Wrapper class overloading

Operators

23 apr examples

=============== Exception Handling in Java==========================  
 - Exception is an event that occurs during the execution due to wrong  
   input and it leads to abnormal termination of program execution.  
     
 - Whenever an exception is raised, JVM creates an object of type  
   of Exception or sub type of Exception and then returns the exception  
   related message to the console.  
     
 - Exception is a predefined class available in java.  
 - Exception is a sub class of Throwable.  
 - Throwable is the super-most class for each Error and Exception and their  
   sub types.  
 - Error is a predefined class.  
 - Error has sub classes such as:  
       - AssertionError  
       - ThreadDeath  
       - VirtualMachineError  
       - LinkageError  
   \*AssertionError thrown to indicate that an assertion has failed.  
   \* It was introduced in java 1.4 version.  
   
  \* ThreadDeath is thrown in the victim thread when the (deprecated) Thread.stop()  
    method is invoked.  
    -It was introduced in Java 1.0 version.  
    -An application should catch instances of this class only if it  
     must clean up after being terminated asynchronously.If  
     ThreadDeath is caught by a method, it is important that it should  
     be re-thrown so that the thread actually dies.  
       
  \*VirtualMachineError object is thrown to indicate that the Java Virtual Machine  
   is broken or has run out of resources necessary for it to continue operating.  
   - It has multiple sub classes:  
        a. InternalError:  
          Its object thrown is to indicate some unexpected internal error that has occurred in  
            the Java Virtual Machine.  
        b. StackOverflowError: It is thrown when a stack overflow occurs because of an application  
            recursing too deeply.  
        c. UnknownError:Thrown when an unknown but serious exception has occurred in the  
           Java Virtual Machine.  
        d.OutOfMemoryError:It is thrown when the JVM cannot allocate an object  
           because it is out of memory, and no more memory could be made  
           available by the garbage collector.  
             
    \* LinkageError: Its  a subclass of Error.  
          Its subclasses indicate that a class has some dependency on another class;  
          however, the latter class incompatibly changed after the compilation  
          of the former class.  
          It has sub classes such as:  
               a. ClassFormatError  
               b. IncompatibleClassChangeError  
               c. ClassCirculatoryError  
               d. ExceptionInitializerError  
               e. VerifyError  
               f. UnsatisfiedLinkError  
               g. NoClassDefoundError  
              Some of the miscellaneous Error as sub class of IncompatibleClassChangeError  
               are:   NoSuchFieldError,NoSuchMethodError,InstanitationError,  
               AbstractMethodError etc.  
    Note:  
    NoSuchFieldError is thrown if an application tries to access or modify a specified  
    field of an object, and that object no longer has that field.  
    Generally, this error is caught by the compiler; this error can  
    only occur at run time if the definition of a class has  
    incompatibly changed.  
     
    NoSuchMethodError: It is thrown if an application tries to call a specified method of a  
    class (either static or instance), and that class no longer has a  
    definition of that method.  
    Normally, this error is caught by the compiler; this error can  
    only occur at run time if the definition of a class has  
    incompatibly changed.  
         
    IllegalAccessError: It  is thrown if an application attempts to access or modify  
    a field, or to call a method that it does not have access to.  
     
     
  ---Exception class in java:  
   - Exception is a predefined class available in java.lang package.  
   - It is the sub class of Throwable.  
   - Exception and its subclasses indicates conditions that are reasonable for an  
     application might to catch.  
    - Exception has in general two categories of sub classes:  
         - Sub classes of RuntimeException  
         - Direct Sub classes of Exception  
       
