    - answer
- Core Java and OOP(Infosys questions)
    - What is DAO?
    - Difference between abstract classes and interfaces?
    - What is Encapsulation and Abstraction?
    - Analytical question: Difference between method and function..
    - How to handle class not found exception
    - Describe one thin that introduced in the java 8?
    - Tell me about functional interface?
    - Do you know Method References?
    - Difference between lambda expression and method reference?
    - Tell me about Stream API?
    - When is each method of software development apply?
    - What is java 8 features?
    - Describe Abstract in OOP
    - what is Java streaming?
    - core java: polymorphism, abstract classes and interfaces(difference), AbstractionAws, aws
    - storage class
    - streams
    - Core java: what is JVM ?
    - What is polymorphism?
    - Encapsulation and access modifier,the types of access modifier,Which one is more protective?
    - How many constructors you used in java?
    - Can you do static overriding?
    - How you connected your db in java?
    - Analytical question: Difference between method and function..
    - How to handle class not found
    - exception
    - Java 8 interfaces and features
    - MapException handling
    - Difference between string buffer and String Builder
    - What is java 8 features?
    - Describe Abstract in OOP
    - what is Java streaming?
    - difrenerence between Throw and Throws?
    - what is pojo design pattern?

Java Interview Prep

Coding Interview Questions

- https://www.hackerrank.com/dashboard
- https://www.youtube.com/watch?v=h36mQC3JFMo&list=PLqq-6Pq4lTTZgXnsBNQwCWdKR6O6Cgk1Z&ab_channel=JavaBrains

Conceptial Questions

1. What is Java?

Motivation

- Familiarity with Java
- Comparison with other languages (C, C++, C#, Python, Ruby, JavaScript, etc.)

Answer

- Object-oriented programming language
- Portable - write once, run anywhere
- Very popular today in server-side programming
- Created by Sun Microsystems, released in 1995, and is now widely used in enterprise applications

2. What are some freatures of Java?

(Why java is platform independent?)

Motivation

- Awareness of what the Java has to offer
- Comparison with other languages (C, C++, C#, Python, Ruby, JavaScript, etc.)
- Importance language characteristics

Answer

- Simple, object-oriented, and familiar syntax to c and c++ programmers
- Robust and secure(virtual machine)

- Architecture-neutral and ***platform-independent***(write once and run anywhere, jvm - another abstraction layer)
- High-performance(JIT compiler)
- Interpreted, threaded and dynamic(compiled, bytecode is interpreted at the runtime, typing is statically checked, behavior is dynamic)

3. What is JVM?

Motivation

- Diffrence bwtween JVM and JRE
- Characteristics of JVM
- Role it plays in execution

Answer

- JVM - Java virtual machine
- It's the runtime environment for Java programs, a extra layer of abstraction over the hardware
- Takes the compiled bytecode and executes it
- input to the JVM is the Bycode, output is the machine code
- Has a specification that outlines how it should work
- Diffrent implementation available
- Essentialy in making Java platform agnostic(write once and run anywhere)

List of JVM

- Codename One – uses the open source ParparVM
- Eclipse OpenJ9 – open-source from IBM J9, for Windows, AIX, Linux (x86, Power, and Z), macOS, MVS, OS/400, Pocket PC, z/OS.
- GraalVM – is based on HotSpot/OpenJDK, it has a polyglot feature, to transparently mix and match supported languages.
- HotSpot – the open-source Java VM implementation by Oracle.
  Jikes RVM (Jikes Research Virtual Machine) – research project. PPC and IA-32. Supports Apache Harmony and GNU Classpath libraries. Eclipse Public License.
- leJOS – Robotics suite, a firmware replacement for Lego Mindstorms programmable bricks, provides a Java programming environment for the Lego Mindstorms RCX and NXT robots.
- Maxine – meta-circular open source research VM from Oracle Labs and the University of Manchester.

