01

Expt. No. Page No. Date

★ Aim: To Print Hello World

★ Program

```
public class main {
    public static void main (String[] args)
    {
        System.out.println ("Hello World!");
        DateTimeFormatter dtf = DateTimeFormatter
        .ofPattern ("dd/MM/yyyy HH:mm:ss");

        LocalDateTime now = LocalDateTime.now();
        System.out.println (dtf.format(now));

    }
}
```

02

Expt. No. | Page No. | Date

Aim:

3.1 Implementation of all Java Literals.
3.2 Implementation of Dynamic initialization of variables.
3.3 Implementation of the scope and lifetime of variables.

★ Program

3.1
```
public class JavaLiterals {
public static void main (string [] args) {

int a = 101 ;
int b = 0100 ;
int c = 0xFACE ;
int d = 0b1111 ;

System.out.println (a);
System.outprintln (b);
System.out.println (c);
System.out.println (d);
}
}
```


Teacher's Signature:.....

Expt. No. Page No. Date

3.2
```
public class DynamicInitialization {
public static void main(String[] args) {

    double a = 3.0, b = 4.0f;

    double c = Math.sqrt(a*a + b*b);
    double d = c * a;

    System.out.println("Hypotenuse is " + c);
    System.out.println("Value of d" + d);
}
}
```

3.3
```
public class Scope
{  int num=20;
   void m1()
   { int num=30;
     System.out.println("Local number:" + num);
     System.out.println("Global number:" + this.num);
   }
   Public static void main(String[] args)
   { Scope st = new Scope();
     st.m1();
   }}
```

GOOD WRITE® A Mark of Quality.....

Teacher's Signature:.....

05
Expt. No.

Page No.

Date

★ Aim: Write a Java Package to show Dynamic Polymorphism and interfaces.

★ Program

```
class A {
    int a, b;
    void print() {
        System.out.println(a + "\t" + b);
    }
    A() {
        a = b = -1;
    }
    A(int a, int b) {
        this.a = a;
        this.b = b;
    }
    A(int a) {
        this.a = this.b = a;
    }
    int greater() {
        return a > b ? a : b;
    }
}
```

GOOD WRITE®
A Mark of Quality.....

Teacher's Signature:.....

Expt. No. ..... Page No. ..... Date .....

```
class B extends A {
        int c;
        void print() {
                System.out.println(c+"\t");
                super.print();
        }
        B() {
                c = super.a = super.b = -1;
        }
        B(int c){
                super.a = super.b = this.c = c;
        }
        B(int a, int b, int c) {
                this.c = c;
                super.a = a;
                super.b = b;
        }
}

Class Polymorphism {

Public static void main (string [] args) {
```

```
A ob1 = new A();
A ob2 = new A(2);
A ob3 = new A(3,4);

B ob4 = new B();
B ob5 = new B(5);
B ob6 = new B(6,7,8);

ob1.print();

ob6.print();

System.out.println("After overwriting");

ob1 = ob6;
ob1.print();

}
}
```

Expt. No. ..... Page No. ..... Date .....

Exp 7

★ Aim: To implement types of Inheritance
a) Single Inheritance
b) Multilevel Inheritance
c) Multiple Inheritance
d) Hierichal Inheritance

★ Program

A) Single Inheritance

```
class A {
        A() {
            System.out.println("Constructor of class A");
        }
    }

class B extends A {
        B() {
            super();
            System.out.println("Constructor of class B"); } }
```

Expt. No. Page No. Date

```
public class Single {
        public static void main(String args[]){
            B obj = new B();
        }
    }
```

B) Multilevel Inheritance

```
class A {
        A(){
            System.out.println("constructor of class A");
        }
    }

class B extends A {
        B() { super();
            System.out.println("constructor of class
            B");
        }
    }

class C extends B {
        C() { super();
            System.out.println("Constructor of
            class C"); } }
```