      -Sub classes of Exception:  
          a. RunTimeException  
          b. ClassNotFoundException  
          c. NoSuchFieldException  
          d. NoSuchMethodException  
          e. InterruptedException  
          f. AccessorException  
          g. ActivationException  
          h. AlreadyBoundException  
          i. AnalyzerException  
          j. AWTEXception  
          k. BackingStoreException  
          l. BrokenBarrierException  
          m. CertificateException  
          n. CGIClientException  
          o. CompilerException  
          p. DataFomatException  
          q. DispatchException  
          r. DestroyFailedException  
          s. DuplicateException  
          t. ExceutionException  
          u. FilterException  
          v. IllegalClassFormatException  
          w. InvalidApplicationException  
          x. InvalidValueException  
          y. IOException  
          z. JAXBException  
          -------------  
          a. LambdaConversionException  
          b. LocaleSyntaxException  
          c. NamingException  
          d. NoSuchEventException  
          e. NoSuchProducerException  
          f. NoSuchRecordingException  
          g. NotBoundException  
          h. ParseException  
          i. PathException  
          j.ReflectiveCopyException  
          k. ReflectiveOperationException  
          l. ScriptException  
          m. ServerNotActiveException  
          n. ServerSideException  
          o. SOAPException  
          p. SQLException  
          q. TargetLostException  
          r. TimeOutException  
          s. WindowsException  
          t. XMLParseException

Ques - 1

----------

Find the statement which will cause compile time error , comment it and find the answer and draw the execution flow.

class A {

void m1() {

System.out.println("A m1");

}

void m2() {

System.out.println("A m2");

m1();

}

}

class B extends A {

void m1() {

System.out.println("B m1");

}

void m3() {

System.out.println("B m3");

m1();

super.m2();

}

}

class C extends B {

void m2() {

System.out.println("C m2");

//m4();

}

}

class D extends C {

void m1() {

System.out.println("D m1");

}

void m2() {

System.out.println("D m2");

}

void m4() {

System.out.println("D m4");

}

public static void main(String[] args) {

D d = new D();

d.m1();

d.m2();

d.m3();

d.m4();

}

}

Ques - 2

----------

Find the statement which will cause compile time error , comment it and find the answer and draw the execution flow.

class A {

static void m1() {

System.out.println("A m1");

}

static void m2() {

System.out.println("A m2");

m1();

}

}

class B extends A {

static void m1() {

System.out.println("B m1");

}

void m3() {

System.out.println("B m3");

m1();

super.m2();

}

}

class C1 extends B {

static void m2() {

System.out.println("C m2");

m4();

}

}

class D extends C1 {

static void m1() {

System.out.println("D ml");

}

static void m2() {

System.out.println("D m2");

}

static void m4() {

System.out.println("D m4");

}

public static void main(String[] args) {

D d = new D();

d.m1();

d.m2();

d.m3();

d.m4();

}

}

Ques - 3

----------

Find the statement which will cause compile time error , comment it and find the answer and draw the execution flow.

class A {

static int a = 10;

int x =20;

static void m1() {

System.out.println("A m1");

}

void m2() {

System.out.println("A m2");

}

void m3(){

System.out.println ("A m3");

System.out.println("A a: "+a);

System.out.println("A x: "+x);

m1();

m2();

}

}

class B extends A {

static int a = 50;

int x = 60;

static void m1() {

System.out.println("B m1");

}

void m2() {

System.out.println("B m2");

System.out.println("B a: " + a);

System.out.println("B x: " + x);

}

void m4() {

super.a = a - 10;

super.x = x - 10;

}

}

public class TestAB {

public static void main(String[] args) {

B b1 = new B();

B b2 = new B();

A a1 = new B();

b1.a = 15;

b2.a = 18;

b1.x = 16;

b2.x = 19;

b1.m4();

b2.m4();

b1.m3();

System.out.println();

b2.m3();

System.out.println();

System.out.println(b1.a);

System.out.println(a1.a);

System.out.println(b1.x);

System.out.println(a1.x);

}

}

Ques - 4

----------

Find the Output and draw the heap and stack diagram.

class Example {

int a;

int b;

Example(int a, int b) {

this.a = a;

this.b = b;

}

void m1(Example e) {

a = e.a + 1;

b = e.b + 2;

e.a = a + 3;

e.b = b + 4;

e = null;

}

public static void main(String[] args) {

Example e1 = new Example(10, 10);

Example e2 = new Example(20, 20);

e1.m1(e2);

System.out.println(e1.a + "........" + e2.a);

System.out.println(e1.b + "........" + e2.b);

e2.m1(e1);

System.out.println(e1.a + "........" + e2.a);

System.out.println(e1.b + "........" + e2.b);

e1.m1(e1);

