**EXCEPTION HANDLING**

**public** **class** Testtrycatch2{

**public** **static** **void** main(String args[]){

**try**{

**int** data=50/0;

   }**catch**(ArithmeticException e){System.out.println(e);}

   System.out.println("rest of the code...");

}

}

OUTPUT:

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

**MULTI CATCH BLOCK**

**public** **class** TestMultipleCatchBlock{

**public** **static** **void** main(String args[]){

**try**{

**int** a[]=**new** **int**[5];

    a[5]=30/0;

   }

**catch**(ArithmeticException e){System.out.println("task1 is completed");}

**catch**(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

**catch**(Exception e){System.out.println("common task completed");}

   System.out.println("rest of the code...");

 }

}

Output:

task1 completed

rest of the code...

**NESTED TRY**

**class** Excep6{

**public** **static** **void** main(String args[]){

**try**{

**try**{

     System.out.println("going to divide");

**int** b =39/0;

    }**catch**(ArithmeticException e){System.out.println(e);}

**try**{

**int** a[]=**new** **int**[5];

    a[5]=4;

    }**catch**(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);

  }**catch**(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");

 }

}

OUTPUT:

going to divide

java.lang.ArithmeticException: / by zero

java.lang.ArrayIndexOutOfBoundsException: 5

other statement

normal flow..

**USAGE OF JAVA FINALLY**

**CASE I:**

**class** TestFinallyBlock{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/5;

   System.out.println(data);

  }

**catch**(NullPointerException e){System.out.println(e);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

OUTPUT:

finally block is always executed

rest of the code...

**CASE II:**

**class** TestFinallyBlock1{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/0;

   System.out.println(data);

  }

**catch**(NullPointerException e){System.out.println(e);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

OUTPUT:

finally block is always executed

Exception in thread main java.lang.ArithmeticException:/ by zero

**CASE III:**

**public** **class** TestFinallyBlock2{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/0;

   System.out.println(data);

  }

**catch**(ArithmeticException e){System.out.println(e);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

OUTPUT:

Exception in thread main java.lang.ArithmeticException:/ by zero

finally block is always executed

rest of the code...

**THROW**

**public** **class** TestThrow1{

**static** **void** validate(**int** age){

**if**(age<18)

**throw** **new** ArithmeticException("not valid");

**else**

      System.out.println("welcome to vote");

+s   }

**public** **static** **void** main(String args[]){

      validate(13);

      System.out.println("rest of the code...");

  }

}

OUTPUT:

Exception in thread main java.lang.ArithmeticException:not valid

**THROWS**

**import** java.io.IOException;

**class** Testthrows1{

**void** m()**throws** IOException{

**throw** **new** IOException("device error");//checked exception

  }

**void** n()**throws** IOException{

    m();

  }

**void** p(){

**try**{

    n();

   }**catch**(Exception e){System.out.println("exception handled");}

  }

**public** **static** **void** main(String args[]){

   Testthrows1 obj=**new** Testthrows1();

   obj.p();

   System.out.println("normal flow...");

  }

}

OUTPUT:

exception handled

normal flow...

**CUSTOM EXCEPTION**

**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...");

  }

}

OUTPUT:

Exception occured: InvalidAgeException:not valid

rest of the code...

**JAVA MEMBER INNER CLASS**

**class** TestMemberOuter1{

**private** **int** data=30;

**class** Inner{

**void** msg(){System.out.println("data is "+data);}

 }

**public** **static** **void** main(String args[]){

  TestMemberOuter1 obj=**new** TestMemberOuter1();

  TestMemberOuter1.Inner in=obj.**new** Inner();

  in.msg();

 }

}

OUTPUT:

data is 30

**JAVA ANONYMOUS INNER CLASS**

**abstract** **class** Person{

**abstract** **void** eat();

}

**class** TestAnonymousInner{

**public** **static** **void** main(String args[]){

  Person p=**new** Person(){

**void** eat(){System.out.println("nice fruits");}

  };

  p.eat();

 }

}

Output:

nice fruits

**JAVA LOCAL INNER CLASS**

**public** **class** localInner1{

**private** **int** data=30;//instance variable

**void** display(){

**class** Local{

**void** msg(){System.out.println(data);}

  }

  Local l=**new** Local();

  l.msg();

 }

**public** **static** **void** main(String args[]){

  localInner1 obj=**new** localInner1();

  obj.display();

 }

}

OUTPUT:

30

**JAVA STATIC NESTED CLASS**

**class** TestOuter1{

**static** **int** data=30;

**static** **class** Inner{

**void** msg(){System.out.println("data is "+data);}

  }

**public** **static** **void** main(String args[]){

  TestOuter1.Inner obj=**new** TestOuter1.Inner();

  obj.msg();

  }

}

OUTPUT:

data is 30

**THREAD**-**BY** **THREAD CLASS**

**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();

 }

}

Output:

thread is running...

