# World of JAVA – High-level OOPL – by Vuk

**Extreme Programming (XP)**
The thrust of the XP is that the customer gets what he wants, when he wants it, even when the spec changes late in the game. XP is based on a set of proven practices that are all designed to work together, although many folks do pick and choose, and adopt only a portion of XP's rules. These practices include things like: **Make small, but frequent releases**, **Develop in iteration cycles**, **Don't put in anything that's not in the spec**, **Write the test code first**, **Work regular hours**, **Refactor whenever and wherever you notice the opportunity**, **Don't release anything until it passes all the tests**, **Set realistic schedules, based around small releases**, **Keep it simple** and **Program in pairs, move people around so that everybody knows pretty much everything about the code.**

**What is a file?**
A file is a named location which stores data or information permanently. A file is always stored inside a storage device using file name. A file name normally has primary and secondary name separated by a dot(.).

**What is a database, a record and a key?**
A database is a unit of data storage composed of many similar records. A record often represents a real-world object, such as an employee or a car part and a Key is a field in a record that's used to carry out some operation on the data.

**What is a data structure and what it allows?**
A data structure is the organization of data in a computer's memory or in a disk file and it allows major improvements in program efficiency if correct choice of data structure is chosen.

**What are examples of data structures?**
Examples of data structures are arrays, stacks and linked lists.

**Does java support multiple inheritance?**
No Java does not. We can simulate multiple inheritance by implementing more than one interfaces. Java class can extend only one class.

**What is a class?**
A class is a blueprint from which objects are created. A class contains methods and variables associated with an instance of a class. Class is made of things it knows – instance variables (state) and things it does – methods (behavior).

**What is a singleton class?**
Singleton class limits the number of objects created for a class to one but gives the flexibility of creating more objects if the situation changes.

**Is string class final and what is main class used for?**
Yes. Main class is used for starting a Java app or testing one.

**What are usually practices when developing a Class?**
As a programmer You should have a methodology or an approach when it comes to writing code. When creating Java class as a "learning experience" usually the steps are: figuring out what the class is supposed to do, listing the instance variables and methods, writing prep code for the methods, writing test code for the methods, implementing class, testing the methods and finally debug and reimplement as needed.

**What is prep code, test code and real code?**
Prep code presents a form of pseudocode, to help focus on the logic without stressing about syntax. Test code presents a class or methods that will test the real code and validate that it's doing the right thing. Real code is the actual implementation of the class.

**What is the difference between objects and data structures?**

Objects hide their data behind abstractions and expose functions that operate on that data. Data structures expose their data and have no meaningful functions. They are virtual opposites.

**What is an object?**
An object is an instance of a class. All objects live on the Heap.

**What are steps for creating an object?**
Three steps of object declaration and assignment: declare a reference variable, create an object, and assign the object to the reference.

**What is a constructor?**
A constructor presents methods which are used to create an object of class. There are two types of constructors: default and parameterized. Constructor is created the second we write "new". Compiler also automatically assigns a constructor to a class where one has the same name. Constructor is instantiated before an object gets its reference.

**What can we declare in methods?**
In methods we can declare statements, loops and branches. Every method has a return type and returns values, but void is the only method declaration that does not return any value.

**Where method calls occur?**
Method calls are always between two objects on the same heap. A design for remote method calls consists of four things: server, client, server helper and client helper. The 'helpers' are the objects that actually do the communicating. They make it possible for the client to act as though its calling a method on a local object. The client calls a method on the client helper, as if the client helper were the actual service. The client helper is actually a proxy, but not the remote service although he acts like it. The client helper doesn't have any of the actual method logic the client is expecting. Instead the client helper

contacts the server, transfers information about the method call and waits return from the server.

**Compare method and a constructor?**
Methods are used to perform some functions/or operations unlike constructors which are used to initialize instance of the class. Also, Methods do have a return type, but constructors do not have a return type.

**Which is the highest operator precedence in java?**
The operator with the highest preference is a postfix operator ( (),[]).

**What is the difference between overloading and overriding?**
Overloading considers when two or more methods in the same class have the same method name but different argument lists.
Overriding is when two methods have the same method name and parameters, but ones are allocated in parent and child class.

