CLASSES

Constructors

- Used for creating objects from a class
- Can be either **user-defined** or **default**

Default	User-defined
Assigns default values to instance variables.	Allows parameters to be passed.

`public class example {`
`  private String name;`
`  public example() {name = "Vladimir";}`
`  // name is Vladimir when object is created`
`}`
	`public class example {`
	`  private String name;`
	`  public example(String n) {name = n;}`
	`  // name of object is decided by the user`
	`}`

Access modifiers

Modifiers define how accessible a method or variable is.
public: can be accessed by other classes
private: cannot be accessed by other classes

Encapsulation

- An OOP concept that involves hiding instance variables from other classes by initialising them with **private** access modifiers. (Example: ***private int age;***)
- These variables can still be accessed through **getter** and **setter** methods

Methods

- **Call by value**: method called by passing **value** of variable as a parameter
- **Call by reference**: method called by passing memory **address** of variable

Code	Explained
`public static String example(String abc)` `{ // your code }`	Declaration of a method in the order: `public static returnType methodName(arguments){}`
`public static double[] example(double... abc)` `{ // abc is now an array of double values }`	Method with variable length arguments, meaning you can pass multiple parameters of the same type and they would auto-fill an array of that type.
`try { // code with IO errors }` `catch (IOException e) {e.printStackTrace();}`	Exceptions are errors that occur in a method. They can be caught using a `try/catch` block.

File IO

Character streams:
- They are used for 16-bit output **unicode**
- Classes for this are **FileReader** and **FileWriter** (in java.io)

Byte streams:
- They are used for 8-bit output **bytes**
- Classes for this are **FileInputStream** and **FileOutputStream** (in java.io)

Both **byte** and **character** streams can only be read in a forward direction.
The classes above can only do one task – either **read** or **write**, not both.

However, if you use java.io.**RandomAccessFile** then you can read from and write to files at any position. There are 4 access modes, the 2 main ones are:
r (read) and **rw** (read + write)

ArrayList

Action	Command
Declaration	`ArrayList<Wrapper> example = new ArrayList<Wrapper>();`
Add item	`example.add(index, "Henry");`
Remove item	`example.remove("Henry");`
Set item value	`example.set(index, "Vlad");`
Get item value	`example.get(index);`
Size of list	`example.size();`

Primitive Type	Wrapper Class (`ArrayList<Wrapper>`)
> `byte`, `boolean`, `double`, `float`, `long`, `short`	First character becomes uppercase. (e.g. double ——> Double)
> `int`, `char`	Integer, Character

Advantages of `ArrayList` over a normal array:

- Dynamic size
- Performance
- Multi-dimensional

GENERAL

Normal array

Action	Command
Declaration	`type[] example = new type[size];`
...with items	`... = {1, 2, 3};`
2D (or Nested)	`type[][] example = new type[rows][columns];`
...with items	`... = {{1, 2, 3}, {4, 5, 6}};`
Set item value	`example[index] = 25;`
Get item value	`example[index];`
Size of list	`example.length;`

Common loops/operations

```
While             while (condition){...}
Do While          do {...} while (condition);
For               for (iterator, condition, iteration){...}
For enhanced      for (type iterator: exampleArray){...}
Ternary           (condition) ? ( // if true code ) : ( // if false code );
Switch            switch (example)
                  {
                   case cond: ...;break; // if example == conditional value cond
                   case cond: ...;break;
                   default: ...;break; // if example != any of the values cond
                  }
```

NOTE: for `String` methods, refer to `StaticMethods.java`

Random numbers

Generate between **min** and **max**:

```
int rand = (int)(Math.random() * (max - min + 1)) + min;
```

For example, from 0 to 99:

```
int rand = (int)(Math.random() * 100);
```