JAVA IMPORTANT NOTES

BASICS :

* Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components Runtime Environment and API.
* Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
* **Static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
* At runtime, following steps are performed: Class file…….classLoader….ByteCode Verifier……Interpreter….Runtime…..Hardware.
* **Classloader:**is the subsystem of JVM that is used to load class files. **Bytecode Verifier:**checks the code fragments for illegal code that can violate access right to objects. **Interpreter:**read bytecode stream then execute the instructions.
* JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.
* Java uses Unicode system than ASCII code system. **lowest value:**\u0000, **highest value:**\uFFFF
* The bitwise | operator always checks both conditions whether first condition is true or false.
* The for-each loop is used to traverse array or collection in java. Example ….. **for**(Type var:array){   //code to be executed   }

OOPS CONCEPTS:

* A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.
* Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages
* Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.
* Anonymous simply means nameless. An object which has no reference is known as anonymous object. It can be used at the time of object creation only. If you have to use an object only once, anonymous object is a good approach. Example .. **new** Calculation();
* There are many ways to copy the values of one object into another in java. These are By constructor, By assigning the values of one object into another or by By clone() method of Object class.
* Does Constructor return any value?? Yes, that is current class instance (You cannot use return type yet it returns a value).
* **Rule:** Constructor runs before object is assigned to reference variable
* The **static keyword** in java is used for memory management mainly. The static variable gets memory only once in class area at the time of class loading.
* A **static method** belongs to the class rather than object of a class. A static method can be invoked without the need for creating an instance of a class. static method can access static data member and can change the value of it.
* **Restrictions in static method:** this and super cannot be used in static context. The static method cannot use non static data member or call non-static method directly.
* **this keyword** is a **reference variable** that refers to the current object.
* The this() keyword written as “ this(); ” in a constructor can be used to invoke the current class constructors(default in this case). It is used to reuse the constructor. In other words, it is used for constructor chaining, i.e. call one constructor from other constructor of same class.
* **Rule:** Call to this() must be the first statement in constructor.
* The **this** keyword can also be passed as an argument in the method. It is mainly used in the event handling in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.
* If a class has an entity reference(i.e. instance variable which is non-primitive), it is known as Aggregation. Aggregation represents HAS-A relationship.
* If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
* **Rule:** In java, Method Overloading is not possible by changing the return type of the method only.
* **Can we overload java main() method?** Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.
* If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**. Method overriding is used for runtime polymorphism.
* **Rule:** static method cannot be overridden because static method is bound with class.
* The **covariant return type** specifies that the return type may vary in the same direction as the subclass. Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.
* The **super** keyword in java is a reference variable which is used to refer immediate parent class object. **Super()** is added in each class constructor automatically by compiler if there is no super() or this().
* **Instance Initializer block** is used to initialize the instance data member. Instance initializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked.
* **Java Final Keyword:** Stop Value Change(final variable) , Stop Method Overriding(final method) , Stop Inheritance(final class).  Final method is inherited but you cannot override it. Blank final variable or say uninitialized final variable can only be initialized in constructor.
* A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.
* **Can we declare a constructor final?** No, because constructor is never inherited.
* If you **overload** static method in java, it is the example of **compile time polymorphism**.
* **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable. **NOTE:** runtime polymorphism can't be achieved by data members.
* **Bindings:** Connecting a method call to the method body is known as binding. When type of the object is determined at compiled time (by the compiler), it is known as **static binding**. If there is any private, final or static method in a class, there is static binding. When type of the object is determined at run-time, it is known as **dynamic binding**.
* **DownCasting with instanceOf Operator :** When Subclass type refers to the object of Parent class, it is known as downcasting. If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime. But if we use instanceof operator, downcasting is possible.
* A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.
* The abstract class can also be used to provide some (not all methods) implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.
* **Rule:** The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.
* A class extends another class, an interface extends another interface but a **class implements an interface**.
* Default method in Interface: We can have method body in interface. But we need to make it default method.
* An interface that has no member is known as **marker** **or tagged interface**. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.
* **Rule:** A class cannot be private or protected except nested class.
* The **default** **modifier** is accessible only within package. The **protected access modifier** is accessible within package and outside the package but through inheritance only.
* **Cloning Object:** The Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.
* **Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive.
* Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.
* **Java strictfp keyword** ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The strictfp keyword **cannot** be applied on abstract methods, variables or constructors. The strictfp keyword **can** be applied on methods, Interface and classes.