**What are different OOP principles?**
Basic OOP principles are: Encapsulation, Abstraction, Inheritance and Polymorphism.

**What does Encapsulation consider?**
Encapsulation is the process or a mechanism where code is combined in a single unit by using classes and objects. Encapsulation provides us with layer of security with data we are manipulating, protecting ones from external misuse. Regularly we define instance variables private and Getters and Setters public.

**What is an abstract class?**
A class that contains the abstract keyword in its declaration is called an abstract class. The properties of an abstract class are: Abstract classes may or may not contain abstract methods, but if a class has at least one abstract method it's automatically declared as abstract class, an abstract class cannot be instantiated, To use an abstract class, we have to inherit it from another class and if we inherit it then we have to provide implementations for all abstracts methods in its parent. Implementing an abstract method is just like overriding a method.

**What is an interface?**
An interface is a reference type in Java. Similar to class but it is a collection of abstract methods. An abstract method has no body. A class can implement multiple interfaces. Interface is like a 100% pure abstract class.

**What is the difference between a class and an interface?**
An interface cannot be instantiated, and it doesn't have any constructors. Interface has only abstract methods and can extend multiple interfaces. A class implements an interface and extends a class.

**What is inheritance?**
Inheritance is a property in which the property of a parent class(superclass) is passed onto a child class(subclass). Using inheritance, we define a protocol for a group of classes.

**What is polymorphism?**
Polymorphism is the ability of an object to take on multiple forms. Most commonly polymorphism is used in OOP when a parent class reference is used to a child class object.

**What are different types of access modifiers?**
There are four types of access modifiers: visible to overall package - no modifier needed, private – visible to class only, public – visible to the world and protected – visible to the package and subclasses.

**What is an instance and what are class variables?**
An instance variable belongs to a particular instance of that class. Whereas a class variable is also known as static variable. Local variables and method invocations live on the Stack.

**What is the difference between local variables and instance variables?**
Instance variables are defined inside the class, but not method. Instance variables live in object in which they belong, whereas local variables live inside the method they're declared. They are temporary and exist only while method is on the stack until one pops off.

**What is a Wrapper class?**
To access the primitive data type as an object we use wrapper class. They start with a capital letter for every primitive type except for "character" and "Integer". Wrapping an array in a class protects the array from being inadvertently altered.

**Explain Exceptions?**
When using "risky" methods, the practice is to wrap the "risky" code in something called try/catch. What You write in a catch block depends on the exception that was thrown. That's when polymorphism comes to play, You can extend particular Exception types from Superclass Exception. The compiler checks for everything except Runtime Exceptions.

**What does compiler guarantee when it comes to exceptions?**
The compiler guarantees that if you throw an exception in your code you must declare it using the throws keyword in your method declaration. Also, it guarantees if you call a method that throws an exception, you must acknowledge that you're aware of the exception possibility. Either you Handle an exception or declare one.

**What's an inner class and what it can do?**
An inner class can use all the methods and variable of the outer class, even the private ones. The inner class gets to use those variables and methods just as if the methods and variables were declared within the inner class. An inner class instance must be tied to an outer class instance. (except when an inner class is defined within a static method).

**Explain saving files briefly?**
If your data will be used by only the Java program that generated it, then use serialization – write a file that holds serialized objects. Then have your program read them. If your data will be used by other programs just write a plan text file, with delimiters that other programs can parse. You can also save data in any format you choose. But regardless of the method you use, fundamental I/O techniques are pretty much the same: write some data to something, and usually that something is a file on disk or a stream coming from a network connection. Reading the data is the same process in reverse.

**Explain what happens when object is serialized?**
When an object is serialized, all the object it refers to from instance variables are also serialized. And all the objects those objects refer to are serialized and so on. It happens automatically. Serialization saves the entire object graph. All objects referenced by instance variables, starting with the object being serialized. If you want your class to be serializable, implement Serializable. Either the entire object graph is serialized correctly, or serialization fails. If you mark an instance variable as *transient* it won't be saved and will be brought back to null once deserialized.