System.out.println(e1.a + "........" + e2.a);

System.out.println(e1.b + "........" + e2.b);

e2.m1(e2);

System.out.println(e1.a + "........" + e2.a);

System.out.println(e1.b + "........" + e2.b);

}

}

Ques - 5

----------

Draw Heap and stack diagram for below code and find the output and find how many objects are eligible for garbage Collector ?

class Example {

Integer i;

Example(int i) {

this.i = i;

}

void m1(Example e) {

e.i+=10;

System.out.println(e.i);

e = new Example(5);

e.i++;

System.out.println(e.i);

}

public static void main(String[] args) {

Example e1 = new Example(10);

Example e2 = new Example(20);

Example e3 = e1;

e1.m1(e3);

System.out.println(e1.i);

e1.m1(e2);

e1 = null;

e3 = e2;

/\*How many objects are eligible for garbage collector at this point ? \*/

}

}

Ques - 6

----------

Draw Heap and stack diagram for below code and find the output and find how many objects are eligible for garbage Collector ?

class Demo {

Demo d = this;

int i = 10;

void m1(Demo d) {

System.out.println(d.i);

d = new Demo();

d.i = 30;

System.out.println(d.i);

d = null;

}

public static void main(String[] args) {

Demo d = new Demo();

System.out.println(d.i);

d.m1(d);

System.out.println(d.i);

d = null;

/\* How many objects are eligible for garbage collector at this point ? \*/

}

}

Q1)  
Find the output of the Program  
  
class InnerTry  
{  
    public static void main(String[] args)  
    {  
        try  
        {  
            String s=null;  
            System.out.println(s.length());  
        }  
        catch (NullPointerException np)  
        {  
            System.out.println("Null Problem");  
            try  
            {  
                String s="xyz";  
                int no=Integer.parseInt(s);  
                System.out.println(no);  
            }  
            catch (NumberFormatException nf)  
            {  
                System.out.println("Number Format Problem");  
            }  
        }  
        System.out.println("Completed");  
    }  
}  
  
A. Compile time error  
B. Null Problem  
   no  
   Completed  
C. Null Problem  
   Number Format Problem  
   Completed  
  
D. 4  
   Null Problem  
   Number Format Problem  
   Completed  
  
Ans :  
  
Q2)  
public class Arithmetic {  
    private static int y;  
    public static void main(String... s) {  
        try {  
            int x = 0;  
            y = 5 / x;  
        } catch (RuntimeException ae) {  
            System.out.println("RuntimeException ");  
        } catch (Exception e) {  
            System.out.println("Exception ");  
        }  
        System.out.println("finished ");  
    }  
}  
  
A. ArithmeticException finished  
B. RuntimeException finished  
C. Exception finished  
D. ArithmeticException  
  
Ans:  
  
Q3)  
class Person{  
     public void talk() {  
           
     }  
}  
  
public class Test  
{  
     public static void main(String[] args) {  
        Person p=null;  
        try {  
             p.talk();  
        }  
        catch(NullPointerException e) {  
             System.out.println("There is NullPointerException");  
        }  
        catch(Exception e) {  
             System.out.println("There is Exception");  
        }  
        System.out.println("Everything went fine");  
    }  
}  
  
A.There is a NullPointerException. Everything went fine.  
B.There is a NullPointerException.  
C.There is a NullPointerException. There is an Exception.  
D.This code will not compile, because in Java there are no pointers.  
  
Ans:  
Q4)  
class Sample{  
    public String test()  
    {  
     try {  
          System.out.println("one");  
          return "hello";  
     }  
     finally {  
          System.out.println("two");  
     }  
}  
}  
public class Test  
{  
    public static void main(String[] args) {  
    Sample s=new Sample();  
    s.test();  
     
}  
}  
A.One Hello two  
B.Two  
C.One Two  
D.Compilation Error  
E.None of these  
  
Ans:  
  
Q5)  
class Demo {  
     
    public int divide(int a,int b)  
    {  
        int result=0;  
    try {  
         int c=a/b;  
        result= c;  
           
    }  
    catch(Exception e ) {  
         System.out.println("Exception");  
    }  
    finally {  
         System.out.println("finally");  
    }  
    return result;  
         
