**Java is pure Object-oriented programming?**

**JVM is platform dependent or independent?**

JVM is platform dependent and java language is platform independent. And java also called as **WORA** (Write once run anywhere)

**Why JVM is called secured?**

Because it runs byte code on load.

**Methods of Object class:**

* toString()
* clone()
* getClass()
* finalize()
* equals()
* hashcode()
* notify()
* notifyAll()
* wait()

**Ways to create Object in java:**

* new
* newInstance()

ClassName className= (ClassName) Class.forName(“ClassName”).newInstance();

* clone()

ClassName obj = new ClassName();

ClassName clonedObj = (ClassName) obj.clone();

* readObject() - deserialization process

FileInputStream fis = new FileInpuitStream(“text.txt”);

ObjectInputStream oos = new ObjectInputStream(fis);

ClassName obj = (ClassName) oos.readObject();

* newInstance() method of Constructor class

ClassName 0bj= ClassName.class.getDeclaredConstructor().newInstance();

**How to get class members?**

Field[] members= className.class.getDeclaredFields(); //It gets all public and private members

Field[] members = className.class.getFields(); //it gets only public members

**Different ways of cloning:**

* Deep cloning
* Shallow cloning

**Contract between hashCode() and equals():**

* If two objects are said to be equals, hash code must be same
* If two objects hash code are same, it doesn’t need to both objects be equal

Ex: MyObject a = new MyObject("a", 123,"something");

MyObject b = new MyObject("a", 123,"something");

Above, looks both the objects are equals, but it’s **hashcode will be different** hence even thou they are equal if we perform a.equals(b) it will be **false**.

**Why hash code? What are the benefits?**

You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.

**Immutable class:**

* Class should be declared as final
* All member variables should be declared as final
* If we have any **mutable objects** as class members, Inside the constructor, make sure to use a **deep clone copy of the passed argument** and set it to the mutable field.
* Define public parametrized constructor
* No setters

\*All wrapper class are immutable class

**Benefits of immutable class’s:**

* Simple to design and implement
* Concurrency- immutable objects can be safely shared among multiple threads.
* These class types can be used as keys when dealing with hashing
* No invalid state, once value is assigned its can’t be changed

**Oops concepts:**

* Polymorphism
* Encapsulation
* Inheritance
* Abstraction
* Aggregation, composition & association

**Difference between Encapsulation and Abstraction?**

Encapsulation is wrapping up data in a specific class.

Abstraction is the process of hiding the implementation of method and showing only function name. It’s all about what it does and hiding how it does?

**Can we achieve Abstraction without encapsulation? NO –** But we can achieve encapsulation without abstraction.

**What is inheritance?**

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. ... The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

**\*\*Multiple inheritance with class’s is not allowed in java. Multi-level inheritance is allowed.**

**What is Polymorphism?**

One to many forms. We have Static (Compile) Polymorphism and Dynamic polymorphism. Overloading process will happen at compile time is known as static polymorphism and Overriding process will happen at runtime it’s known as dynamic polymorphism.

**Overloading:**

* Overloading happens only in single class. It’s used to more readable purpose.
* Overloading can’t be done just by changing the return type, no of parameters or data types of parameters must be modified in order to achieve overloading.

**Can we overload static main method?**

Yes, main method which has parameter of String[] type will be executed.

**Overriding:**

* It happens in two different class’s and its meant to provide very specific implementation for a specific class.
* Method name and parameters should be same.
* Return type can be of same or covariant return type.

**Can we override static methods?**

No, static refers to compile type and it’s belongs to class. Overriding is dynamic, it always happens during runtime. If do same, it will be referred as method hiding.

**What is Interface?**

An interface is a reference type in Java. ... It is a collection of abstract methods. Like a class, an interface can have methods and variables, default methods, static methods, and nested types (class inside in it)

* By default, variables declared in an interface are referred as **public static final.**
* Methods declared in an interface are by default **public abstract** (only method signature, no body).
* An interface which is declared inside another interface or class is called **nested interface**.
* An empty interface is known as **tag or marker interface**. For example, Serializable, Cloneable are tag interfaces. These interfaces **do not have any field and methods** in it.

**Why do we use interface?**

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance.
* It is also used to achieve loose coupling.

**why use interfaces when we have abstract classes?**

The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static. A subclass can extend one class, but it can implement many interfaces.

**What is Serialization?**

Serialization is the translation of your Java object's values/states to bytes to send it over network or save it. **We need serialization because** your Hard disk and network infrastructure are hardware components that understand bits and bytes but not JAVA objects.

**What is UUID?**

Java.util.UUID class in Java that represents an immutable universally unique identifier (UUID). A UUID represents a 128-bit value. It implements **Serializable** and **Comparable** interfaces

**Serial version ID-**By default its value is 1L, it’s used to uniquely identify object during serialization and deserialization.

Ex: While receiving the data at B side from A, if a receiving object of class is modified at the B side and same class is unchanged at A. In this scenario at B side somehow must map that of object to old class design so that object sent from A will be able receive at B side.

**Java features:** <https://www.javatpoint.com/New-features-in-java>

**Java 8 features:**

* Method Reference
* Lambda expression
* Functional interface
* Default and static methods in interface
* forEach
* streams
* Optional class
* Date API and Time API

**Java 7 features:**

* Try-with resources
* Catching multiple exceptions in single catch
* String in switch statement

**Java 5 features:**

* Enhanced for loop (for-in loop)
* Enum
* Static import
* Generics
* Covariant return type
* Autoboxing & Unboxing
* Varargs -- public void display (String… values)

**Singleton Class:**

* Restricts the instantiation of a class to one object and same will be used in the entire application.

Ex: logging class, DB connection class, configuration manager, error manager etc.

* **Design:** declare static private member, private constructor and public static factory method which creates or returns created object.

**Exceptions types:**

* Checked Exception - Compile time exception
* Unchecked Exception- Runtime or Null Pointer exception, must be handled by programmer
* Error-Virtual machine error, out of memory error should not be included in catch block
* Key points:

\* Throwable is the parent of all the exception class’s

**\*** try block is always followed by catch or finally block.

**\*** For each try block there can be zero or more catch blocks, but **only one** finally block.

**\*** Finally block will be executed always except in few cases such as System.exit(0) or any

Errors.

**Exception rules:**

**Throws():**

By default, a checked exception won’t be propagated, using throws keyword in the method declaration checked exceptions are gets propagated.

**Collections:**

Collections is the class and collection is the interface and it’s derived from Iterable interface.

**Difference between ArrayList & LinkedList?**

**List v/s Set**

**What is fail fast and fail safe?**

**Spring Boot:** one of most popular and developer friendly technology of spring framework

**Advantages:**

* Developer friendly and avoids boiler plate code, Annotation and xml Configuration.
* Production ready application - reduces lots of development time and increases productivity.
* Its **opinionated default configurations–** it makes decision on it’s own and we do have control over on it.
  + For example, if you include the spring boot starter pom for JPA, you'll get autoconfigured for you an in a memory database, a hibernate entity manager, and a simple datasource.
* In builtembedded tomcat server – no need generating war and deployment
* **Actuator –** to monitor and manage our application
* **Spring Initializer**
* It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.
* Easy to integrate with Spring JDBC, Spring ORM, Spring Data, in memory databases, Spring Security etc.
* Exception handling

What does @SpringBootApplication?

What does @SpringTransactional?

How spring data JPA works?

**Why Spring boot is used to develop microservices?**

**What are Microservices?**

**Advantages of Microservices?**

**What are Rest Services?**