**THREAD-BY RUNNBLE INTERFACE**

**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();

 }

}

Output:

thread is running...

**SLEEP METHOD**

**class** TestSleepMethod1 **extends** Thread{

**public** **void** run(){

**for**(**int** i=1;i<5;i++){

**try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}

    System.out.println(i);

  }

 }

**public** **static** **void** main(String args[]){

  TestSleepMethod1 t1=**new** TestSleepMethod1();

  TestSleepMethod1 t2=**new** TestSleepMethod1();

  t1.start();

  t2.start();

 }

}

Output:

1

1

2

2

3

3

4

4

**JOIN() METHOD**

**class** TestJoinMethod1 **extends** Thread{

**public** **void** run(){

**for**(**int** i=1;i<=5;i++){

**try**{

    Thread.sleep(500);

   }**catch**(Exception e){System.out.println(e);}

  System.out.println(i);

  }

 }

**public** **static** **void** main(String args[]){

 TestJoinMethod1 t1=**new** TestJoinMethod1();

 TestJoinMethod1 t2=**new** TestJoinMethod1();

 TestJoinMethod1 t3=**new** TestJoinMethod1();

 t1.start();

**try**{

  t1.join();

 }**catch**(Exception e){System.out.println(e);}

 t2.start();

 t3.start();

 }

}

Output:

1

2

3

4

5

1

1

2

2

3

3

4

4

5

5

**NAMING A THREAD**

**class** TestMultiNaming1 **extends** Thread{

**public** **void** run(){

   System.out.println("running...");

  }

**public** **static** **void** main(String args[]){

  TestMultiNaming1 t1=**new** TestMultiNaming1();

  TestMultiNaming1 t2=**new** TestMultiNaming1();

  System.out.println("Name of t1:"+t1.getName());

  System.out.println("Name of t2:"+t2.getName());

  t1.start();

  t2.start();

  t1.setName("SHAZIA");

  System.out.println("After changing name of t1:"+t1.getName());

 }

}

OUTPUT:

Name of t1:Thread-0

Name of t2:Thread-1

id of t1:8

running...

After changeling name of t1:SHAZIA

running...

**CURRENT THREAD()**

**class** TestMultiNaming2 **extends** Thread{

**public** **void** run(){

  System.out.println(Thread.currentThread().getName());

 }

**public** **static** **void** main(String args[]){

  TestMultiNaming2 t1=**new** TestMultiNaming2();

  TestMultiNaming2 t2=**new** TestMultiNaming2();

  t1.start();

  t2.start();

 }

}

OUTPUT:

Thread-0

Thread-1

**THREAD PRIORITY**

**class** TestMultiPriority1 **extends** Thread{

**public** **void** run(){

   System.out.println("running thread name is:"+Thread.currentThread().getName());

   System.out.println("running thread priority is:"+Thread.currentThread().getPriority())

 }

**public** **static** **void** main(String args[]){

  TestMultiPriority1 m1=**new** TestMultiPriority1();

  TestMultiPriority1 m2=**new** TestMultiPriority1();

  m1.setPriority(Thread.MIN\_PRIORITY);

  m2.setPriority(Thread.MAX\_PRIORITY);

  m1.start();

  m2.start();

 }

}

OUTPUT:

running thread name is:Thread-0

running thread priority is:10

running thread name is:Thread-1

running thread priority is:1

**DAEMON THREAD**

**public** **class** TestDaemonThread1 **extends** Thread{

**public** **void** run(){

**if**(Thread.currentThread().isDaemon()){//checking for daemon thread

   System.out.println("daemon thread work");

  }

**else**{

  System.out.println("user thread work");

 }

 }

**public** **static** **void** main(String[] args){

  TestDaemonThread1 t1=**new** TestDaemonThread1();//creating thread

  TestDaemonThread1 t2=**new** TestDaemonThread1();

  TestDaemonThread1 t3=**new** TestDaemonThread1();

  t1.setDaemon(**true**);//now t1 is daemon thread

  t1.start();//starting threads

  t2.start();

  t3.start();

