**Java:** Java is a high-level, compiled, strongly typed object-oriented programming (OOP) language. Java

is write once run anywhere programming language (**WORA**). Java is **safe and easier** to use.

Java is **open source** and **free**. Java has many **library and frameworks**. **Automatic memory**

**management**

**WORA**: Platform independent, not constraint to single operating system

**JVM**: Java Virtual Machine reads the compiled java bytecode and converts it into machine code. It

executes code line by line

**JRE**: JRE provides the class, libraries & other resources to run the Java program. The JRE contains JVM

in it. So, to run Java we need JRE

**JDK**: JDK develops and executes Java programs. The JDK also has a JRE inside of it, so if you install a JDK

you can compile your Java code as well as execute it.

**Stack Memory**: used to store items with short life. Temporary memory. Ex: variables & reference of

objects. Stack memory contains reference to the object.

**Heap Memory**: used to store global variable and objects. Heap is the run time memory and store global variables

**Variable:** In java variable are used to store data. Variables are stored in memory. Two Types: **Primitive**

and **Reference**

**Primitive**: byte-8, short-16, char-32, int, long, float, double, Boolean

**Reference**: Object, String, Wrapper classes, Arrays

**Class**: blueprint or templates used to instantiate the object. Contains variables and methods

**Object**: entity with state and behavior. Create by instantiation of class. Used new keyword to create

**Method**: reusable block of code that can take in arguments and can return something (datatype or

object)

**Ex (Signature): accessModifier returnType Name (Parameter) throws … {}**

**Constructor**: type of method as a class name that has no return type and used to create an instance of

a class (object). When the new object is created, the constructor is run. Constructor is used to

initialize instance variables.

**Types**: default, no args, parameterized(overloaded)

**Constructor Changing:** The process of calling one constructor from another constructor is called

constructor chaining. Or the way of calling the constructor of parent class from the constructor

if child class is called constructor chaining. Default no args constructor call super class

constructor.

**OOP**: OOP is the concept where java code can be break down into smaller pieces called classes and

Object where possible. There are four pillars of OOP:

**Inheritance**: The ability for entity to adopt the variable (fields) and methods (behavior) of super class.

Can be done by extending classes or implementing interface. Help to code reuse, reduce

duplication Ex:

**Polymorphisms**: Ability for code structure to be treated as different form during the compilation or the

run time. Can be done by method overloading or method overriding

Ex: Mobile Phone: (phone, calculator, play game, watch video)

**Method Overloading**: creating multiple methods in the same class with the same method name

and different parameters. Example of compile time Polymorphisms. Can not be perform by

changing return type

**Method Overriding**: method in sub class with the same method signature as the parent class but the different implementation. Example of run time polymorphism.

**Encapsulations**: It is the process of restricting the access to the abstracted code by wrapping the code

and data together. It can be done by providing access modifiers. Ex:

**Public**: least restrictive, all the class within the package have access

**Protected**: with in the class in the same package and sub classes have access

**Default**: All the class within the same package have access

**Private**: most restrictive, only the class have access. Can access from other class using getter

and setters

**Abstraction**: Process of hiding the implementation details and showing the function only. It is the way

of making the simple thing into complex. Can be achieve by abstract keyword or interface.

Abstract class cannot be instantiated. May have abstract and non-abstract method

**Interface**: Interface are the special type of class with the abstract methods. It declares the behavior for

classes to implement. All the variables are public static final. Can not be instantiated. Can

implement as many interfaces we want

**Marker Interface**: interface with no declare method and used to give information to the complier

**Functional Interface**: Interface with only one abstract method declared

**Arrays**: An array is a special type of object that act as the collection of same type of data. Array use

special memory in the heap, and it has fixed size.

String [] name = {abc, xyz}; or String [] name = new String [2]

**Varargs**: Variable argument list is a feature that allows you to pass an arbitrary number of values

and treat the data as single array. If used, varargs must be the last parameter and only one

varargs parameter can be used.

**Public void methodName (String name, varargs … name2){}**

**Scope of variables:**

**Instance:** Class level variables (lifetime), **Method:** Variables in method **Block level:**

**Static Keyword**: Static means the methods, or the variable belongs to the class instead of object. It can

invoke directly form class. Static method can be call from other static method only.

