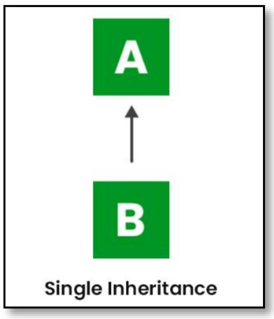
1. Explain the types of inheritance with diagram.

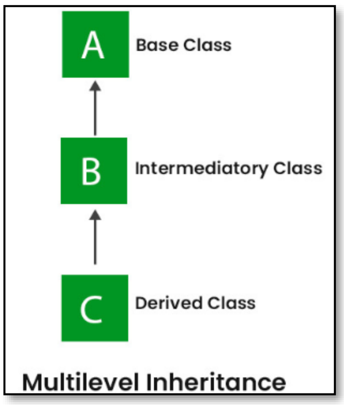
Ans: Inheritance is an important pillar of OOP (Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features (fields and methods) of another class.

Types:

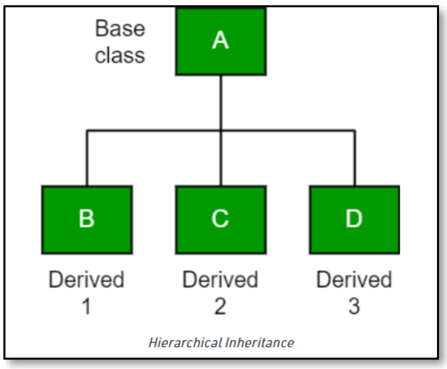
i. Single Inheritance: In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.



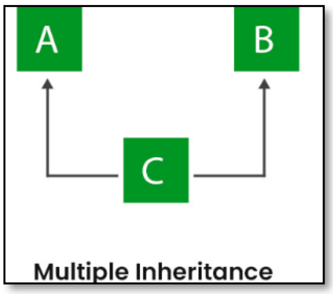
ii. Multilevel Inheritance: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.



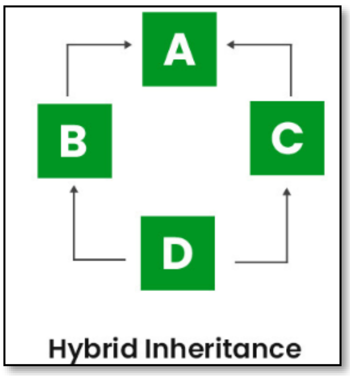
iii. Hierarchical Inheritance: In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.



iv. Multiple Inheritance (Through Interfaces): In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritances with classes. In java, we can achieve multiple inheritances only through Interfaces. In the image below, Class C is derived from interface A and B.



v. Hybrid Inheritance (Through Interfaces): It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



2. How are exceptions handled in Java. Explain with syntax.

Ans: Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Suppose If any code block is prone to exception than we surround it with try catch block. In try block the code which is prone to exception is placed and in catch block we handle the exception thrown in try block. Finally is a block that runs despite the try block throws error or not.

Default Exception Handling: Whenever inside a method, if an exception has occurred, the method creates an Object known as an Exception Object and hands it off to the run-time system(JVM). The exception object contains the name and description of the exception and the current state of the program where the exception has occurred. Creating the Exception Object and handling it in the run-time system is called throwing an Exception. There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of the methods is called Call Stack. Now the following procedure will happen.

Syntax:

