UNIT-1 [Around 10+5=15 marks]

Programming in Java

#JAVA Architecture: (How Java Works? / Working of JVM)

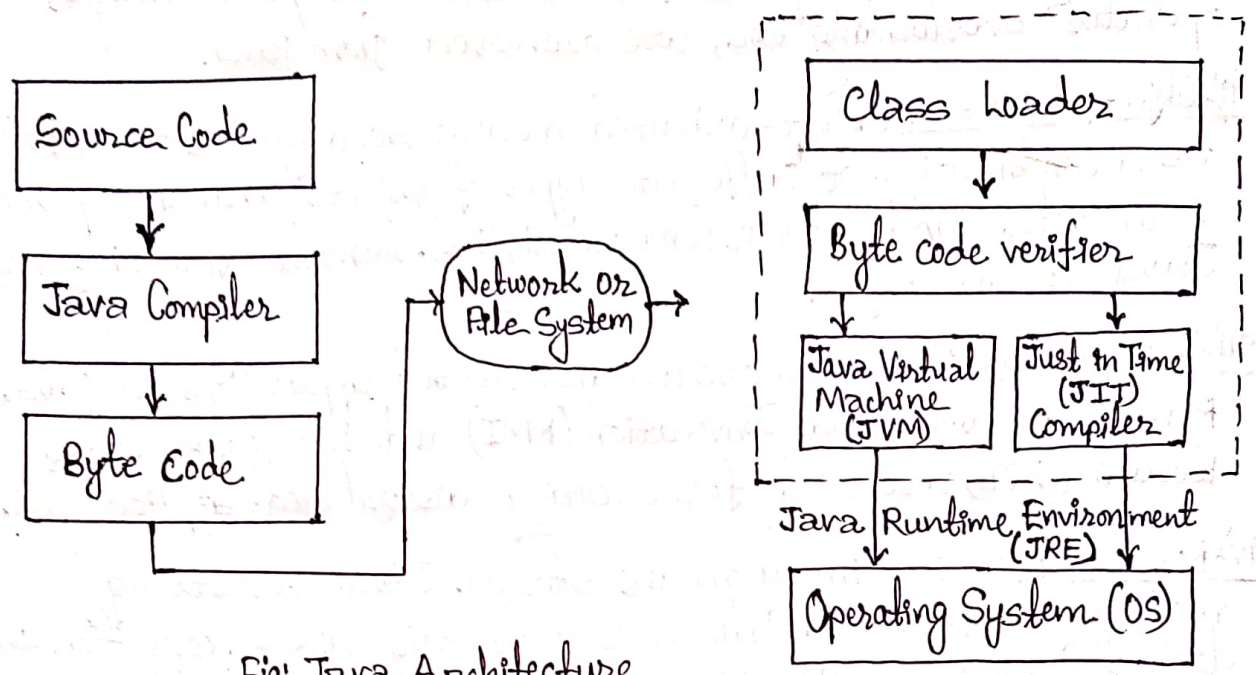


Fig: Java Architecture

→ Java source code (.java files) are compiled into byte code by java compiler.

→ Byte code is not executable code for target machine; rather its a object code for JVM. Byte code will be stored in class files (.class).

→ Class loader loads all the class files required to execute the program.

→ Byte code verifier checks the byte code and ensure the following:
- ↳ The Code follows JVM Specifications.
- ↳ There is no unauthorized access to memory
- ↳ The code doesn't cause any stack overflows.
- ↳ There is no illegal data conversions in code such as float to object refrences.

→ Once the code is verified, JVM will convert the byte code into machine code.

→ JIT compiler helps the program execution to happen faster. Code is cached by JIT Compiler and will be reused for the future needs.

→ Finally with the help of JRE, Operating System runs the code.

