**Core Java\_\_**

**OOP's Concept:**

**Inheritence:** one class inherites the field member and member function of another class. code **reusablity**, abstraction. used to achieve runtime polymorphism.

**Abstraction**: Abstraction is a process of hiding the implementation details and showing only functionality to the user. abstraction is achieved by interfaces and abstract classes.

**Polymorphism**: one task is performed in different ways, it is known as polymorphism. we use method overloading(runtime) and method overriding(compile time) to achieve polymorphism.

**Encapsulation**: Binding (or wrapping) code and data together into a single unit are known as encapsulation. Java bean is the fully encapsulated class because all the data members are private here and have public getter and setter.

**Coupling**: refers to the knowledge or information or dependency of another class. we use private, protected, and public modifiers to display the visibility level of a class, method, and field. use interfaces for the weaker coupling.

**Cohesive** refers to level of component which performs a well defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts.

**Association**: represents the relationship between the objects. four types - One to one, one to many, many to one and many to many.

Aggregation: is a way of achieve association. It represents the weak relationship. It termed as a has-a relationship.

**Composition**: is also a way to achieve Association. It represents the strong relationship.

**Primitive datatype**: 8, are the building blocks of data manipulation. default deiifers for different primitive type.

**NonPrimitive datatype**: class, interface, arry, user-defined and used to store multiple values. An object reference variable lives on the stack memory and the object in heap. default value of any non-primitive is null. Class, String, Array, Object....

**Constructor, initializer block, static block, methods**

**public, private, protected, Default**

**static, final, this, super**

**CompileTime Exceptions**

**Runtime Exceptions**

**Exception Propagation**

Difference between **abstract class and interface**: interface can perform multiple inheritence but abstract class

**use of this keyword?**

**can we override a default method**. - yes but it is not recommended or needed.

**how to resolve the ambiguity throw default method.** - interface.super.function()

**can we override the static method**. - no

**benefits of using functional interface?** - helps to achieve functional programming approach, we can shorten our code using Functional interface and lamda expression.

**What are Consumer interface** - it take single element and returns no result. abstrct accept() and default andThen()

**benefits of lamda expression** - methods store in variable, method of interface multiple implementations, code reduce.

**Stream API** - process the data in Collection, processing, lazy intialization. terminal

String literal, String with new keyword, String builder.

**Method Reference**: used to refer method of functional interface. compact and easy form of lambda expression.

Reference to a static method.

Reference to an instance method.

Reference to a constructor.

7. **Collection** (List, ArrayList, LinkedList, Set, HashSet, LinkedHashSet, TreeSet, Queue, PriorityQueue, Map, HashMap, LinkedHashMap,TreeMap, Comparable, Comparator)

8. **String Literals** - simply a reference to an instance of the String class, which consists of zero or more characters enclosed in double quotes.

**String** - is a sequence of characters.

**String Builder** - It is basically used to make a string mutable, it is used when we want to manipulate string again and again, it is asynchronized

**String Buffer** - It is synchronized and slower than String builder,

**String Poo**l - String pool is a common space or memory where all the String are stored which is defined using String literals.

9. **Default and static method in interface**

10. **Functional Interface** - contains one abstract method, helps to achieve functional programming approach.

11. **Predefined Functional Interfac**e

(**Predicate** - boolean-valued function, one argument, test(), and() or(), java.util

**Consumer** - used as assignment target interate, one args, accept(), andThen(), java.util

**Supplier** - used to produce values, no args, get(), java.util

**Function** - used to refer method by specifying type of parameter. It sends result back to referred function, one args one return, apply(), andthen() compose() identity())

12. **Lambda** - A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method. It simplifies the annonymous inner class and compiler understand the code and write it for us.

Method Reference -

13. **Stream API** - Java streams represent a pipeline through, which followed by zero or more intermediate operations, and a terminal operation

Reflection in Java:

**Reflection** is an API that is used to examine and modify the behaviour of methods, classes and interfaces at runtime.

infor about class and method that can be execute using that object.

Class c = Class.forName("com.main.Emp");

we can execute method at runtime irrespective of access spefier used.

to invoke method in reflection API getDeclaredMethod() and invoke()

Class.getDeclaredMethod(name, parametertype)

Method.invoke(Object, parameter)

Class.getDeclaredField(FieldName)

Field.setAccessible(true)

**Collections**:

**Queue**: uses FIFO order.

queues in java.util package are known as unbounded queue. i.e. queue that are not bound by capacity i.e. we do not need to provide size of the queue.

queues in java.util.concurrent are known as bounded queue.

**Java-8\_\_**

**Java Functional interface**: An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method.

**How we can acheive polymorphism?**

**Difference between abstract class and interface?**

**How we can achieve abstraction? hiding implementation.**

**What is encapsulation? with private and public getter/setter.**

**What is difference in primitive and non-primitive? in terms of storing**

**can we overlaod and override the static method? static belongs class**

**how final not allow to overide the method?**

**Difference between throw and throws?**

**What is an exception? and Exception Propagation?**

**try-catch? handle multiple-exception with single catch**

**use of super()**