**Explain changes that can or can't hurt deserialization?**
Version control is crucial. Changes to a class that can hurt deserialization include: deleting an instance variable, changing the declared type of an instance variable, changing a non-transient instance variable to transient, moving a class up or down the instance hierarchy, changing a class from serializable to not serializable or changing an instance variable to static. Changes that are usually ok are: adding new instance variable to the class, adding, or removing classes to the inheritance tree, changing the access level of an instance variable, or changing an instance variable from transient to non-transient.

**Explain streams?**
Streams are either connection stream or chain streams. Connection streams can represent a connection to a source or destination, typically a file, network socket connection or the console. Chain streams cannot connect to a source or destination and must be chained to a connection (or other) stream.

**What is a socket?**
To connect to another machine, we need a socket connection. A socket is an object that represents a network connection between two machines. A connection is a relationship between two machines where two pieces of software know about each other. In other words, they communicate to each other and know how to send bits likewise. In order to make a socket connection we need to know two things about the server: IP address and TCP port number.

**How communication goes over a socket?**
To communicate over socket connection, you use streams. One of Java features is that most of your I/O work won't care what your high-level chain stream is actually connected to. In other words, you can use a BufferedReader. The steps for reading are following: Make a socket connection to the server, make an InputStreamReader chained to socket's low-level (connection) input stream and then make a BufferedReader and read. For writing: make socket connection to the server, make a PrintWriter chained to socket's low-level (connection) output stream and write (print) something.

**What is RMI?**
RMI(Remote method invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. RMI uses stub and skeleton object for communication with the remote object.

**What's a stub and what's a skeleton?**
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. The skeleton is an object, acts as a gateway for the server-side object. All the incoming requests are routed through it.

**What are Servlets?**
Servlets are Java programs that run entirely on (and with) an HTTP web server. When a client uses a web browser to interact with a web page, a request is sent back to the web server. If the request needs the help of a Java servlet, the web server runs (or calls, if the servlet is already running) the servlet code. Servlet code is simply code that runs on the server, to do work as a result of whatever the client requests. Even Servlets can use RMI. To compile a servlet, you need the servlet packages which are in the servlets.jar file. The servlet classes are not part of the Java standard libraries, so you need to download the servlets.jar from a servlet-capable web server with a servlet "container".

**Explain JINI briefly?**
Jini uses RMI, but gives you a few key features like **adaptive discovery and self-healing networks.** With JINI, the client has to know only one thing: the interface implemented by the service. The trick revolves around JINI lookup services which are far more powerful and flexible than the RMI registry. JINI services announce themselves to the network automatically. When a lookup service comes online, it sends a message (using IP multicast) out to the network that one is present. The main difference between application using RMI and JINI application is how the service is discovered.

**Explain threads in java?**
A thread is a separate 'thread of execution'. In other words, a separate call stack. A Thread is a Java class that represents a thread. To make a thread, make a Thread. A Thread object represents a thread of execution. You'll create an instance of class Thread each time you want to start up a new thread of execution. A lower-case t (thread) is a separate thread of execution. That means a separate call stack. A capital T (Thread) is a class that represents a

thread of execution. It has methods for starting a thread, joining one thread with another, and putting a thread to sleep.

**How to launch a new thread?**
Make a Runnable object (the thread's job), make a Thread object(the worker) and give it a Runnable (the job) and finally start the Thread. Runnable is an interface (it's what's a job to a worker – is the job a thread is supposed to run), then you pass the new Runnable to the Thread constructor. This tells the new Thread object which method to put on the bottom of the new stack – the Runnable's run() method. When the new thread starts up, it takes the Runnable object's run () method and puts it on the bottom of the new thread's stack.

**Explain briefly TreeSet?**
TreeSet can't read you mind to figure out how the object's should be sorted. You have to tell the TreeSet how. To use a TreeSet, one of these things must be true: The elements in the list must be of a type that implements Comparable or You use the TreeSet's overloaded constructor that takes a Comparator.

**Explain Collections API and three main interfaces?**
Three main interfaces from the Collection API are List, set and Map. **List** is used when we need to know about index position. Lists know where something is in the list. You can have more than one element referencing the same object. **Set** is used when we don't need duplicates. Sets know whether something is already in the collection. You can never have more than one element referencing the same object. **Maps** are used when finding something by key matters. Maps are collections that use key-value pairs. Maps know the value associated with a given key. You can have two keys that reference the same value, but you cannot have duplicate keys. A key can be any object.