 }

}

OUTPUT:

daemon thread work

user thread work

user thread work

**JAVA THREAD POOL**

|  |
| --- |
| **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  **class** WorkerThread **implements** Runnable {  **private** String message;  **public** WorkerThread(String s){  **this**.message=s;      }  **public** **void** run() {  System.out.println(Thread.currentThread().getName()+" (Start) message = "message);  processmessage();  System.out.println(Thread.currentThread().getName()+" (End)");  }  **private** **void** processmessage() {  **try**  {  Thread.sleep(2000);  }  **catch** (InterruptedException e)  {  e.printStackTrace();  }  }  }  **public** **class** TestThreadPool {  **public** **static** **void** main(String[] args) {  ExecutorService executor = Executors.newFixedThreadPool(5);  **for** (**int** i = 0; i < 10; i++) {   Runnable worker = **new** WorkerThread("" + i);  executor.execute(worker);            }          executor.shutdown();  **while** (!executor.isTerminated()) {   }            System.out.println("Finished all threads");      }   }  Output:  pool-1-thread-1 (Start) message = 0  pool-1-thread-2 (Start) message = 1  pool-1-thread-3 (Start) message = 2  pool-1-thread-5 (Start) message = 4  pool-1-thread-4 (Start) message = 3  pool-1-thread-2 (End)  pool-1-thread-2 (Start) message = 5  pool-1-thread-1 (End)  pool-1-thread-1 (Start) message = 6  pool-1-thread-3 (End)  pool-1-thread-3 (Start) message = 7  pool-1-thread-4 (End)  pool-1-thread-4 (Start) message = 8  pool-1-thread-5 (End)  pool-1-thread-5 (Start) message = 9  pool-1-thread-2 (End)  pool-1-thread-1 (End)  pool-1-thread-4 (End)  pool-1-thread-3 (End)  pool-1-thread-5 (End)  Finished all threads |

**SHUTDOWN HOOK**

**class** MyThread **extends** Thread{

**public** **void** run(){

        System.out.println("shut down hook task completed..");

    }

}

**public** **class** TestShutdown1{

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

Runtime r=Runtime.getRuntime();

r.addShutdownHook(**new** MyThread());

System.out.println("Now main sleeping... press ctrl+c to exit");

**try**{Thread.sleep(3000);}**catch** (Exception e) {}

}

}

OUTPUT:

Now main sleeping... press ctrl+c to exit

shut down hook task completed..

**SINGLE TASK BY MULTIPLE THREADS**

**class** TestMultitasking1 **extends** Thread{

**public** **void** run(){

   System.out.println("task one");

 }

**public** **static** **void** main(String args[]){

  TestMultitasking1 t1=**new** TestMultitasking1();

  TestMultitasking1 t2=**new** TestMultitasking1();

  TestMultitasking1 t3=**new** TestMultitasking1();

  t1.start();

  t2.start();

  t3.start();

 }

}

OUTPUT:

task one

task one

task one

**TWO TASKS BY TWO THREADS**

**class** Simple1 **extends** Thread{

**public** **void** run(){

   System.out.println("task one");

 }

}

**class** Simple2 **extends** Thread{

**public** **void** run(){

   System.out.println("task two");

 }

}

**class** TestMultitasking3{

**public** **static** **void** main(String args[]){

  Simple1 t1=**new** Simple1();

  Simple2 t2=**new** Simple2();

  t1.start();

  t2.start();

 }

}

Output:

task one

task two

**GARBAGE COLLECTION**

**public** **class** TestGarbage1{

**public** **void** finalize(){System.out.println("object is garbage collected");}

**public** **static** **void** main(String args[]){

  TestGarbage1 s1=**new** TestGarbage1();

  TestGarbage1 s2=**new** TestGarbage1();

  s1=**null**;

  s2=**null**;

  System.gc();