4. What is JRE?

Motivation

- Diffrence bwtween JVM and JRE
- Characteristics of JRE
- Role it plays in execution

Answer

- JRE - Java run time environment
- It is a set of software elements that together provide the environment in which Java programs are executed
- Constists of
    - Class loader
    - JVM
    - Libraries and utilities
- The JRE orchestrates the activities of these software elements
- Installed on machines that run Java

6. What is JDK?

Motivation

- Diffrence bwtween JDK and JRE
- Characteristics of JDK
- Role it plays in the development time(Author time)

Answer

- JDK - Java development kit
- It is a set of tools to help developers write Java programs
- Comes with a JRE(becuase you need a JRE to run the JDK)
- Based off the Java language specification
- Includes
    - Java compiler
    - Class libraries
    - Utilities

7. What is the difference between JVM and JRE and JDK?

Motivation

- Characteristics of JVM and JRE and JDK

- How they work together

Answer

- JDK - Software used for building Java applications
- JRE - Software used for running Java applications
- JVM - abstract virtual machine that the JRE spins up to run Java applications

8. What is Java bytecode?

Motivation

- Understanding of how Java bytecode is created/the compile process
- How it suports the portability of Java
- Relation with JVM

Answer

- Instructions set for the JVM to execute
- Generated by the process of the compilation of a Java program
- JVM takes the bytecode and executes it
- Can not be run natively on a machine
- Bycode is consistent across machines. But JVM implementation may vary
- This enable the "Write Once, Run Anywhere" property of Java

9. What is the difference between Path and classpath?

Motivation

- Understanding system variables
- Understanding the classpath concept
- Why we need to set this variables

Answer

- Both are enviromenment variables
- Path is an operation system specific variable that infuluesces what binaries(executable) are available for running
- Classpath is a Java construct to indicate where all the compiled classes and jars are available. This could be multiple locations
- Path: when I type `java` in the command line, ***JRE*** will search for the `java` binary in the PATH variable

- classpath: all the directory of packages and classes are available in the classpath, ***compilor*** will search for the class in the classpath at the time of compilation, to see if the class is available

10. What is the difference between source path and classpath?

Motivation

- Understanding the project structure
- Java IDE folder structure

Answer

- Sourcepath is where the classes reside(that you write and compile)
- `src` directory is the default sourcepath
- Classpath is where your dependencies are located - libraries, jars, etc.
- Compile loads these when required for compilation.
- Runtime can use these to load bytecode - class path scan

11. What are the memory areas allocated in the JVM?

Motivation

- Distinction between heap and stack
- Class area and native area

Answer

- Heap
  - Space for objects in the memory
  - "Global" - like shared space
  - When instances of objects are created, they are allocated in the heap
  - Reference variables can point to objects base the the context
  - Largest of the memory spaces
- Stack
  - Holds thread level data
  - Local variables and object references
  - Call frames for each method execution
- Code area(meta space)
  - Holds the bytecode, JIT info
- Implementation / native area
  - Resister
  - C implementation stack

- a lot JVM are implemented in C and C++, JVM run on C and C++.
- The low level code to run the JVM

12. What is differnecce bwtween Java permgen and metaspace?

Motivation

- Undersanding the the changes in memory management
- Awareness of the metaspace

Answer

- Memory areas in the JVM
- Before Java 8, there was a memory area called PermGen
- Java 8 onwards, there is a new memory area called Metaspace
- PermGen is gone!
- Memory for the JVM is allocated in the Metaspace
- PermGen had a fiexed max size allocated to it(configurable)
- Metaspace grows dynamically as the program runs
- Has maxMetaSpaceSize configuration. Triggers garbage collection when the metaspace exceeds this size
- Because of class loading / unloading, the metaspace can grow and shrink, no "permanent generation" anymore

13. What is garbage collection?

Motivation

- Awareness of the process of garbage collection
- Some details of working
- Benefits and drawbacks

Answer

- Process of removing unused and orphaned objects from the heap
- Automatic process.Does not need programmer intervention
- Has APIs to trigger programmaticlly(not recommended)
- Generation based approach(young generation, old generation, permanent generation)
- First in first out(FIFO)
- Benefit - no manual memory management
- Drawback - performance intrustion

14. What is JIT compiler?

Motivation

- Familiarity with the how JVM works
- Bytecode vs native code

Answer

- JVM selecctively converts certain bytecode instructions to native code(machine code)
- Converts to the native instruction set of the CPU it is running on
- Makes a judgement call based on usage of the bytecode, and performance characteristics
- Hence "Just in time"

15. Explain heap space configuration.

Motivation

- Familiarity with the heap space
- Configration flag details
- What informs the setting values?