#Java Buzzwords: (or Features of Java)

i) **Simple:** Java language is simple because, its syntax is similar to C and C++ and hence easy to learn after C and C++. Again confusing and rarely used features such as explicit pointers, operator overloading etc, are removed from java.

ii) **Object Oriented:** Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour (characteristics). Almost everything in Java is an object.

iii) **Distributed:** We can create distributed applications in java. Remote ~~level~~ Method Invocation (RMI) and Enterprise Java Beans (EJB) are used for creating distributed applications.

iv) **Robust:** Robust simply means strong. Java uses strong memory management. There is automatic garbage collection and lack of pointers that avoids security ~~plo~~ problem. There is exception handling and type checking mechanism in Java.

v) **Secure:** Java programs run in a virtual machine that protects the underlying operating system from harm. Also, the absence of pointers in Java ensures that programs cannot gain access to memory locations using proper authorization.

vi) **Portable:** We can carry java byte code to any platform and execute there. The phrase "write once, run anywhere" is the major concept for the portablity. This is because byte code will run on any operating system for which there exists a compatible Virtual Machine.

vii) **High Performance:** Java is faster than traditional interpretation since byte code is close to native code. The performance gain is due to caching mechanism of JIT compiler.

viii) **Multithreaded:** We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory.

#Path and Classpath Variables in JAVA:

→ Path is a mediator between developer and operating system to inform binary file path. Classpath is a mediator between developer and compiler to inform library file path those are used in our source code.

→ The path points to the location of JRE. The classpath points to the classes we developed.

→ We don't need to set PATH and CLASSPATH to compile and run java program while using IDE like Eclipse. Environment variables are required to compile and run java program using CMD.

#Sample JAVA Program:

We can write this on any text editor like notepad, wordpad etc. to check

```
class Sample{
    public static void main(String args[]){
        System.out.println("Hello Java");
    }
}
```

Save this file as Sample.java

To Compile: javac Sample.java

To Execute: java Sample

write these commands Using CMD

Output: Hello Java

How to get input from user in Java:

Java Scanner class allows the user to take input from the console. It belongs to java.util package.

Syntax: `Scanner sc = new Scanner(System.in);`

Example:

```
import java.util.*;
class UserInputDemo{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a string: ");
        String str=sc.nextLine(); //reads string
        System.out.println("You have entered: "+str);
    }
}
```

#Arrays:

Array is a collection of similar types of elements under same variable name having contigious memory location. Java array is an object that contains elements of similar data type. There are two types of array:

i) Single Dimensional Array:

Syntax: data_type array_name[];
Example: int arr[5];
Program:

```
class testArray{
    public static void main(String args[]){
        int a[]=new int[5]; //declaration
        a[0]=10; //initialization
        a[1]=20;
        for(int i=0; i<a.length; i++){
            System.out.println(a[i]); //printing array
        }
    }
```

ii) Multidimensional Array:

Syntax: data_type array_name[][];
Example: int arr[3][3]; //i.e, 3 row 3 column.

#For Each Loop:

For each loop provides an alternative approach to traverse the array or collection in JAVA. The advantage is that it eliminates the possibility of bugs and makes code more readable. It ~~traves~~ traverses each element one by one. The drawback is it cannot traverse elements in reverse order and cannot have option to skip any element because it does not work on an index basis. But, it is recommended to use because it makes the code readable.

Syntax:
```
for (data_type variable_name : array|collection)
{
    //body of for-each loop
}
```

(array or collection)

Example:

```
class forEachDemo{
    public static void main(String args[]){
        int a[]={10,11,12,13};
        for(int i:a){
            System.out.println(i);
        }
    }
}
```

#Class and Object: [Imp]

Java is an object-oriented programming language. Everything in Java is associated with classes and objects, along with its attributes and methods. For Example: A bird is class. Parrot is an object of class bird. Characteristics like color, weight, height etc. are the attributes. How it flys, eats. etc. are its methods.

