in BufferedWriter **)**writer**.flush();** - record filewriter**.close();** - close and record file

**InputStream**

**BufferedInputStream** inputStream **= new BufferedInputStream(new FileInputStream("**where**"));** - faster than inputStream

**InputStream** inputStream **= new FileInputStream("**where**"));** - inputstream

**byte[]** i **= new byte[…];** - where to collectinputStream**.read(**i**)** – read, by the end of the stroke returns **-1**

**OutputStream**

**BufferedOutputStream** outputStream **= new BufferedOutputStream(new FileOutputStream("**where**"));** - faster than outputStream

**OutputStream** outputStream **=** **new FileOutputStream("**where**"));** - outputstream

outputStream**.write(1);** - write or insert arr of **bytes**outputStream**.flush();** - saveoutputStream**.close();** - close and save

**Path and Files**

**Path** path **= Paths.get("text.txt").toAbsolutePath()** - path of the file or directory, this variable has lots of usable methods, which returns part of path or all path in correct form **Files.copy(Paths.get("**from where **"), Paths.get("**where**"), StandardCopyOption.REPLACE\_EXISTING);** - copy file

(3 – variable is to tell how to do this ) **Files.deleteIfExists(Paths.get("**where**"));** - delete file if it exists, or only **delete(…)  
Files.move(Paths.get("**from where**"), Paths.get("**where**"));** - move files

**Files.size(**path**);** - returns size **Files.isDirectory(**path**);** - returns if it is directory **Files.isHidden(**path**);** - if it is hidden **Files.isReadable(**path**);** - if it is readable **Files.isWritable(**path**);** - if it is writable

**byte** b**[] = Files.readAllBytes(**path**);** - read from file

**Files.write(**path**, "**…**".getBytes());** - write text in file, by transforming str into bytes **List<String>** list **= new ArrayList<String>();  
Files.write(**path**,** list**);** - you can also record lists **InputStream** inputStream **= Files.newInputStream(**path**);  
OutputStream** outputStream **= Files.newOutputStream(**path**);  
Reader** reader **= Files.newBufferedReader(**path**);  
Writer** writer **= Files.newBufferedWriter(**path**);  
  
DirectoryStream<Path>** entries **= Files.newDirectoryStream(Paths.get("."));** - get all files from current directory (**Path** array )

**Scanner writer** the fastest way to write in file

**Scanner** scanner **= new Scanner(new FileReader("**where**"));** - create scannerscanner**.hasNext()** – if it is nextscanner**.next()** – get next **PrintWriter** writer **= new PrintWriter(new FileWriter("**where**"));** - create writerwriter**.write("**…**");** - writewriter**.flush();** - savewriter**.close();** - save and close