Answer

- Configuration flag: -Xms, -Xmx
- -Xms: initial memery allocation pool size(starting size of the heap)
- -Xmx: maximum memory allocation pool size(don't go beyond this)
- When max heap size is reached
  - The JVM will start to collect garbage
  - or Out of memory errors will be thrown
- You don't want space to be too small(garbage collection will keep happening)
- You don't want space to be too big(waste of memory)

16. Explain the stack space configuration.

Motivation

- Familiarity with the stack space
- Configration flag details

Answer

- Configuration flag: -Xss
- -Xss: initial stack size

- Stack grows dynamically as the program runs
- When all available stack space is used
    - Stack overflow exception will be thrown
- Usually not a big deal
- Stack overflow usually happens when you have a circular invocation

17. What is a classloader?

Motivation

- Awareness of the class loading concept
- Role of classloader

Answer

- Part of the JRE
- It load Java classes into the runtime
- This happens only when needed
    - JVM requests a class
    - Class loader tries to find it and loads it
    - If it can't find it, it will throw a ClassNotFoundException

18. What are the diffrent types of classloaders?

Motivation

- Awareness of the class loading concept

Answer

- Application / System classloader
    - Classes in the class path
- Extension classloader
    - Core Java JDK classes
- Bootstrap classloader
    - Loads the other classloaders
    - Core Java runtime classes
- Custom classloader

19. What is `public static void main(String[] args)` ?

Motivation

- Familiarity with modifiers
- Understanding the application startup process

Answer

- `public static void` are modifiers to a method called `main`
- `public` - method is accessible outside the class
- `static` - class instance is not required to run the method
- `void` - method does not return any value
- `main` - special name convention to indicate an execution entry point

20. What is the order of the modifiers are switched in the `public static void main` ?

answer

- order of modifiers doesn't matter
- Modifiers must be before the method

21. Can you run code before the main method starts?

Motivation

- Understanding static block

Answer

- Yes, you can
- This can be done by using static block
- Static block is executed when the class is loaded
- So, this runs before the main method is executed

22. What is the difference between float and double?

Motivation

- Knowledge of primitive number types
- Familiarity with precision
- Knowledge of sizes allocated

Answer

- Both are real numbers
- Both are imprecise(need inifinite precision)

- Float takes 32 bit. Double takes 64 bit.
- Can not use equality operator to compare float and double
- Double is literally double the size of float
- Double has more precision than float
- By default, floating point numbers are double
- Use float mostly for space optimization
- A double can be cast to a float(with possible loss of precision)

23. Why would you need a break in a switch statement?

Motivation

- understanding the switch statement fall through pattern
- What happens if you don't use a break?
- Possible uses of the fall through behavior

Answer

- Switch case statements are't "discrete"
- If / else is discrete(ether if executes or else executes)
- There isn't a one-to-one map between the matched case the block, if there is not break, the next case will be executed
- Case match is for where the execution starts. After that, execution falls through to the next case all the way to the end of the switch.
- One way to break it is using a break statement
- Can be used without break to group serveral matching statements

24. What are the primitive types in java?

Motivation

- List of primitive types
- Nature of date
- Space consumed

Answer

- Primitive types are:
  - byte - 8 bit signed two's complement integer
  - short - 16 bit signed two's complement integer
  - int - 32 bit signed two's complement integer
  - long - 64 bit two's complement integer(signed / unsigned)

    - float - 32 bit floating point
    - double - 64 bit floating point
    - char - 16 bit unicode character
    - boolean - One of two values: true or false, Size unknown

25. What is default value of local variables?

answer

- The local variable do not have default value.
- They need to be initialized explicitly by the programmer
- Does bot happen automatically.

26. Why does complier complain about local varibles initialization?

Answer

- Java does not initialize local variables automatically
- If you call the local variable before it is initialized, it will throw a compiler error
- These local varibles could be primitive types or reference types
- Ebore "using" a local variable, you need to have put some value to it first.
- Can be completly uninitialized. As long as it is not used, it will not be initialized.