    }  
}  
public class Test  
{  
    public static void main(String[] args) {  
    Demo s=new Demo();  
    System.out.println(s.divide(0,5));  
     
}  
}  
  
A.Prints out: Exception  
B.Prints out: Exception Finally  
C.Compile with error  
D.Prints out: Finally  
E.Finally  
  0  
  
Ans:  
  
Q6)  
public class ExcProgram01 {  
public int m1(int a) {  
        System.out.println("m1 started");  
        while (a == 5) {  
            try {  
                ++a;  
                return 10;  
  
            } finally {  
  
                continue;  
            }  
        }  
        return 20;  
  
    }  
  
    public static void main(String[] args) {  
        ExcProgram01 e3 = new ExcProgram01();  
        System.out.println(e3.m1(5));  
    }  
}  
  
A.m1 started  
  20  
  
B.m1 stated  
  10  
  
C.Compile time error  
  
D.Exception Ocuured at m1  
  
Answer-  
  
Q7)  
public class ExcProgram02 {  
public int m1(int a) {  
while(a==5) {  
try {  
try {  
a=10;  
}  
  
finally {  
a=20;  
}  
a=30;  
return a;  
}  
finally {  
if(a==30) {  
continue;  
}  
}  
}  
return 40;  
}  
  
public static void main(String[] args) {  
ExcProgram02 e4=new ExcProgram02();  
System.out.println(e4.m1(5));  
  
}  
  
}  
  
A.10  
B.20  
C.30  
D.40  
  
Answer-  
  
Q8)  
public class ExcepProgram04 {  
public static void main(String[] args) {  
System.out.println(1);  
try {  
System.out.println(2);  
System.out.println(3);  
System.out.println(4/0);  
}  
  
catch(ArithmeticException e) {  
System.out.println(5);  
System.out.println(6/0);  
System.out.println(7);  
}  
System.out.println(8);  
}  
}  
  
A) Abnormal Termination  
B) 1 2 3 Abnormal Termination  
C) Compile time error  
D) 1 2 3 5 Abnoraml Termination  
  
Ans:  
  
Q9)  
import java.util.Scanner;  
  
public class ExcepProgram05 {  
  
public static void main(String[] args) {  
  
try {  
      Scanner sc=new Scanner(System.in);  
   System.out.println("enter the number for a and b");  
   a=sc.nextInt();  
   int b=sc.nextInt();  
   int result=a/b;  
   System.out.println("The result is :"+result);  
   }  
   catch(ArithmeticException e) {  
     System.out.println(" don’t divide the number with 0");  
   }  
   catch(NumberFormatException ne) {  
    System.out.println("don’t enter anything apart from int");  
   }  
   catch(Exception e) {  
    System.out.println("Exception raised");  
    }  
    finally {  
       System.out.println("Finally execute ");  
    }  
      System.out.println("complete");  
    }  
  
}  
Note:-  
 If the user enters non-integer values, it displays an error message:  
  
A)don’t enter anything apart from int  
   finally execute ?  
    complete  
  
B)Exception raised  
    finally execute ?  
     
C)Exception raised  
   finally execute ?  
    complete  
  
D)don’t enter anything apart from int  
   finally execute ?  
  
Ans:  
  
Q10)  
Who is the super class of all exception and error ?  
A.Exception  
B.Error  
C.RuntimeException  
D.Throwable  
  
Ans :  
  
Q11)  
Given:  
try { int x = Integer.parseInt("two"); }  
  
Which could be used to create an appropriate catch block? (Choose all that apply.)  
A. InputMismatchException  
B. IllegalStateException  
C. NumberFormatException  
D. ExceptionInInitializerError  
  
Answer -  
  
Q12)  
Find the output of the Program  
  
class ExceptionPropogation  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Main method started");  
        m1();  
        System.out.println("Main method ended");  
    }  
    public static void m1()  
    {  
        try  
        {  
            m2();  
        }  
        catch (Exception e)  
        {  
            System.out.println("Handled in m1");  
        }  
    }  
    public static void m2()  
    {  
           m3();  
    }  
    public static void m3()  
    {  
        System.out.println(10/0);  
    }  
}  
  