Teacher's Signature:.....

```
public class Multilevel {
    public static void main (String args[]) {
        C obj = new C();
    }
}
```

c) Multiple Inheritance

```
class A {
    A() {
        System.out.println("constructor of class A");
    }
}

interface B {
    void print();
}

class C extends A implements B {
    C() {
        super();
```

```
        System.out.println ("Constructor of class C");
    }

@override
public static void ~~main~~ print () {

        System.out.println ("Inside Interface B");
    }
}

public class Multiple {
        public static void main (String args[])
        {
            C obj = new C();
            obj.print();
        }
```

D) Hierichal Inheritance

```
class A {
        A() {
            System.out.println ("Constructor of class A");
        }
    }
```

Expt. No. ..... Page No. ..... Date .....

```
class B extends A {
        B() {
            super ();
            system.out.println (" Constructor of
            class B");
        }
    }

class C extends A {
        C() {
            super ();
            system.out.println (" Constructor of
            class C");
        }
    }

public class Hierichal {
        public static void main (String[]args Ct)

    C ob1 = new C();
    B ob2 = new B();
```

Expt. No. ..... Page No. ..... Date .....

Exp 9

☆ Aim: To implement Exception handling and catch throw

(A) Program - Exception Handling

```
public class Exception_Handling {

    public static void main (String[] args) throws
    Exception {

        try {
            int a,b;
        BufferedReader in = new BufferedReader (new
        InputStreamReader (System.in));

        a = Integer.parseInt (in.readLine());
        b = Integer.parseInt (in.readLine());
        }

        catch (NumberFormatException e2) {
```

```
            System.out.println (ex.getMessage () + " is not a
            numeric value");
            System.exit(0);
        }
    }
}
```

B) Try and Catch

```
class Trycatch {
        public static void main (String [] args)
        {
            int i=0;
            string greetings [] ={
                "Hello Ramit!",
                "Hello Java!",
                "Hello World!" };

            while ( i<4) {
                try (
                    System.out.println (Gre
                    eting [i]);
```

Expt. No. ..... Page No. ..... Date .....

```
} catch (Exception e) {
        System.out.println(e.toString());
        System.out.println("Resetting index value");
    }
    finally {
        System.out.println("i:"+i);
        i++;
    }
  }
 }
}
```


Teacher's Signature:.....

10
Expt. No.

Page No.

Date

Exp 10

★ Aim: To implement Producer Consumer problem using Multithreading

★ Program

```
class Q {
        int n;
        boolean valueSet = false;
        synchronized int get() {
        while (! valueSet)
        try {
            wait();
            }
        catch (Exception e) {
         System.out.println("Interrupted Exception caught");
        }
        System.out.println("got: " + n);
        valueSet = False;
        notify();
        return n;
        }
```


Teacher's Signature:.....

Expt. No. ..... Page No. ..... Date .....

```
synchronized void put (int n) {
    while (valueSet) {
        try{
            wait(); }
        catch (Exception e) {
            System.out.println ("Caught Exc");
            this.n = n;
            valueSet = true;
            System.out.println ("Put: " + n);
            notify();
        }
    }

class Producer implements Runnable {
    Q q;
    Producer (Q q) {
        this.q = q;
        new Thread(this, "producer").start();
    }
```

Expt. No. ..... Page No. ..... Date .....

```
synchronized void put (int n) {
while (valueSet) {
                    try{
                        wait(); }
catch (Exception e) {
System.out.println ("Caught Exc");

this.n = n;
valueSet = true;
System.out.println ("Put: " + n);
notify();
    }
}

class Producer implements Runnable {

Q q;
Producer (Q q) {
                this.q = q;
                new Thread(this, "producer").start();
            }
```

GOOD WRITE® A Mark of Quality.....

Teacher's Signature:.....

Expt. No. ..... Page No. ..... Date .....

```
class Pro_Cons {
    public static void main(String args[])
    {
        Q q = new Q();
        new Producer(q);
        new Consumer(q);
    }
}
```


Teacher's Signature:.....