**What do ordered and unordered arrays offer?**
Ordered arrays can be accessible with a binary search whilst unordered arrays offer fast insertion but slow searching and deletion.

**What is the difference between equals () method and == operator?**
The equals() is a method and it matches the content of the strings whereas == is an operator and matches object or reference of the strings. Also, we compare two primitive valuables with == and two objects with .equals() method.

**Explain Java Object Law for HashCode() and equals()?**
The API docs for class Object state the rules you must follow: If two objects are equal, they must hav matching hashcodes, if two objects are equal, calling equals() on either must return true, if two objects have the same hashcode value, they are not required to be equal. But if they're equal, they must have the same hashcode value, so if you override equals(), you must override hashcode(), the default behaviour of hashcode() is to generate a unique integer for each object on the heap, so if you don't override hashcode() in a class, no two objects of that type can ever be considered equal, the default behavior of equals*( is to do an == comparison, a.equals(b) must also mean a.hashcode() == b.hashcode() but that doesn't have to mean a.equals(b).

**What is a JAR?**
A JAR file is a Java Archive. It's based on the pkzip file format, and it lets you bundle all your classes in a single JAR file. JAR files should be made executable so that client run them. An executable JAR means the end-user doesn't have to pull the class files out before running the program. The user can run the app while the class files are still in the JAR. The trick is to create a manifest file, that goes in the HAR and holds information about the files in the JAR. To make a JAR executable, the manifest must tell the JVM which class has the main() method.

**What is JWS (Java Web Start) and what is .jnlp file?**
Java Web Start is, among other things, a small Java program that lives on the client machine and works much like a browser plug-in. JWS has the ability to detect when even a small part of application has changed on the server, and – without any end-user intervention-download and integrate the updated code. (Java Network Launch Protocol) file describes my application> This is the file the JWS app reads and uses to find your JAR and launch the app. A .jnlp file is a simple XML document that has several different things you can put in. Briefly, JWS consists of executable JAR```` and a .jnlp file.

**What is a stream?**
A stream can be defined as a sequence of data. There are two types of streams: InputStream – used to read data from a source and OutputStream – used to write data into a destination.

**What is a byte stream?**
Java Byte Stream is basically used to perform input and output for 8 bit Unicode. The main classes related to byte stream are FileInputStream and FileOutputStream.

**What is a character stream?**
Java Character Stream is basically used to perform input and output for 16 bit Unicode. The main class users are FileReader and FileWritter which internally uses FileInputStream and FileOutputStream so the basic difference is that FileReader and FileWritter read and writes two bites at a time effectively.

**What is a Big-O notation and how does run speed differ?**
Big O notation provides a convenient way to compare the speed of algorithms. An algorithm that runs in O(1) time is the best, O(log N) is good, O(N) is fair, and O($N_2$) is the worst.

**What data structures are usually used to simplify certain programming operations?**
Stacks, queues, and priority queues. In these data

structures only one data item can be accessed. Stacks allow LIFO and queues allow FIFO access to data.

**Name important stack operations and queue operations?**
The important stack operations are pushing(inserting) an item onto the top of the stack and popping(removing) the item that's on the top. The important queue operations are inserting an item at the rear of the queue and removing the item from the front of the queue.

**What does priority queue allow and what are important priority queue's operations?**
A priority queues allows access to the smallest (or sometimes the``` largest) item. The important priority operations are inserting an item in sorted order and removing the item with the smallest key.

**Explain linked lists?**
A linked list consist of one LinkedList object and a number of link objects. The LinkedList object contains a reference, often called first, to the first link in the list. A next value of null signals the end of the list.

**What is Git?**
Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot to be efficient, if files haven't changed. Git things about its data more like stream of snapshots. This is an important distinction between Git and nearly all other VCSs. Nearly every operation is local with Git.