STRING CONCEPTS:

* **The String class** implements *Serializable*, *Comparable* and *CharSequence* interfaces.
* The **CharSequence** interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.
* **RULE:** The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.
* **NOTE:** String objects are stored in a special memory area known as string constant pool.
* **How to compare string?**  String **equals() method** : It compares values of string for equality,, The **= = operator** compares references not values,, String **compareTo() method**: compares values lexicographically and returns an integer value.
* **RULE:** After a string literal, all the + will be treated as string concatenation operator. Example
* **Substring :** startIndex: inclusive while endIndex: exclusive.
* **String Intern():**When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.
* **Rule:** Java StringBuffer class(Mutabel) is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order. It means two threads can't call the methods of StringBuffer simultaneously
* The Java **StringBuilder** class is same as StringBuffer class except that it is non-synchronized, i.e. not thread safe. It is available since JDK 1.5.
* **IMMUTABLE CLASS:** There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members.
* The **toString()** method returns the string representation of the object. If you print any object, java compiler internally invokes the toString() method on the object.

EXCEPTION HANDLING:

* The Java throw keyword is used to explicitly throw an exception. Example
* **RULE:** By default Unchecked Exceptions (Runtime) are forwarded in calling chain (propagated). By default, Checked Exceptions are not forwarded in calling chain (propagated).
* Exception Handling is mainly used to handle the checked exceptions. The **Java throws keyword** is used to declare an exception.
* **Rule:** If you are calling a method that declares an exception, you must either caught or declare the exception.
* **Finalize** is used to perform clean up processing just before object is garbage collected. It is a method.
* **RULES:** If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
* **RULES:** If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

REFLECTION API:

* **Java Reflection** is a process of examining or modifying the run time behavior of a class at run time. The java.lang and java.lang.reflect packages provide classes for java reflection.
* The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.
* **3** **ways to get the instance of Class class:** forName() method of Class class , getClass() method of Object class and the .class syntax
* **forName():**t should be used if you know the fully qualified name of class. This cannot be used for primitive types. Example
* **getClass():** It should be used if you know the type. Moreover, it can be used with primitives.Example
* **.Class :** If a type is available but there is no instance then it is possible to obtain a Class by appending ".class" to the name of the type. Example
* The**newInstance()** method of Class class and Constructor class is used to create a new instance of the class. The newInstance() method of Class class can invoke zero-argument constructor whereas newInstance() method of Constructor class can invoke any number of arguments. So Constructor class is preferred over Class class.
* **How to call private method from another class in java?** See this example:
* ****
* ****

DATE TIME AND CONVERSIONS:

* **LocalDate** class is an immutable class that represents Date with a default format of yyyy-MM-dd.
* **LocalTime** class is an immutable class that represents time with a default format of hour-minute-second.
* **LocalDateTime** class is an immutable date-time object that represents a date-time, with the default format as yyyy-MM-dd-HH-mm-ss.zzz.
* **Calendar** class is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. Get its instance using the getInstance() method in it.
* Two classes for formatting date in java: **DateFormat**(Abstract class)and **SimpleDateFormat(**Concrete class**)**. Formatting means date to string*and*parsing means string to date*.*
* We can convert **String to int in java** using Integer.parseInt() method. Example
* We can convert **int to String in java** using String.valueOf() and Integer.toString() methods. Example

COLLECTIONS IN JAVA:

* The **java.util** package contains all the classes and interfaces for Collection framework.



* **Iterator** interface provides the facility of iterating the elements in forward direction only. It has 3 methods **public boolean hasNext(),public object next() and public void remove()** .