Creating class: Let we create a class called "Bird" as follows:

```
public class Bird{
    int weight=250;
    public void myMethod(){
        System.out.println("Weight is: "+weight+"pounds");
    }
```

Creating object: To create an object of class, we specify class name, followed by object name and use the keyword new. We can create any number of objects once we create a class. Then, the attributes and methods of class can be accessed using dot (.) operator. Let we create an object for above created class Bird:

```
Bird myObj=new Bird();
```

Now we can access our class attributes and methods of Bird using myObj as follows:

```
System.out.println(myObj.weight);
```

#Overloading:

In Java, we can define different methods with the same name but either with different number of parameters or with different type of parameters, which is called method overloading. This is one of the examples of polymorphism. In this case the return type of method does not matter.

Example:

```
class Overloading{
    public static void main(String args[]){
        System.out.println(sum(5,7));
        System.out.println(sum(5,7,2));
        System.out.println(sum(5.2f,7));
        System.out.println(sum(5.3,7.4));
    }
    static int sum(int a, int b){
        return a+b;
    }
    static int sum(int a, int b, int c){
        return a+b+c;
    }
    static float sum(float a, int b){
        return a+b;
    }
    static double sum(double a, double b){
        return a+b;
    }
}
```

#Access Privileges/Access Modifiers:

There are four types of Java access modifiers:

i) Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

ii) Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If we do not specify any access level, it will be the default.

iii) Protected: The access level of a protected modifier is within the package and outside the package through child class. If we do not make the child class, it cannot be accessed from outside the package.

iv) Public: The access level of a public modifier is everywhere. It can be accessed within the class, outside the class, within the package, and outside the package.

#Interface in Java: [Imp]

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There are mainly three reasons to use interface:

- → It is used to achieve abstraction.
- → By interface, we can support the functionality of multiple inheritance.
- → It can be used to achieve loose coupling.

An interface is declared by using interface keyword. All the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface interface_name{
    //declare constant fields
    //declare methods that abstract
    // by default.
}
```

#Inner Class:

Inner class refers to the class that is declared inside class or interface which were introduced, to sum up, same logically relatable classes. Following are certain advantages associated with inner classes:

- → Making code clean and readable.
- → Optimizing the code module.
- → Private methods of the outer class can be accessed, so bringing a new dimension and making it closer to real world.

We use mainly inner class, where we want certain operations to be performed, granting access to limited classes. There are basically four types of inner classes in java:

- → Nested Inner Classes
- → Method Local Inner Classes
- → Static Nested Classes
- → Anonymous Inner Classes.

Final and Static Modifiers:

Java provides a number of non-access modifiers to achieve many functionalities. Among them Final and Static modifiers are two main non-access modifiers.

→ The static modifier for creating class methods and variables.
→ The final modifier for finalizing the implementations of classes, methods and variables.

Static Modifier:

Static Variables: The static keyword is used to create variables that will exist independently of any instances created for class. Only one copy of the static variable exists regardless of the number of instances of class.

Static Methods: The static keyword is used to create methods that will exist independently of any instances created for class. Class variables and methods can be accessed using the class name followed by a dot and the name of variable or method.

Final Modifier:

Final Variables: A final variable can be explicitly initialized only once. A refrence variable declared final can never be reassigned to refer to an different object.

Final Methods: A final method cannot be overridden by any subclasses. The final modifier prevents a method from being modified in a subclass.

Packages: [Imp]

A java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form: built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing etc.

Advantages of Java Package:

→ Used to categorize classes and interfaces so that they can be easily maintained.
→ It provides access protection.
→ It removes naming collision.

How to compile java package:

Syntax: javac -d directory javafilename

Example: javac -d .Simple.java

where, -d specifies destination to put generated class file.

Simple example of java package:

```
//save as Sample.java
package mypack;
public class Sample{
    public static void main(String args[]){
        System.out.println("Welcome to package");
    }
}
```

#Inheritance: [Imp]

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. The idea behind inheritance is that we can create new classes based upon existing classes. When we inherit from an existing class, we can reuse methods and fields of the parent class. We can add new methods and fields in our current class also.

Why use inheritance:

→ For Method Overriding.
→ For Code Reusability.

Syntax:

```
class subClass-name extends superClass-name
{
    //methods and fields
}
```

#Overriding:

If subclass has the same method as declared in parent class, it is known as method overriding.

Why use overriding:

→ To provide the specific implementation of a method which is already provided by its superclass.
→ Used for runtime polymorphism.

Rules for Java Method Overriding:

→ The method must have the same name as in the parent class.
→ The method must have same parameter as in the parent class.
→ There must be parent-child (IS-A) relationship (i.e, inheritance).

Note: A static method cannot be overridden. Main method can not be overridden because it is also a static method.

#Handling Exceptions: [V.Imp]

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained. It handles runtime errors such as ClassNotFoundException, IOException, SQLException etc. There are 5 keywords which are used in handling exceptions in Java:

try: The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

catch: The "catch" block is used to handle the exception. It must be preceded by try block. It can be followed by finally block later.

finally: The "finally" block is used to execute the important code of the program. It is executed whether an exception handled or not.

throw: The "throw" keyword is used to throw an exception.

throws: The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example:

```
public class ExceptionExample{
    public static void main(String args[]){
        try{
            //code that may raise exception
            int data=100/0;
        }
        catch(ArithmeticException e){ System.out.println(e); }
        //rest code of the program
        System.out.println("rest of the code...");
    }
}
```

Common Scenarios of Java Exceptions:

1) If we divide any number by zero, there occurs ArithmeticException.

```
int a=50/0; //ArithmeticException
```

ii) If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;
System.out.println(s.length()); //NullPointer Exception.
```

iii) The wrong formatting of any value may occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s); //NumberFormatException.
```

iv) If we are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException.

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException.
```

⊛ Which exception should be declared?

Ans- There are three types of exceptions: checked, unchecked and error. The checked exception should only be declared. Since unchecked exception is under our control so we can correct our code and error exception is beyond our control e.g. we are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

#Differences between throws and throw in Java:

throw	throws
i) Java throw keyword is used to explicitly throw an exception.	i) Java throws keyword is used to declare an exception.
ii) Checked exception cannot be propagated using throw only.	ii) Checked exception can be propagated with throws.
iii) Throw is followed by an instance.	iii) Throws is followed by class.
iv) Throw is used within the method	iv) Throws is used with the method signature.
v) We cannot throw multiple exceptions.	v) We can declare multiple exceptions.

Creating Exception Class: (Or Custom exception Or User-defined exception)

Java platform provides a lot of exception classes that we can use. If we need an exception type that isn't represented by those in the Java platform, we can write one of our own. When we create custom exceptions in Java, we extend either Exception class or RuntimeException class.

Example:

```
class InvalidAgeException extends Exception{
    InvalidAgeException(String s){
        super(s);
    }
}

class TestCustomException1{
    static void validate(int age)throws InvalidAgeException{
    if(age<18)
        throw new InvalidAgeException("not valid");
    else
        System.out.println("welcome to vote");
    }

    public static void main(String args[]){
      try{
          validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}
      System.out.println("rest of the code...");
    }
}
```

If age given below 18 as input. If greater given then welcome to vote will be output