27. Can a double be cast into a byte?

Motivation

- Understanding the casting process
- Understanding the precision of floating point numbers

Answer

- Yes, you can. A higher precision floating point number can be cast to a lower precision floating point number
- This needs explicit casting

```
byte b = (byte) d;
```

- Possible loss of data(lossy conversion)

28. Can a byte be cast into a double?

Motivation

- Understanding the precision
- Implicit casting

Answer

- A byte does not need to be cast to a double
- It can be automatically assigned

```java
byte b = 1;
double d = b;
System.out.println(d);
```

- This is called implicit casting
  - When the lower precision number is assigned to a higher precision number
- No possible loss of data(lossless conversion)

29. How do you break a nested loop?

Motivation

- Break statement with label
- Thoughts on best practice
  - Like a goto statement
  - should be used with caution

Answer

- Break statement with label

```java
outer:
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    if (i == 5) {
      break outer;
    }
  }
}
```

- Indicates the statent to break out of, not where to go to
- Not a "bad" thing to use
- Diffrent from goto statement in that you cannot alter the flow of control

30. What are the diffrent access modifiers?

Motivation

- Familiarity with modifiers
- Java access rules and diffrent types of access modifiers

Answer

- Private
  - Within the class only
- Pckage private(default)
  - Within the package only
- Protected
  - Within the same package and subclasses
- Public
  - Cab be accessed by all

31. Can you overload constructors? How does it work?

Motivation

- Method overloading concept
- Constructor arguments with new operator

Answer

- Yes, you can
- Same concept as method overloading.Diffrent is that you can use the new operator to create an object
- diffrent arguments list
- Needs arguments passed via new keyword constructor invocation

```
class Person {
  String name;
  int age;
  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  Person(String name) {
    this.name = name;
  }
}
```

- Overloaded contructor "hides" default no-arg constructor. Can't construct without those args anymore.
- No-arg constructor has to be implemented by the programmer

32. How to copy a constructor?

Motivation

- Constructor arguments
- What is the pattern and when to use it

Answer

- Specific pattern used to create a new object as a copy of another object
- Custtrotor of a class takes as argument of the same class type
- Constructor copies memebers from that instance

```
class Person {
  String name;
  int age;
  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  Person(Person p) {
    this.name = p.name;
    this.age = p.age;
  }
}
```

33. What is constructor chaining?

Motivation

- What is the pattern and when to use it
- Why do you need it

Answer

- Constructor chaining is a pattern that allows you to call a constructor from another constructor to delegate part of the initialization
- Usually constructor with more args delegates to constructor with less args

```java
class Person {
  String name;
  int age;
  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  Person(String name) {
    // this has to be the very first line in the constructor
    this(name, 0);
  }
}
```

34. What is the singleton pattern?

Motivation

- Design pattern
- How to implement it in java
- Benefits of using it

Answer

- Singleton pattern is a design pattern that restricts the instantiation of a class to one object.
- Java naturally allows multiple instances. The requirement is to enforce one instance.
- Done by
  - Private constructor
  - Static instance
  - Static method that returns the instance

```java
class Person {
  private static Person instance = new Person();

  private Person() {
  }

  public static Person getInstance() {
    return instance;
  }
}
```

- Benefits and uses
  - Single instance guaranteed
  - Useful for certain objects Example: DB connection

    - Controlled life cycle

35. What is auto-boxing and unboxing?

Motivation

- Knowledge of diffrent wrapper classes
- Why are they needed
- Boxing and unboxing behavior

Answer

- Java has an automatic mechanism for converting primitive types to their corresponding wrapper classes and vice versa.
- you don't need to call the constructor of the wrapper class for create a new object
- Assiemments work with primitive types

```
int i = 1;
Integer i2 = 2;// equivalent to Integer i2 = new Integer(2);
```

- Method arguments and collections work

```
List<Integer> list = new ArrayList<Integer>();
list.add(1);
list.add(2);
list.add(3);
```

36. What is the final keyword?

Motivation

- Understaning of modifiers
- Immutability concept
- Limitations with object references
- Other uses
    - final class
    - final field
    - final method
    - final variable

Answer

- Final keyword marks something as in its final state and not to be changed

- On a varibalbe, it means that it is immutable(At least on primitive types)

```
final int i = 1;
```

- On object refrence, that means the freference is constant, but the object instance is not

```
final Person p = new Person();// p can not point to another object
```

- On a method, that means the method is final and cannot be overridden in the children class

```
final void method() {
}
```

