Object-oriented paradigm allows for encapsulation, abstraction, inheritance and polymorphism in the software.

Structurally, an object-oriented app is composed of classes and interfaces.

Class is like a blueprint of specific object. A Class wraps fields/properties and methods, can throw an exception.

A java class is composed of:

- constructor
- Fields/Properties
- methods

The Java class thus provides for encapsulation by binding both the data as well as the methods that operate on it. The data (maintained in fields) is generally accessed through its getter and setter methods to support data hiding.

A class is instantiated using the “new” operator resulting in an object instance.

Classes and interfaces could be grouped/scoped as follows:

- package
- jar file (contains one or more namespaces) .

Static fields and methods:

- belong to the class itself and not to the objects of that type.
- A static field will have one value for all the objects whereas if the field is not static then each object can have its own copy of the field.
- The static methods similarly exist even before an instance of the class is created and can be accessed directly by referencing the class name.
- The static methods in turn cannot access the instance methods and fields directly.

The members of a java class can be

- public:
    - accessible from all classes
- private:
    - accessible only from the code within the same class
- protected:
    - accessible only from the code within the same class, in a class that is derived from that class, or in the same package
- default:
    - accessible only within the same package.

A java class itself can be:

- public
- default (visible within the package)
- abstract
- final
- nested/inner

A nested class can be static or private as now it is simply a member of the outer class. Nested classes that are declared static are called static nested classes. Non-static nested classes are called inner classes. An inner class has access to other members of the enclosing class even if these members are private.

A class can also be a generic that encapsulates operations that are not specific to a particular data type.

Abstraction is hiding of the implementation details from the users. It is supported in Java via abstract classes and interfaces. An interface is composed of fields and method signatures. The implementation of the methods is not provided and only the name, parameters, return type and the exceptions that the methods may throw are specified. A class implementing an interface then provides the implementation of the methods. An abstract class is thus also a type of class that may have abstract methods i.e. methods whose signatures are defined but implementation is not provided. An abstract class cannot be instantiated but it can be subclassed. The subclass usually provides the implementations for all of the abstract methods in its parent hierarchy for it to be instantiated.

Support for inheritance in object-oriented programming allows classes to inherit commonly used state and behavior from other classes. In the Java programming language, each class is allowed to have one direct superclass but each superclass has the potential for an unlimited number of subclasses. A class can implement multiple interfaces though. The use of the keyword “extends” denotes inheritance whereas “implements” obviates implementation of interfaces. Constructors and methods or classes designated as final are not inherited. The constructor of the derived class can call the constructor of the super class and cause its creation by calling super() method and specifying the required parameters. Any other method of the base class could similarly be called by using the notation super.methodName(…).

Polymorphism is supported through method overloading and overriding. Method overloading is an example of static polymorphism, whereas method overriding manifests dynamic polymorphism. Overloading occurs when two or more methods in one class have the same method name but different parameters. Overriding means having two methods (one in the parent class and the other in the child class) with the same signature. Another factor that facilitates polymorphism is the static and dynamic bindings. The private, final and static methods and variables use static binding and bonded by compiler while other methods are bonded during runtime based upon runtime object.

Try-catch-finally block is as follows:

try – A try block is used to encapsulate a region of code. ...
catch – When an exception occurs, the Catch block of code is executed. ...
finally – The finally block allows you to execute certain code if an exception is thrown or not.

Self study the keyword volatile and compare it with keyword final when used with a variable.