 }

}

OUTPUT:

object is garbage collected

object is garbage collected

**EXEC()**

**public** **class** Runtime1{

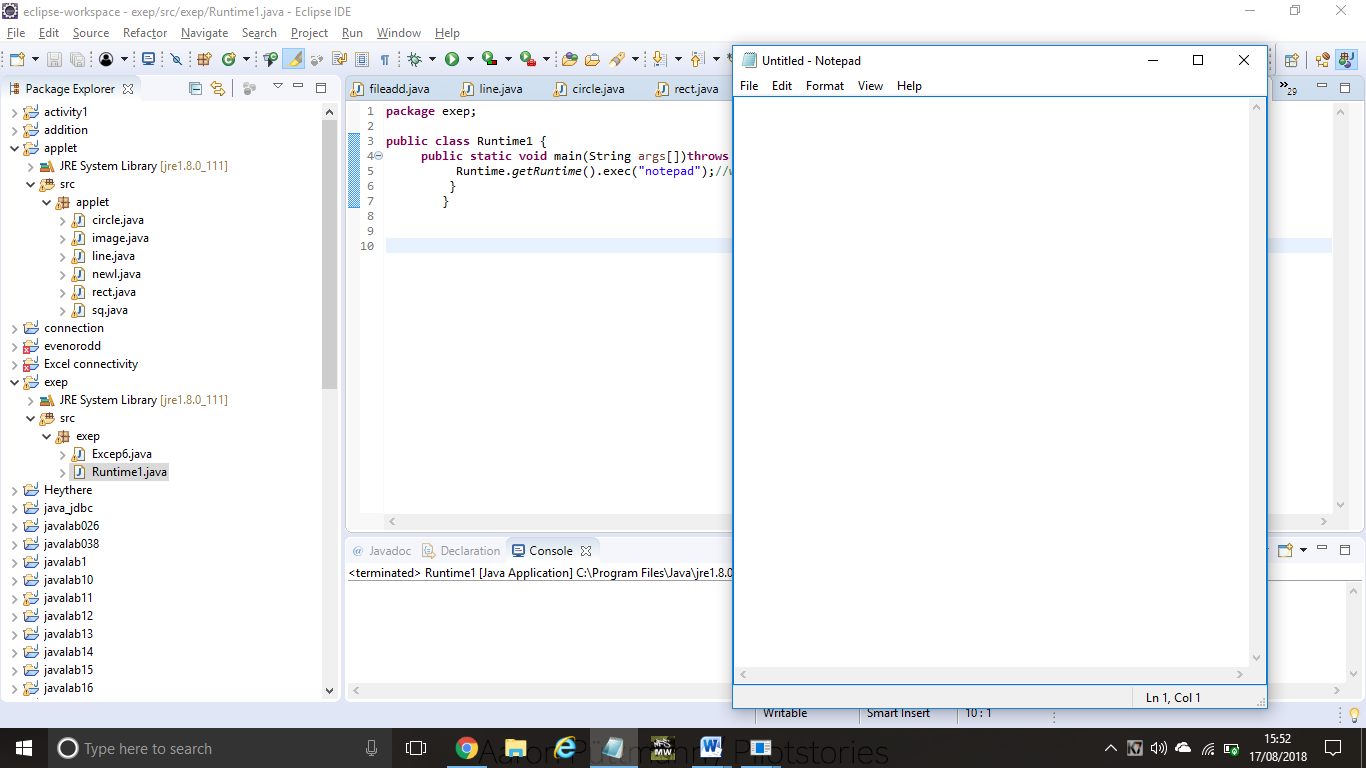
**public** **static** **void** main(String args[])**throws** Exception{

  Runtime.getRuntime().exec("notepad");//will open a new notepad

 }

}

OUTPUT:



**freeMemory() and totalMemeory()**

**public** **class** MemoryTest{

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

  Runtime r=Runtime.getRuntime();

  System.out.println("Total Memory: "+r.totalMemory());

  System.out.println("Free Memory: "+r.freeMemory());

**for**(**int** i=0;i<10000;i++){

**new** MemoryTest();

  }

  System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());

  System.gc();

  System.out.println("After gc(), Free Memory: "+r.freeMemory());

 }

}

OUTPUT:

Total Memory: 126877696

Free Memory: 125535480

After creating 10000 instance, Free Memory: 124827744

After gc(), Free Memory: 125884392

**SYNCHRONIZED BLOCK**

**class** Table{

**void** printTable(**int** n){

**synchronized**(**this**){//synchronized block

**for**(**int** i=1;i<=5;i++){

      System.out.println(n\*i);

**try**{

       Thread.sleep(400);

      }**catch**(Exception e){System.out.println(e);}

     }

   }

0 }//end of the method

}

**class** MyThread1 **extends** Thread{

Table t;

MyThread1(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(5);

}

}

**class** MyThread2 **extends** Thread{

Table t;

MyThread2(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(100);