**Final Keyword**: Declares a resource (class, method, or field) as the last implementation, which cannot

be extended, overridden, or changed.

**Abstract Keyword**: Allows for the declaration of a class or method without implementation.

**Synchronized:** Limits the number of threads that can access a resource at a given time to 1.

**Wrapper Class**: Wrapper class allow us to treat primitive like objects. Ex: Integer, Float etc

Autoboxing:

Unboxing:

**Upcasting**: Instantiating by declaring parent class reference type

Animal mammal = new Dog ()

**Down casting**: Instantiating by declaring child class reference type

Dog dog = (Dog) (new Animal ())

**String**: array of characters in java. Strings are the immutable class in java. Strings are usually stored in

the string pool(heap) in the memory.

**String Manipulation**: String manipulation is the process of handling and analyzing the string. It is the

way of modifying and changing string data. Ex: compare (), concat (), equals (), Length (), etc.

**StringBuilder & String Buffer**: mutable alternatives to strings with methods that allow for manipulation (append, replace etc.)

StringBuilder is not synchronized (thread safe) -faster

String Buffer is synchronized (thread safe) – slower

**Generic in Java**: Generics are the methods, set of similar types. Generics allow Types, Integer, String or

user defined types to passed as parameter.

**Data Structure**: solution with interact or work with data

**Collection**: Implements Iterable. Collection is the interface that can be considered the “super” interface

from which list, set, queue come from. Work only with wrapper classes(objects).

**List**: Interface that implements collection and can hold duplicate values. List is an ordered list. We can add, change, and remove element anywhere in the list.

**Array List**: Dynamically sized array where element can be added or removed. Elements

can be accessed by index.

**Linked List**: Consists of a series of nodes which store data and a reference to the next

node

**Set**: Interface that extends collection and do not hold duplicate values. Elements are not in

ordered and cannot be find by index. Can add and remove elements.

**Queue**: Interface that extends collection and stored element at the end. Elements are ordered.

All the elements are inserted at the end and remove from beginning. Cannot find out by

the index.

**Map**: Interface that does not implements collection, leverage a key value pair system. They do not implement iterable interface, can not use for each loop directly on map

**Comparable & Comparator**: both are interface and can be used to sort the collection of elements

**Comparable**: Single sorting sequence (sort the collection based on Id, name etc.) This affects the

original class while sorting. CompareTo () method is used. Ex: collection.sort(List).

**Comparator**: Multiple sorting sequence (sort the collection based on multiple elements). Does

not affect the original class while sorting. Compare() method is used. Ex: collection.sort(List,

comparator)

**Functional Interfaces**: Functional interfaces are the interfaces that have only one abstract method. The

purpose of functional interface is to provide a specific function that can be applied easily to any

class.

Ex: **Runnable, Comparable, Comparator** etc.

**Lambdas**: The Lambda expression is one of the biggest new features of Java 8 and introduces aspects

important to the use of functional programming in java. Lambdas allow for the creation and

execution of a function or method without needing to create a dedicated space in memory

**Garbage Collection**: Garbage collection is the process of removing objects from the heap which have no

references to them. There is no way to explicitly force garbage collection in Java; however,

garbage collection can be requested using one of the following:

system.gc(), .runfinalize(), System.runFinalize()

**Finalize**: Finalize is used to perform clean up processing just before object is garbage collected. Finalize is the method, finally is the block and final is the keyword

**Reflections API:** The Reflections API enables Java code to discover information about the fields, methods

and constructors of classes which have been loaded into memory. The API will ‘reflect’ meta data about the fields, methods, and constructors to unveil their underlying information within security restrictions.