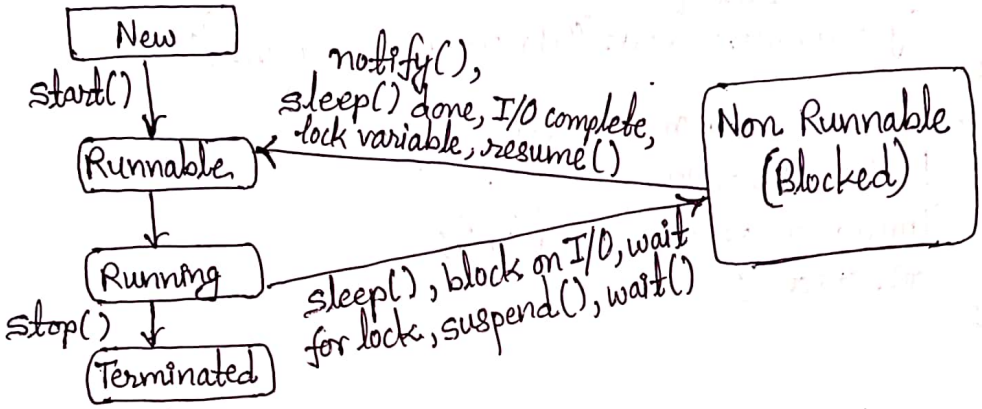
Output: Exception occured: InvalidAgeException: not valid
rest of the code...

Concurrency:

Concurrency is the ability to run several or multi programs or applications in parallel. The backbone of Java concurrency is threads (A lightweight process, which has its own files and stacks and can access the shared data from other threads in same process). Java Concurrency package covers concurrency, multithreading, and parallelism on the Java platform. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize CPU time.

#Thread States (Life cycle of a Thread):

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area. The life cycle of thread in java is controlled by JVM. The java thread states are as follows:



New: The thread is in new state if we create an instance of Thread class but before the invocation of start() method.

Runnable: The thread is in runnable state after invocation of start() method, but the thread schedular has not selected it to be the running thread.

Non-Runnable: This is the state when the thread is still alive, but is currently not eligible to run.

Terminated: A thread is in terminated or dead state when its run() method exists.

#Writing Multithreaded Programs (Creating Multithreaded Programs): [Imp]

There are two ways to create a thread:

1) **By extending Thread class:** Here we need to create a new class that extends the Thread class. The class should override the run() method which is the entry point for the new thread. Then we need to call start() method to start the execution of a thread.

Example:

```
class Multi extends Thread{
    public void run(){
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        Multi t1=new Multi();
        t1.start();
    }
}
```

2) By implementing Runnable interface: Here we need to give the definition of run() method. This run method is the entry point for the thread and thread will be alive till run method finishes its execution. Once the thread is created it will start running when start() method gets called.

Example:

```
Class Multi3 implements Runnable{
public void run(){
    System.out.println("thread is running...");
}
public static void main(String args[]){
    Multi3 m1=new Multi3();
    Thread t1=new Thread(m1);
    t1.start();
}
}
```

#Thread Properties:

We can get and set values of most of the thread properties. Major properties of Java threads are: Id, Name, Priority, currentThread, isAlive etc. Some of the thread methods are static and others are non-static. Static methods can be directly invoked through Thread class. But non-static methods need to be invoked by using object of Thread class. Following are important methods available in Thread class that can be used for getting and setting values of thread properties:

Public final void setName(String name): Changes name of the Thread object. There is getName() method for retriving the name.

Public final void setPriority(int priority): Sets the priority of this Thread object. The possible values are between 1 and 10. There is getPriority() method for retriving the priority.

Public final boolean isAlive(): Returns true if thread is alive.

Public static void yeild(): Causes the currently running thread to yield to any other threads of same priority.

Long getId(): This method returns the identifier of this thread.

Void join(): Waits for this thread to die.

#Thread Synchronization: [Imp]

Threads in a program are in the same process memory, and therefore have access to the same memory. The biggest problem of allowing multiple threads sharing the same data set is that one operation in one thread could collide with another operation in other threads on the same data. When this happens, the result is un-desirable.

Synchronization is the process of getting lock on the object so that only one thread can access that object. A running thread will enter either a blocked/waiting state when it tries to get the locked object. We can implement synchronization in two ways:

i) Method level synchronization: To acquire the object's lock, just call a method that has been modified with the synchronized keyword.

ii) Block level synchronization: We can use synchronized block when when we want to block or synchronize only some part of method. Using this we can lock third party objects.