**What is a VCS?**
With VCS (Version Control System) we can track our history and work together. There are centralized and distributed VCS-s. The problem with centralized architecture VCS is the "single point of failure". If the server goes offline, we can't collaborate or save our snapshots of the project. On the contrary, the distributed systems every collaborator has saved version of the project on their local machine. Git is the example of a distributed architecture VCS. Git benefits are that one is free, open source, fast and scalable.

**What are the three states of Git?**
Git has three main states that your files can reside in: modified, staged and committed. Modified means that you have changed the file but have not committed it to your database yet. Staged means that you have marked a modified file in its current version to go into your next commit snapshot and committed means that the data is safely stored in your local database.

**What are the three main sections of a Git project?**
Three main sections of a Git projects are: the working tree, the staging area, and the Git directory.

**Explain The Working tree?**
The working tree is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

**What is a branch?**
Branch is a copy of the repository at a certain point in time which contains different changes.

**Explain the staging area?**
The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git parlance is the "index", but the phrase "staging area" works just as well.

**Explain Git directory?**
The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

**What kind of files can be in your working directory in Git?**
Each file in your working directory can be in one of two states: tracked or untracked. Tracked files are files that were in the last snapshot, as well as any newly staged files. They can be unmodified, modified or staged. In short, tracked files are files that Git knows about. Untracked files are everything else – any files in your working directory that were not in your last snapshot and are not in your staging area. When you first clone a repository, all your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

**Git tags – Annotated and lightweight?**
A lightweight tag is very much like a branch that doesn't change - it's just a pointer to a specific commit. Annotated tags, however, are stored as full objects in the Git database. It's recommended to create annotated tags because of all the information they hold, but if some temporary tag is needed lightweight tags can get the job done. By default, git push doesn't transfer tags to remote servers, as ones have to explicitly be pushed to a shared server after creation.

**What is Maven?**
Maven is a build tool that helps in project management. The tool helps in building and documenting the project. Maven is written in Java and c# and is based on the Project Object Model (POM). The tool is used to build and manage any Java-based project. It simplifies the day to day work of Java Developers and helps them in their projects. Maven helps to create the right project structure which is essential for execution.

**What does Maven do?**
Maven simplifies the process of building the project. The task of downloading Jar files and other dependencies does not need to be done manually. Maven also provides easy access to all the required information. Apache Maven helps manage all the processes, such as building, documentation, releasing, and distribution in project management.

**Explain briefly Maven UIDs?**

Maven stands for “Accumulator of Knowledge”. Core configuration file is pom.xml. It’s a central repository for dependencies as well as an universal build system in Java. When we define our project, we give it **groupID, artifactID, and version.** These are **unique identifiers** of our unit that we’re building. Ones can be referred to **dependencies.** GroupID is an identifier of the organization that is creating application or artifact (jar file, etc.). ArtifactID is going to play into the name of the final build artifact that are we creating with script (name of final compilation unit). Version considers version of created artifact. Snapshot indicates work in progress. Packaging can be .war, .jar, .ear, .etc. and description considers documentation.

**What is Project Object model?**

POM is an XML file that has all the information regarding project and configuration details. When we tend to execute a task, Maven searches for the POM in the current directory.

**What is the build lifecycle of Maven?**

Build lifecycle defines what are the different steps Maven would execute during its lifecycle. Build lifecycle considers: validate, compile, test, package, integration test, verify, install and deploy. We can add plugins that can do any of the steps.

**What is spring?**

Spring is a complete and a modular framework for developing enterprise applications in java. Spring can be used for the development of a **particular layer** of a real time application. Spring framework can be used for **all layer implementations** for real time applications.

**What does Spring framework contain?**

The Spring framework contains a lot of features, which are well-organized in about twenty modules. These modules can be grouped together based on their primary features into following: **Core container, Data Access/Integration, Web, AOP, Instrumentation and Test**.

**What is SQL?**

SQL (Structured Query Language) is a language we use to communicate to databases. Database is a collection of data. It is a method for accessing and manipulating data. Database is a structured set of computerized data with accessible databases. DMS (Database Management System) allows us to communicate with databases. Some of DMS-s are PostgreSQL, MySQL, Oracle Database, SQLite. We are giving commands to DMS which gives commands to the database to extract data.