- On a class, that means the class is final and cannot be extended in the children class

```
final class Person {
}// Person can not be extended,can not be inherited
```

- On a field, that means the field is final and cannot be overridden in the children class

```
final class Person {
  final int age;
}
```

37. What are wrapper classes?

Motivation

- Knowldge of diffrent wrapper classes
- Why are they are used
- Boxing and unboxing behavior

Answer

- Classes corresponding to primitive types - that "wrap" primitive values
- Used to with Collections -Java collections support only reference types only
- byte, short, int, long, float, double, char, boolean
- Byte, Short, Integer, Long, Float, Double, Character, Boolean

38. What is *this* keyword?

Motivation

- The object reference
- Why do you need it?

Answer

- This is a reference that points the object whose code (method) is executing.
- Used only in instance methods(Can not be used in static methods)
- Useful for accessing member variables when shadowed

```
class Person {
  private String name;
  private int age;
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

```
class Person {
  private String name;
  private int age;
  public Person(String name){
    this.name = name;
  }
  public Person(String name, int age) {
    this(name);
    this.age = age;
  }
}
```

39. What is abstraction?

Motivation

- Concept understanding
- How it affects the class design
- Interface vs implementation

Answer

- Abstraction is the design principle of separating the interface from the implementation so that the consumer / client only concerned with the interface.
- Example:
  - Button on an electric / electronic device vs internal circuit board
  - Manifests in the class design using interfaces (and sometimes abstract classes)

- abstract classes vs interfaces
  - abstract classes are used for consolidating common methods and variables are exsiting in multiple classes
  - abstract classes are for author
  - interface is for consumer
- Abstraction vs encapuslation
  - Abstracton cab be enforced by encapsulation
  - encapuslation is strict abstraction and is guranteed can not be accessed
  - abstraction hides implementation details
  - encapsulation protects the implementation details from access

40. What is encapsulation?

Motivation

- Concept understanding

Answer

- Encapsulation is the process of restricting access to the inner implementation details of a class. It enforces abstracton concepts by not just ***hiding*** but by guaranteeing the internals are not exposed.
- it is not just the outer do not see it, the outer is imposible to see it.
- Example:

```java
class Person {
  private String name;
  private int age;
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
}
```

- Manifests in class desgin using access modifiers - like private.
- Benefits include ability to refactor/change internals without breaking others

41. How does a constructor work?

Motivation

- Object creation and memory allocation
- How constructor is called
- Diffrence from methods

Answer

- A method that can initialize values of an object
- The runtime creates an instance. You can work on the instance before it is ready
- for doing all the prep work for the instance
- Looks like any instance method but is not.
  - No return
  - specific naming convention
- Invoked by the new operator + object creation process

```
class Person {
  private String name;
  private int age;
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

- constructor exits by default(no args constructor)
- You can implement or parameterize the constructor

42. What is marker interface in java?

Motivation

- Why are they used
- example

Answer

- An interface that marks or "tags" classes and their corresponding instances
- Don't have any methods of their own
- Example: Serializable, Cloneable, Comparable
- Formerly used because annotations didn't exist
- Not recommended these days

43. What is the initial value of instance variables?

Motivation

- Understading of the default values
- Constrast with local variables

Answer

- Instance varaibles don't need to be initialized
- Constrasts with local variables that error when used without initialization
- Primitive values take "default" values
  - `0` for numeric types
  - `\000` for char
  - `false` for boolean
  - `null` for reference types
- Recommended initialization in constructor(especially object reference)
  - if you don't initialize, it will be null
  - at the runtime, you will get `NullPointerException`

44. What is method overriding?

Motivation

- Parent class and sub class concept
- How to override a method
- What it results in (polymorphism)

Answer

- Overriding is when a subclass changes the behavior of an inherited method
- Done by the subcalss implementing a method with the same signature as the parent class
- Works for multiple levels too. Methods can be overridden irrespective of where it inherited from.
- It can be overridden the method inherited from the `Object` class
- Actual method gets resolved at runtime. This allows for polymorphism.(runtime polymorphism)

45. Is java Pass by value or pass by reference?

Motivation

- Familiarity with the java instance
- Understanding of the difference between primitive and reference types