**Reflection API**: includes class, constructor, method, field, modifier, parameter

**Class Feature**: getMethod(), getConstructor(), getParameter(), getDeclaredFields() etc.

**Constructor Feature**: getName(), getModifiers(), getParameterName(), getParameterType() etc.

**Method Feature**: getName(), getParameterType(), getParameterName(), getReturnType() etc.

**Field feature:** getModifiers(), getType(), getByte(), getDouble() etc.

**Threads:** A thread is a subset of a process that is also an independent sequence of execution.

**Synchronization**: synchronized keyword which prevents more than one thread at a time to access a

resource

**Multi-Threading**: Multithreading extends the multitasking of applications by subdividing operations in a

single application into individual, parallel threads

**Creating Thread**:

Create a class the implements the runnable functional interface

Create a class that extends Thread

Create a Lambda\*

Override the run method

**States of Thread**:

**New**: Created a thread object but run has not been called

**Runnable**: Run has been called and been executing

**Waiting**: stops the execution of a thread until another thread is done executing

**Blocked**: waiting for another thread to be done accessing a resource on which a lock has

been placed on

**Terminated**: Everything has been executed

**POJO:** Plain Old Java Object. Any java object not bound to any specific restriction Used to define an entity

**Java Bean**: Special type of POJO. All java beans are POJO but all POJO are not Java beans. Should have

no-args constructor, all fields should be private, getters and setters, override toString, equals and

hascode method.

**Exception**: Exception are the problems that occurs during the execution of the java code. Or exception

is a condition that prevents the method for completing successfully. Exceptions **inherits from**

**Throwable** class.

**Types of Exception**:

**Checked Exception**: checked exceptions are checked by compiler. Can not compile code until

exception is handled

Ex: FileNotFoundException, ClassNotFoundException etc.

**Unchecked Exception**: Subclasses of Runtime Exception. Arises from coding error. Occurs in

Runtime

ArthmeticException, ArrayOutOfBoundException etc.

**Exception Handling**:

**Try Block**: used to execute the code or statement that may throw exception

**Catch Block**: Exception handler. Used to catch the exception

**Finally Block**: Always execute regardless of outcomes. Used to close resources of try block

**Try-With-Resources**: allows resources to close without needing finally block

**Throws**: used in method signature and used to throw exception. The exception will

handled in method where it is called

**Throwing Exception**: throw new Exception(“messages”)

**Custom Exception**: can create own exception by extending Exception or its sub class

**Error**: Errors are irrecoverable problems in java occur at compile time. Ex: out of memory

**Maven**: Maven is the project management and build automation tool which create artifact to be

deployed and manages the lifecycle of an application. Maven is used for properties and

dependency management also.

**lifecycle**: maven clean, maven compile, maven deploy

**dependency**: Frameworks or libraries written by other developers to be used in our project.

**Testing**: Testing is the process of making sure that the application is working as expected

**Positive Testing**: Making sure application handles correct inputs

**Negative Testing**: Making sure application handles incorrect inputs

**Regression Testing**: Making sure new features added to the application do not break existing

Features

**Testing Types (Pyramid)**: Unit Testing, Integration Testing, System Testing, User Acceptance Testing

**Unit Testing**: Unit testing is the testing of the individual software components in isolation from the rest

of the system. In simple word unit testing is the testing of the method of the code

**Junit**: Junit is an open-source java framework designed for the purpose of writing and running the unit

Testing. Junit allows for the creation of application code and run during testing.

**Junit Annotations**:

@BeforeClass: declare set up method that run once

@Before: run before every test are run

@Test: declare method as a test method

@After: run after every test are run

@AfterClass: declare tear-down method, run at the end of the test

**Junit Assertions**:

AssertEquals(), AssertNotEquals(), AssertTrue(), AssertFalse(), AssertNull(), AssertNotNull() etc.

**Test Driven Development**: The TDD process consists of writing unit tests first before the application code

has been written. Then, code can be written to make the test pass, and the process can be

completed for each piece of functionality required

**TDD Process**:

Write a unit test

Run the test: test will fail

Fixed the text by writing application code

Rerun the test

**Logging**: Logging refers to the act of recording granular events within an application, such as transactions

with a database, exceptions or debugging events

**Log4j2**: Log4j is a reliable, fast, and flexible logging framework for Java supported by Apache. It is

commonly used to record application events and write them to a file

**Main Components**:

**Logger**: logs the messages

**Appender**: publishes logs to destination(s)

**Layout**: formats logging information

**Configuration**: stores settings

**Filter**: used to filter logs that do not meet some threshold or are not required

**Logging Levels**:

**ALL:** all levels

**TRACE:** finer-grained informational events than DEBUG

**DEBUG:** designates informational events that are most useful to debug an application

**INFO:** informational messages that highlight the progress of the application at the coarse-

grained level

**WARN:** designates potentially harmful situations

**ERROR:** designates error events that might still allow the application to continue running

**FATAL:** severe error events that presumably lead the application to abort

**OFF:** highest possible level, intended to turn off logging