Thread Priorities:

We can set priorities of java threads. Java provides setPriority() and getPriority() methods for setting and reading priorities of threads. Value of priority can range from 1-10. Default value of priority is 5. High priority threads get higher chances of execution from JVM. However, exact behaviour depends upon JVM.

Example:

```
class ThreadDemo extends Thread{
        String tname;
        ThreadDemo(String n){
                tname=n;
        }
}
```

```
public class ThreadPriority{
    public static void main(String args[]){
        ThreadDemo x = new ThreadDemo("T1");
        ThreadDemo y = new ThreadDemo("T2");
    //Displaying priority of threads or getting priority
        System.out.println("Priority of x: "+x.getPriority());
    //Setting priority of threads and displaying again
        x.setPriority(8);
        System.out.println("Priority of x: "+x.getPriority());
    }
}
```

Working with Files:

Java I/O (Input and Output) is used to process the input and produce the output based on input. Java uses the concept of stream to make I/O operation fast. The **java.io** package contains all the classes required for input and output operations. A stream is sequence of data. In java stream is composed of bytes. In java, 3 streams are created for us automatically. All these streams are attached with console.

- System.out → Standard output stream
- System.in → Standard input stream
- System.err → Standard error stream

#Byte Stream Classes:

Byte streams process data byte by byte (8 bits). For example FileInputStream is used to read from source and FileOutputStream to write to the destination. Following is an example:

```
import java.io.*;
public class CopyFile{
    public static void main(String args[]) throws IOException{
        FileInputStream in = null;
        FileOutputStream out = null;
```

```
        try{
            in=new FileInputStream ("input.txt");
            out=new FileOutputStream ("output.txt");
            int c;
            while ((c=in.read())!= -1) {
                    out.write(c);
            }
        } finally{ if (in != null){ in.close(); }
                   if(out != null) {out.close(); }
                }
    }
}
```

Question may be asked like this. Write character stream class example in ans:
Or, Write a simple java program that reads data from one file and writes to another file.

#Character Stream Classes:

In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. For example FileReader and FileWriter are character streams used to read from source and write to destination.

Example:

```
import java.io.*;
public class CopyFile{
    public static void main(String args[]) throws IOException{
        FileReader in=null;
        FileWriter out=null;
        try{
            in= new FileReader ("input.txt");
            out=new FileWriter ("output.txt");
            int c;
            while ((c= in.read()) !=-1) {
                out.write(c);
            }
        } finally { if (in !=null){ in.close();}
                    if (out!=null){out.close();}
                  }
    }
}
```

Random Access File Class:

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructor	Description
RandomAccessFile (File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile (String name, String mode)	Creates a random access file stream to read from and optionally to write to, a file with the specified name.

Reading and Writing Objects:

Java object Serialization is an API that allows us to serialize Java objects. Serialization is a process to convert objects into a Writable byte stream. Once converted into a byte-stream, these objects can be written to a file. The reverse process of this is called de-serialization. A Java object is Serializable if it's class implement the java.io.Serializable interface. ObjectInputStream and ObjectOutputStream classes are used to read/write java objects.

#Multiple Inheritance with Interface: V.V. Imp.

In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Java does not support multiple inheritances with classes. In java, we can achieve multiple inheritances only through Interfaces. An interface is declared by using interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface.

Multiple inheritance can be achieved with interfaces, because the class can implement multiple interfaces. To implement multiple interfaces, seperate them with a comma. In the following example class Animal is derived from interface AnimalEat and AnimalTravel.

```
interface AnimalEat{
        void eat();
}
interface AnimalTravel{
        void travel();
}

class Animal implements AnimalEat, AnimalTravel{
    public void eat(){
            System.out.println("Animal is eating");
    }
    public void travel(){
            System.out.println("Animal is traveling");
    }
}
```