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}finally {
   // The finally block always executes.
}
```

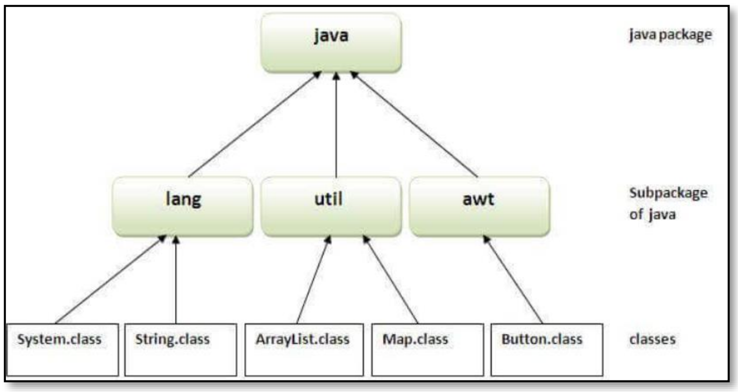
3. Explain the concept of packages in Java. Write a program to demonstrate the same.

Packages:

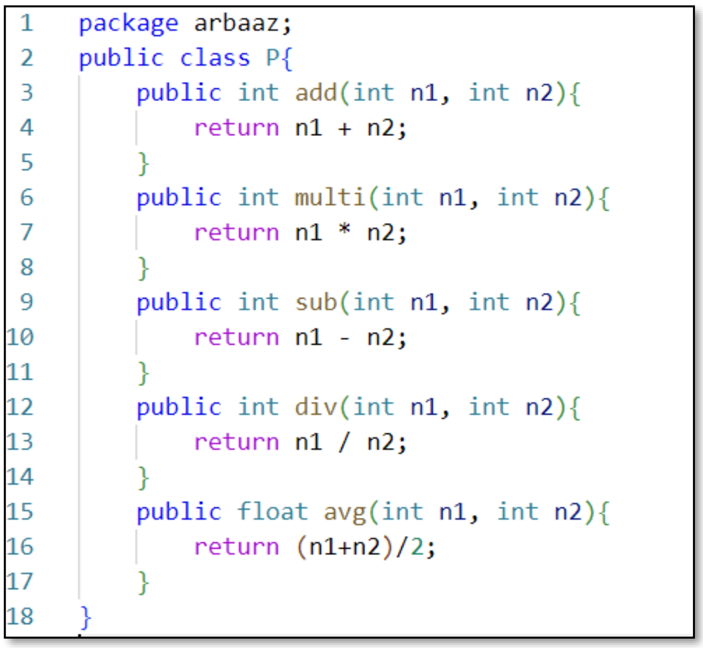
i. A java package is a group of similar types of classes, interfaces and sub-packages.
ii. Package in java can be categorized in two form, built-in package and user-defined package.
iii. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
iv. Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

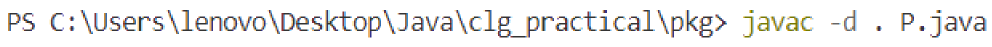
i. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
ii. Java package provides access protection.
iii. Java package removes naming collision.



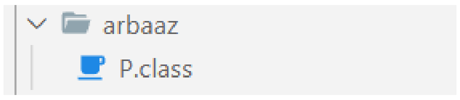
Demonstration:


```
package arbaaz;
public class P{
    public int add(int n1, int n2){
        return n1 + n2;
    }
    public int multi(int n1, int n2){
        return n1 * n2;
    }
    public int sub(int n1, int n2){
        return n1 - n2;
    }
    public int div(int n1, int n2){
        return n1 / n2;
    }
    public float avg(int n1, int n2){
        return (n1+n2)/2;
    }
}
```

Creating a package arbaaz with single class P.



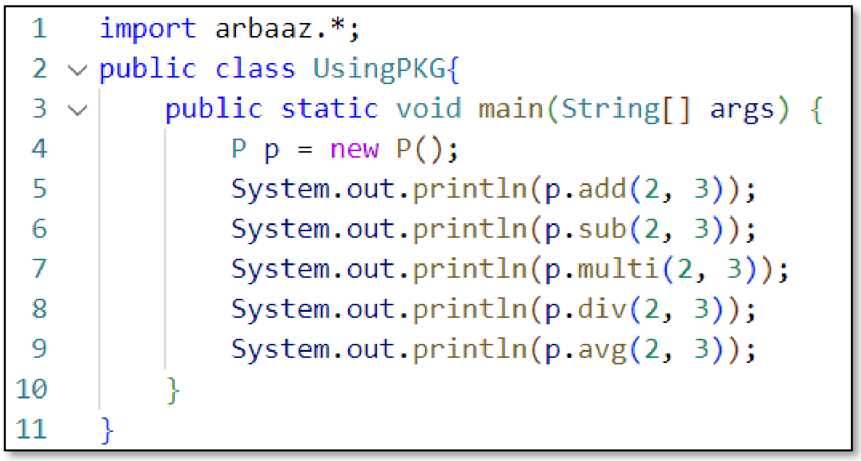
```
PS C:\Users\lenovo\Desktop\Java\clg_practical\pkg> javac -d . P.java
```



Package is created successfully.

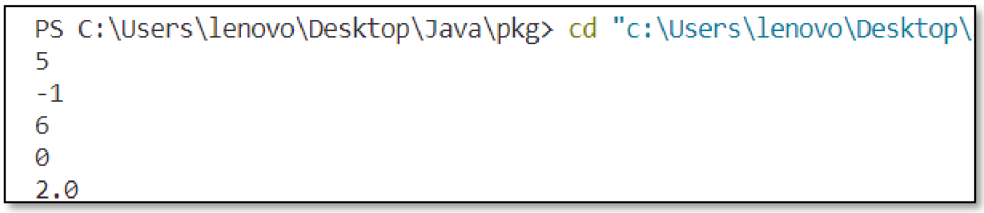
Now import and used the created package.

Code:



```java
import arbaaz.*;
public class UsingPKG{
    public static void main(String[] args) {
        P p = new P();
        System.out.println(p.add(2, 3));
        System.out.println(p.sub(2, 3));
        System.out.println(p.multi(2, 3));
        System.out.println(p.div(2, 3));
        System.out.println(p.avg(2, 3));
    }
}
```

Output:



```
PS C:\Users\lenovo\Desktop\Java\pkg> cd "c:\Users\lenovo\Desktop\
5
-1
6
0
2.0
```

4. **Explain multithreading in java.**

Multithreading:

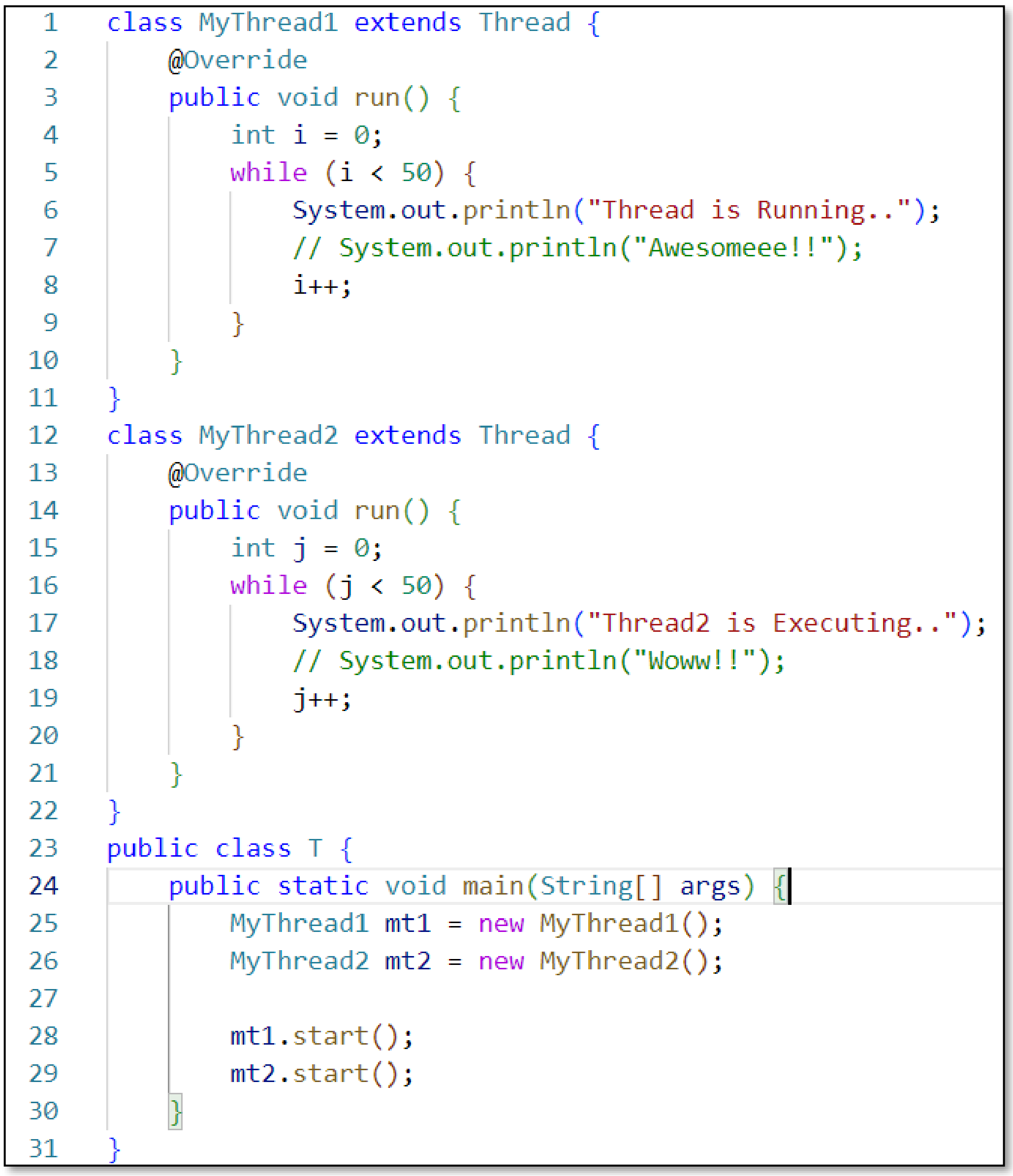
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

i. Extending the Thread class:

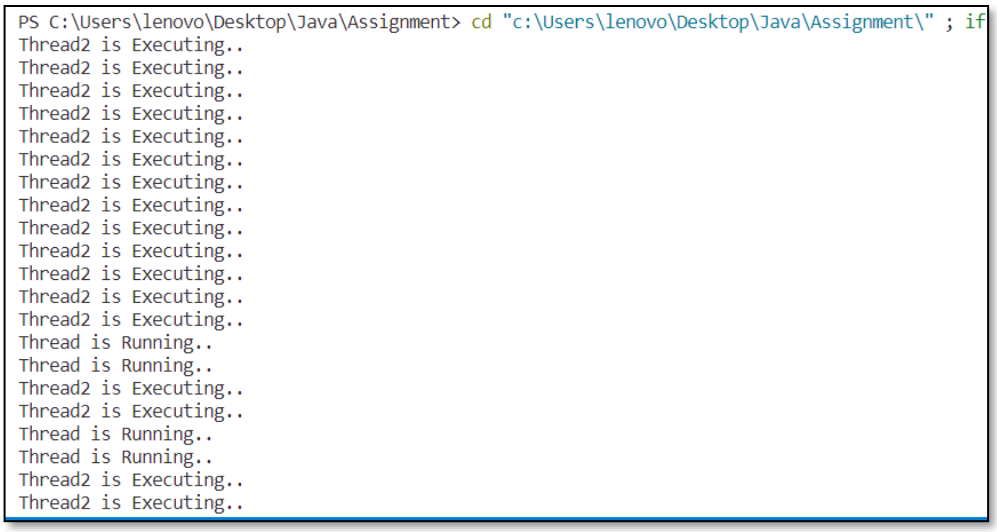
We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Code:



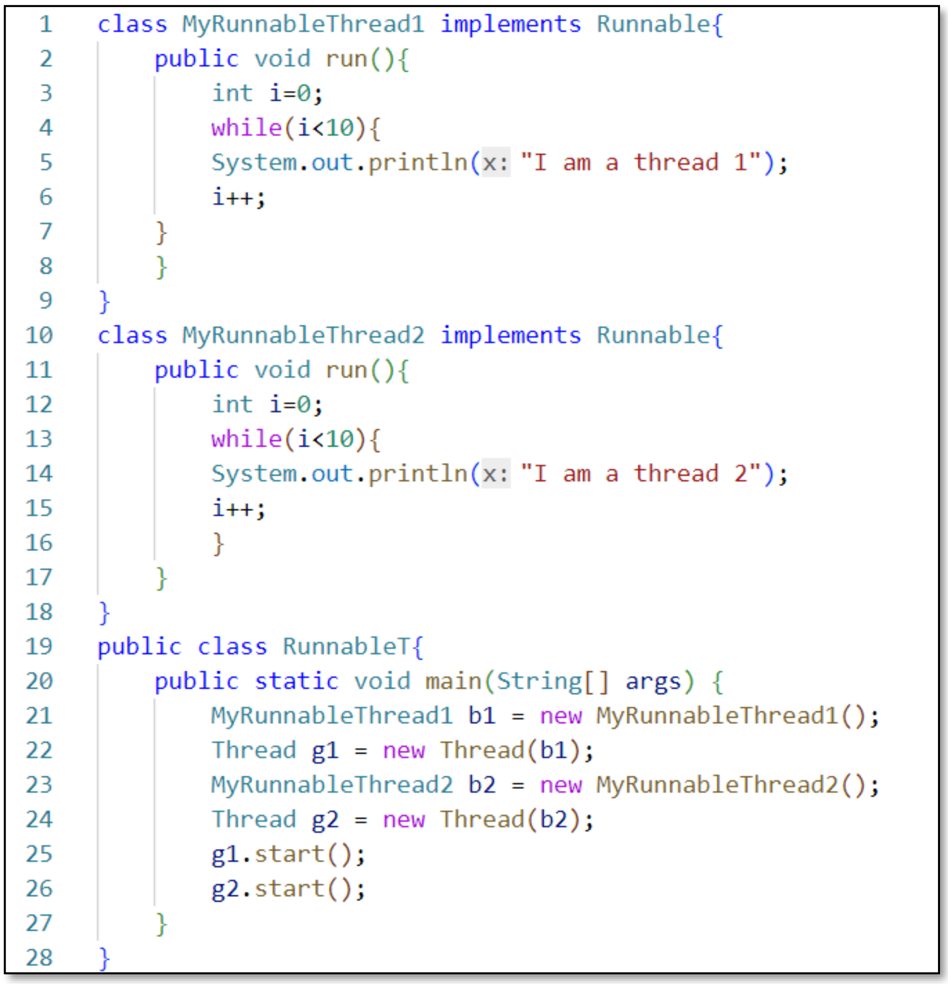
```java
class MyThread1 extends Thread {
    @Override
    public void run() {
        int i = 0;
        while (i < 50) {
            System.out.println("Thread is Running..");
            // System.out.println("Awesomeee!!");
            i++;
        }
    }
}
class MyThread2 extends Thread {
    @Override
    public void run() {
        int j = 0;
        while (j < 50) {
            System.out.println("Thread2 is Executing..");
            // System.out.println("Woww!!");
            j++;
        }
    }
}
public class T {
    public static void main(String[] args) {
        MyThread1 mt1 = new MyThread1();
        MyThread2 mt2 = new MyThread2();