Answer

- ***Java is pass by value***
- for primitive types
  - the variable holds the actual value of the primitive type
  - it is passed by value
- for reference types
  - the variable holds the reference to the object allocated in the heap(never use pointer)
  - it is passed by the value of the reference

46. What is the superclass of all classes in java?

Motivation

- `Object` superclass
- What it provides

Answer

- `Object` is the superclass of all classes in java
- Root of the inheritance hierarchy
- `Object` class contains some handy methods like
  - `toString()`
  - `equals()`
  - `hashCode()`

47. What is static modifier?

Motivation

- familiarity with the java instance
- Instance value vs "global"

Answer

- (static varibles) not associated with any instance
  - When you need to store data that is relivent across instances (static variables)
  - Example:
    ```
    Math.PI
    ```
- static method
  - when you need to provide method before creating the instance
  - `public static main(String[] args){}`

- static block
    - when you need the code to be executed before creating the instance
    - `static { }`
    - Example:
      `static { System.out.println("Hello"); }`

48. What is the differnce between .equals() and ==?

Motivation

- Object reference comparison vs value comparison

Answer

- `==` is an operator that compares two values for equality
- Works with primitive type.
- Does not work with reference type
- Only compare the values of reference to the object
- Every object inherits equals method from the `Object` class
- Classes can implement their own `equals` method to check for equality

49. What are the fifferent varible scopes in java?

Motivation

- Understanding of scoping
- What forms a scope

Answer

- class level scope
    - member variables
- Method level scope
    - local variables
    - parameters
    - return values
- Block level scope
    - loops
    - if blocks

50. `Overloading vs Overriding`

Motivation

- Understanding of the concept
- How overloading works with overriding

Answer

- Overloading
  - Overloading is when you have multiple methods with the same name but different signatures(arguments)
  - Argumetns dicide which one to call
- Overriding
  - A child class providing implementation to change the behavior of the inherited method
  - Instance decide which one gets called
- Overriden methods can be overloaded

51. `continue` vs `break` in java

Motivation

- Understanding of loops and control flow
- When to use continue
- When to use break
- Sample situation

Answer

- Both are used in loops
- Break is used to end the loop immediately
- Continue is used to end the particular ***iteration*** of the loop
- Example:
- Looping an array until you find a value

```
for(int i = 0; i < array.length; i++) {
  if(array[i] == value) {
    break;
  }
}
```

- Example:
- Processing every element, but skipping some on some condition

```
for(int i = 0; i < array.length; i++) {
  if(array[i] == value) {
    continue;
  }
  // do something with array[i]
}
```

- `break` in `switch`
  - Example:

```
switch(value) {
  case 1:
    System.out.println("one");
    break;
  case 2:
    System.out.println("two");
    break;
  default:
    System.out.println("other");
}
```

52. Can static methods be overridden?

answer

- static method can not be overridden in java
- Method overriding has been designed to work with polymorphism

Core Java and OOP(Infosys questions)

What is DAO?

Difference between abstract classes and interfaces?

What is Encapsulation and Abstraction?

Analytical question: Difference between method and function..

How to handle class not found exception

Describe one thin that introduced in the java 8?

Tell me about functional interface?

Do you know Method References?

Difference between lambda expression and method reference?

Tell me about Stream API?

When is each method of software development apply?

What is java 8 features?

Describe Abstract in OOP

what is Java streaming?

core java: polymorphism, abstract classes and interfaces(difference), AbstractionAws, aws

storage class

streams

Core java: what is JVM ?

What is polymorphism?

Encapsulation and access modifier,the types of access modifier,Which one is more protective?

How many constructors you used in java?

Can you do static overriding?

How you connected your db in java?

Analytical question: Difference between method and function..

How to handle class not found

exception

Java 8 interfaces and features

MapException handling

Difference between string buffer and String Builder

What is java 8 features?

Describe Abstract in OOP

what is Java streaming?

difrenerence between Throw and Throws?

what is pojo design pattern?

- ***Transfer Object*** is a simple POJO class having ***getter/setter*** methods and is ***serialized*** so that it can be ***transferred over the network***. Server Side business class normally fetches data from the database and fills the POJO and sends it to the client or passes it by value. For clients, the transfer object is read-only. The client can create its own transfer object and pass it to the server to update values in the database in one shot.