Java Programs

1. Conditional statement

```
import java.util.Scanner;

public class Condition {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = 30;
        if(a>b && a>c){
            System.out.println(+a+" is Biggest");
        }
        else if (b>c) {
            System.out.println(+b+" is Biggest");
        }
        else{
            System.out.println(+c+" is Biggest");
        }
        int d;
        System.out.println("Enter your choice");
        Scanner s = new Scanner(System.in);
        d = s.nextInt();
        switch (d) {
            case 1:
                System.out.println("Choice 1");
                break;

            case 2:
                System.out.println("Choice 2");
                break;

            default:
                break;
        }
        for(int i=0;i<10;i++){
            if(i%2==0){
                continue;
            }
            System.out.println(i);
        }
    }
}
```

2.Looping Statement

```
public class Loop {
    public static void main(String[] args) {
        int n=5;
        int sum=0,i=0;
        for(i=0;i<n;i++){
            sum=sum+i;
        }
        System.out.println("Sum of array elements from for is: "+sum);
        while (i<n) {
            sum=sum+i;
        }
        System.out.println("Sum of array elements from while is: "+sum);
        do{
            sum=sum+i;
            i++;
        }while(i<n);
        System.out.println("Sum of array elements from do while is: "+sum);
        int a[]={1,2,3,4,5};
        for(int ele:a){
            sum=sum+ele;
        }
        System.out.println("Sum of array elements from for each is: "+sum);
    }
}
```

3.Constructor and method overloading

```
public class Con {
    int a,b;
    public Con(int a,int b){
        this.a=a;
        this.b=b;
        System.out.println("Inside two parameter constructor");
        System.out.println("a="+a+" and b="+b);
    }
    Con(int a){
        System.out.println("Inside single parameter constructor");
        System.out.println("a="+a);
    }
    void met(int a){
        a*=a;
        System.out.println("Inside one parameter method");
        System.out.println("a="+a);
    }
    int met(int a,int b){
        return a*b;
    }
    public static void main(String[] args) {
        Con c = new Con(10);
        Con b = new Con(20, 30);
        c.met(15);
        int s=c.met(20, 30);
        System.out.println("s="+s);
    }
}
```

4. Inheritance

```
class Person {
    Person(int id,String name){
        System.out.println("Id is "+id+"\nName "+name);
    }
}
public class InnerPerson extends Person{
    InnerPerson(int id,String name,String clg,String address){
        super(id, name);
        System.out.println("College is "+clg+"\nAddress is "+address);
    }
}
class  Main {
    public static void main(String[] args) {
        InnerPerson i = new InnerPerson(100,"Pannu","NIE", "Koppa");
    }
}
```

5. Hierarchial inheritance with dynamic method dispatch

```
package PracticeLab;


class Hierar {
    void mat(){
        System.out.println("Parent");
    }
}


class InnerHierar extends Hierar {
    void mat(){
        System.out.println("First child");
    }
}


class  Outer extends Hierar {
    void mat(){
        System.out.println("Second child");
    }
}


class Mine {
    public static void main(String[] args) {
        Hierar h;
        h = new InnerHierar();
        h.mat();
        h = new Outer();
        h.mat();
    }
}
```

6.Implements interface class methods

```
interface Inter {
    void show();
}

class InnerInter implements Inter{
     public void show(){
        System.out.println("Interphase inside InnerInter");
     }
}

public class InnerInter_1 {
    public static void main(String[] args) {
        InnerInter i = new InnerInter();
        i.show();
    }
}
```

7. Creation of Abstruct class.

```
abstract class InnerMain {
    abstract void show();
}

class InnerMain_1 extends InnerMain{
    void show(){
        System.out.println("Abstruct");
    }
}

public class Main {
    public static void main(String[] args) {
        InnerMain_1 i = new InnerMain_1();
        i.show();
    }
}
```

8. Use of Throw,Throws try,catch,finally.

```
public class MyExcept {
    public static void check(int age) throws IllegalArgumentException{
        if(age<18){
            throw new IllegalArgumentException("You are under age");
        }
        else{
            System.out.println("Eligible");
        }
    }
    public static void main(String[] args) {
        try{
            check(15);
        }
        catch(IllegalArgumentException e){
            System.out.println(e);
        }
        finally{
                System.out.println("Verification completed");
            }
        }
    }
```

9. Custom exception.

```
import java.util.Scanner;

class Myexception extends Exception{
    Myexception(){
        System.out.println("Age is less than 18");
    }
}

public class Custom {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter age");
        int age = s.nextInt();
        if(age>18){
            System.out.println("Eligible");
        }
        else{
            try{
                throw new Myexception();
            }
            catch(Myexception e){

            }
        }
    }
}
```

10.Creating Extending thread.

```
class DisplayMessage1 extends Thread {

    private String message;
    public DisplayMessage1(String message)
    {
        this.message = message;
    }
    public void run()
    {
            System.out.println(message);
    }
}
public class ThreadDemo_using_Thread_Class
{
    public static void main(String[] args)
    {
        System.out.println("Creating the hello thread...");
        DisplayMessage1 hello = new DisplayMessage1("Hello");
        System.out.println("Starting the hello thread...");
        hello.start();

        System.out.println("Creating the goodbye thread...");
        DisplayMessage1 bye = new DisplayMessage1("Goodbye");
        System.out.println("Starting the goodbye thread...");
        bye.start();
    }
}
```

11.Creating Implementing thread.

```
class DisplayMessage implements Runnable {

    private String message;
    public DisplayMessage(String message)
    {
        this.message = message;
    }
    public void run()
    {
        try
        {
            System.out.println(message);
            Thread.sleep(1000);
        }
        catch(InterruptedException e)
            {

            }
    }

}
public class ThreadDemo_using_runnable
{
    public static void main(String[] args) {

        System.out.println("Creating the hello thread...");
        DisplayMessage hello = new DisplayMessage("Hello");
        Thread thread1 = new Thread(hello);
        System.out.println("Starting the hello thread...\n");
        thread1.start();

        System.out.println("Creating the goodbye thread...");
        DisplayMessage bye = new DisplayMessage("Goodbye");
        Thread thread2 = new Thread(bye);

        System.out.println("Starting the goodbye thread...");
        thread2.start();
    }

}
```

12.Enumeration.

```
enum fruit{
    APPLE,BALL,CAT;
}

public class Enum_Show {
    public static void main(String[] args) {
        for (fruit f : fruit.values()) {
            System.out.println(f);
        }
        System.out.println(fruit.valueOf("APPLE"));
    }
}
```