}

}

**public** **class** TestSynchronizedBlock1{

**public** **static** **void** main(String args[]){

Table obj = **new** Table();//only one object

MyThread1 t1=**new** MyThread1(obj);

MyThread2 t2=**new** MyThread2(obj);

t1.start();

t2.start();

}

}

OUTPUT:

5

10

15

20

25

100

200

300

400

500

**STATIC SYNCHRONIZATION**

**class** Table{

**synchronized** **static** **void** printTable(**int** n){

**for**(**int** i=1;i<=10;i++){

     System.out.println(n\*i);

**try**{

       Thread.sleep(400);

     }**catch**(Exception e){}

   }

 }

}

**class** MyThread1 **extends** Thread{

**public** **void** run(){

Table.printTable(1);

}

}

**class** MyThread2 **extends** Thread{

**public** **void** run(){

Table.printTable(10);

}

}

**class** MyThread3 **extends** Thread{

**public** **void** run(){

Table.printTable(100);

}

}

**class** MyThread4 **extends** Thread{

**public** **void** run(){

Table.printTable(1000);

}

}

**public** **class** TestSynchronization4{

**public** **static** **void** main(String t[]){

MyThread1 t1=**new** MyThread1();

MyThread2 t2=**new** MyThread2();

MyThread3 t3=**new** MyThread3();

MyThread4 t4=**new** MyThread4();

t1.start();

t2.start();

t3.start();

t4.start();

}

}

OUTPUT:

1

2

3

4

5

6

7

8

9

10

10

20

30

40

50

60

70

80

90

100

100

200

300

400

500

600

700

800

900

1000

1000

2000

3000

4000

5000

6000

7000

8000

9000

10000

**DEADLOCK IN JAVA**

**public** **class** TestDeadlockExample1 {

**public** **static** **void** main(String[] args) {

**final** String resource1 = "SHAZIA FARHEEN";

**final** String resource2 = "SAIMA SEEMA";

    // t1 tries to lock resource1 then resource2

    Thread t1 = **new** Thread() {

**public** **void** run() {

**synchronized** (resource1) {

           System.out.println("Thread 1: locked resource 1");

**try** { Thread.sleep(100);} **catch** (Exception e) {}

**synchronized** (resource2) {

            System.out.println("Thread 1: locked resource 2");

           }

         }

      }

    };

    // t2 tries to lock resource2 then resource1

    Thread t2 = **new** Thread() {

**public** **void** run() {

**synchronized** (resource2) {

          System.out.println("Thread 2: locked resource 2");

**try** { Thread.sleep(100);} **catch** (Exception e) {}

**synchronized** (resource1) {

            System.out.println("Thread 2: locked resource 1");

          }

        }

      }

    };

    t1.start();

    t2.start();

  }

}

OUTPUT:

Thread 1: locked resource 1

Thread 2: locked resource 2

**INTER THREAD COMMUNICATION IN JAVA**

**class** Customer{

**int** amount=10000;

**synchronized** **void** withdraw(**int** amount){

System.out.println("going to withdraw...");

**if**(**this**.amount<amount){

System.out.println("Less balance; waiting for deposit...");

**try**{wait();}**catch**(Exception e){}

}

**this**.amount-=amount;

System.out.println("withdraw completed...");

}

**synchronized** **void** deposit(**int** amount){

System.out.println("going to deposit...");

**this**.amount+=amount;

System.out.println("deposit completed... ");

notify();

}

}

**class** Test{

**public** **static** **void** main(String args[]){

**final** Customer c=**new** Customer();

**new** Thread(){

**public** **void** run(){c.withdraw(15000);}

}.start();

**new** Thread(){

**public** **void** run(){c.deposit(10000);}

}.start();

  }}

 Output:

going to withdraw...

Less balance; waiting for deposit...

going to deposit...

deposit completed...

withdraw completed

**INTERUPPTING A THREAD**

**public** **class** TestInterruptingThread4 **extends** Thread{

**public** **void** run(){

**for**(**int** i=1;i<=2;i++){

**if**(Thread.interrupted()){

System.out.println("code for interrupted thread");

}

**else**{

System.out.println("code for normal thread");

}

}//end of for loop

}

**public** **static** **void** main(String args[]){

TestInterruptingThread4 t1=**new** TestInterruptingThread4();

TestInterruptingThread4 t2=**new** TestInterruptingThread4();

t1.start();

t1.interrupt();

t2.start();

}

}

Output:

Code for interrupted thread

code for normal thread

code for normal thread

code for normal thread