Ans:  
  
  
  
Q13)  
public class Test{  
 public static void main(String [] args){  
 try{  
      int i;  
      return;  
    }  
catch(Exception e)  
   {  
     System.out.println("In catchBlock");  
    }  
finally{  
     System.out.println("In Finally");  
    }  
}  
}  
  
A.inCatchBlock  
B.inCatchBlock inFinallyBlock  
C.inFinallyBlock  
D.The program will return without printing anything  
  
Ans:  
  
Q14)  
public class Test  
{  
    public static void main(String[] args) {  
        int i;  
        try {  
             i=calculate();  
        }  
        catch(Exception e) {  
             System.out.println("Error occured");  
        }  
    }  
  
    private static int calculate() {  
        return (7/2);  
    }  
}  
  
A.3  
B.3.5  
C.Error occured  
D.Compilation Error  
E.None of these  
  
Ans:  
  
  
Q15)  
class Sample{  
    public String test()  
    {  
     try {  
          System.out.println("one");  
          return "hello";  
     }  
     finally {  
          System.out.println("two");  
     }  
}  
}  
public class Test  
{  
    public static void main(String[] args) {  
    Sample s=new Sample();  
    s.test();  
     
}  
}  
A.One Hello two  
B.Two  
C.One Two  
D.Compilation Error  
E.None of these  
  
Ans:  
  
Q16)  
public class ExcepProgram08 {  
System.out.print("main start");  
System.out.print(m1());  
System.out.print("main end");  
}  
static public int m1() {  
try {  
   System.out.print("In try");  
   int a=10;  
   int b=0;  
   int result=a/b;  
   return result;  
}  
catch(NumberFormatException ArithmeticException) {  
System.out.print("In catch");  
 return 20;  
}  
finally {  
System.out.print("In finally");  
return 30;  
}  
}  
  
}  
  
A)main start In try In catch 20 In finally 30 main end  
B)main start In try In catch  In finally 30 main end  
C)Compile time error  
D)main start In try In finally 30 main end  
  
Ans:  
  
Q17)  
public class ExcepProgram09 {  
public static void main(String[] args) {  
System.out.print("main start");  
System.out.print(m1());  
System.out.print("main end");  
  
}  
static public int m1() {  
try {  
System.out.print("In try");  
int a=10;  
int b=0;  
int result=a/b;  
return result;  
}  
catch(NumberFormatException e) {  
System.out.print("In catch");  
return 20;  
}  
finally {  
System.out.print("In finally");  
int [] a=new int [-10];  
return 10;  
}  
}  
  
}  
  
A)main start In try In finally 20 main end  
B)main start In try In  Abnormal Termination  
C)main start In try In finally main end  
D)main start In try In finally Abnormal Termination  
  
Ans:  
  
Q18)  
public class ExcepProgram10 {  
public static void main(String[] args) {  
System.out.print("main start");  
System.out.print(m1(5));  
System.out.print("main end");  
}  
public static int m1(int a) {  
  
while(a==5) {  
try {  
return 10;  
}  
finally {  
break;  
}  
}  
return 20;  
}  
}  
  
A)main start 10 main end  
B)Compile Time error  
C)main start 10 Abnormal termination  
D)main start 20 main end  
  
Ans:  
  
  
Q19)  
public class ExcepProgram11{  
public static void main(String[] args) {  
System.out.print("main start");  
System.out.print(m1(5));  
System.out.print("main end");  
}  
public static int m1(int a) {  
  
while(a==5) {  
try {  
return 10;  
}  
finally {  
continue;  
}  
}  
return 20;  
}  
}  
  
A)Compile Time error  
B)Infinite Loop Execution  
C)main start 10 Abnormal termination  
D)main start 10 20 main end  
  
Ans:  
  
Q20)  
public class ExcepProgram12 {  
public static void main(String[] args) {  
System.out.println("main start");  
System.out.println(m1(5));  
System.out.println("main end");  
}  
public static int m1(int a) {  
  
while(a++==5) {  
try {  
return 10;  
}  
finally {  
  
}  
}  
return 20;  
}  
}  
  
A)main start 10 20 main end  
B)main start 20 main end  
C)main start 10 main end  
D)There must be a statement in finally that's why it's Compile  
  Time error  
  
