1. Math Operations Package - mathops/Operations.java

```
package mathops;

public class Operations {
    public static int add(int a, int b) { return a + b; }
    public static int subtract(int a, int b) { return a - b; }
    public static int multiply(int a, int b) { return a * b; }
    public static int divide(int a, int b) {
        if (b == 0) throw new ArithmeticException("Division by zero");
        return a / b;
    }
}
```

Main Class to use Math Operations - Main.java

```
import java.util.Scanner;
import mathops.Operations;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();

        System.out.println("Addition: " + Operations.add(a, b));
        System.out.println("Subtraction: " + Operations.subtract(a, b));
        System.out.println("Multiplication: " + Operations.multiply(a, b));
        try {
            System.out.println("Division: " + Operations.divide(a, b));
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

2. Factorial with Exception Handling - Factorial.java

```
public class Factorial {
    static long[] factArray = new long[21];

    public static long factorial(int x) {
        if (x < 0) throw new IllegalArgumentException("Value of x must be positive");
            if (x >= factArray.length) throw new IllegalArgumentException("Result will
overflow.");
        if (factArray[x] != 0) return factArray[x];
        if (x == 0 || x == 1) return factArray[x] = 1;
        return factArray[x] = x * factorial(x - 1);
    }

    public static void main(String[] args) {
        try {
            System.out.println("20! = " + factorial(20));
            System.out.println("21! = " + factorial(21));
        } catch (Exception e) {
```

```
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

3. Manual Exception Throwing - ThrowExample.java

```
public class ThrowExample {
    public static void main(String[] args) {
        int age = -5;
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative");
        } else {
            System.out.println("Age is valid: " + age);
        }
    }
}
```

4. Ping PONG using Multithreading - PingPong.java

```
class Ping extends Thread {
    public void run() {
        while (true) {
            System.out.println("Ping");
            try { Thread.sleep(1000); } catch (InterruptedException e) {}
        }
    }
}

class Pong extends Thread {
    public void run() {
        while (true) {
            System.out.println("PONG");
            try { Thread.sleep(1000); } catch (InterruptedException e) {}
        }
    }
}

public class PingPong {
    public static void main(String[] args) {
        new Ping().start();
        new Pong().start();
    }
}
```

5. Multithreaded Multiplication Table - MultiplicationTable.java

```
class TableThread extends Thread {
    int number;

    TableThread(int number) {
        this.number = number;
    }

    public void run() {
```

```
        for (int i = 1; i <= 10; i++) {
            System.out.println(number + " x " + i + " = " + (number * i));
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        }
    }
}

public class MultiplicationTable {
    public static void main(String[] args) {
        TableThread t1 = new TableThread(2);
        TableThread t2 = new TableThread(3);
        t1.start();
        t2.start();
    }
}
```

6. Producer-Consumer Problem - ProducerConsumer.java

```
class Q {
    int item;
    boolean valueSet = false;

    synchronized void put(int item) {
        while (valueSet) {
            try { wait(); } catch (InterruptedException e) {}
        }
        this.item = item;
        valueSet = true;
        System.out.println("Produced: " + item);
        notify();
    }

    synchronized void get() {
        while (!valueSet) {
            try { wait(); } catch (InterruptedException e) {}
        }
        System.out.println("Consumed: " + item);
        valueSet = false;
        notify();
    }
}

class Producer extends Thread {
    Q q;

    Producer(Q q) {
        this.q = q;
    }

    public void run() {
        int i = 0;
        while (true) {
            q.put(i++);
            try { Thread.sleep(1000); } catch (InterruptedException e) {}
```

```
        }
    }
}

class Consumer extends Thread {
    Q q;

    Consumer(Q q) {
        this.q = q;
    }

    public void run() {
        while (true) {
            q.get();
            try { Thread.sleep(1000); } catch (InterruptedException e) {}
        }
    }
}

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