        mt1.start();
        mt2.start();
    }
}
```

Output:



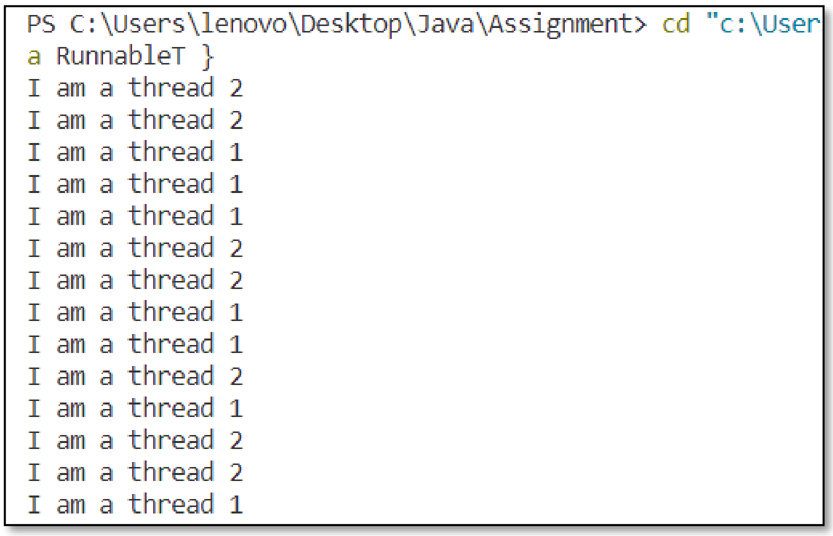
```
PS C:\Users\lenovo\Desktop\Java\Assignment> cd "c:\Users\lenovo\Desktop\Java\Assignment\" ; if
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread2 is Executing..
Thread is Running..
Thread is Running..
Thread2 is Executing..
Thread2 is Executing..
Thread is Running..
Thread is Running..
Thread2 is Executing..
Thread2 is Executing..
```

ii. Implementing the Runnable Interface:
We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.
Code:



```java
class MyRunnableThread1 implements Runnable{
    public void run(){
        int i=0;
        while(i<10){
        System.out.println(x: "I am a thread 1");
        i++;
    }
    }
}
class MyRunnableThread2 implements Runnable{
    public void run(){
        int i=0;
        while(i<10){
        System.out.println(x: "I am a thread 2");
        i++;
        }
    }
}
public class RunnableT{
    public static void main(String[] args) {
        MyRunnableThread1 b1 = new MyRunnableThread1();
        Thread g1 = new Thread(b1);
        MyRunnableThread2 b2 = new MyRunnableThread2();
        Thread g2 = new Thread(b2);
        g1.start();
        g2.start();
    }
}
```

Output:



```
PS C:\Users\lenovo\Desktop\Java\Assignment> cd "c:\User
a RunnableT }
I am a thread 2
I am a thread 2
I am a thread 1
I am a thread 1
I am a thread 1
I am a thread 2
I am a thread 2
I am a thread 1
I am a thread 1
I am a thread 2
I am a thread 1
I am a thread 2
I am a thread 2
I am a thread 1
```