Ans:  
  
Q21)  
What is the output of the following code ?  
  
public class ExceptionTest {  
public static void main(String[] args) {  
try {  
int[] arr = new int[5];  
arr[10] = 10;  
System.out.println("Success!");  
} catch (Exception e) {  
System.out.println("Error: " + e.getMessage());  
}  
}  
}  
A. Success!  
B. Error: ArrayIndexOutOfBoundsException  
C. Error: Index 10 out of bounds for length 5  
D. Error: null  
  
Answer:  
  
Q22)  
 What is the output of the following code ?  
  
public class ExceptionTest {  
public static void main(String[] args) {  
try {  
String s = null;  
System.out.println(s.length());  
} catch (Exception e) {  
System.out.println("Error: " + e.getMessage());  
}  
}  
}  
A. Error: NullPointerException  
B. Error: null  
C. Error: Exception  
D. Error: Unhandled exception  
  
Answer:  
  
Q23)  
What is the output of the following code snippet?  
  
public class ExceptionTest {  
public static void main(String[] args) {  
try {  
int a = Integer.parseInt("abc");  
System.out.println(a);  
} catch (NumberFormatException e) {  
System.out.println("Error: " + e.getMessage());  
}  
}  
}  
A. Error: NumberFormatException  
B. Error: null  
C. Error: Exception  
D. Error: Unhandled exception  
  
Answer:  
Q24)  
public class MyException {  
    public static void main(String[] args) {  
          try  
          {  
              return; //#1  
          }  
  
          finally  
          {  
              System.out.print("Finally ");  
              return;  //#2  
          }  
          System.out.print("main end");  //#3  
      }  
    }  
A. Compiletime Error for line #1  
B. Compiletime Error for line #3  
C. Finally main end  
D. Finally  
  
ans:  
  
Q25)  
public class BlockFinal  
{  
    public static void main(String[] a)  
    {  
        try  
        {  
            int i = 10 / 0;  
        }  
        catch (Exception ex)  
        {  
            System.out.print("1st catch Block ");  
        }  
 public class Arithmetic {  
    private static int y;  
    public static void main(String... s) {  
        try {  
            int x = 0;  
            y = 5 / x;  
        } catch (RuntimeException ae) {  
            System.out.println("RuntimeException ");  
        } catch (Exception e) {  
            System.out.println("Exception ");  
        }  
        System.out.println("finished ");  
    }  
}  
  
A. ArithmeticException finished  
B. RuntimeException finished  
C. Exception finished  
D. ArithmeticException  
  
Ans:  
  
Q26)  
 public static void main(String[] args) {  
 try {  
 args = null;  
 args[0] = "test";  
 System.out.println(args[0]);  
 } catch (Exception ex) {  
 System.out.println("Exception");  
 } catch (NullPointerException npe) {  
 System.out.println("NullPointerException");  
 }  
 }  
What is the result?  
A. test  
B. Exception  
C. Compilation fails.  
D. NullPointerException  
  
Ans :  
  
Q27)  
public class Test {  
    public static void main(String[] args) {  
        String s="";  
        int x = 10;  
        try {  
            s+="a";  
            System.out.println("Outer try..");  
            try {  
                 s+="b";  
                 int y = x/0;  
                 System.out.println("Inner try of outer try");  
            } catch (ArithmeticException e) {  
                  int i = Integer.parseInt(s);  
                  s+="c";  
                  System.out.println("Inner catch of outer try");                        
            } finally {  
                  s+="d";  
                  System.out.println("Inner finally of outer try");  
            }  
        } catch (NullPointerException ex) {  
            try {  
                  s+="e";  
                  System.out.println("Inner try of Outer catch");  
            } catch (Exception e) {  
                s+="f";  
                System.out.println("Inner catch of Outer catch");  
            } finally {  
                s+="g";  
                System.out.println("Inner finally of outer catch");  
            }  
        } catch (Exception e) {  
            s+="h";  
            System.out.println("Outer catch of outer try..");  
        } finally {  
            System.out.println(s);  
        }  
  
    }  
  
}  
  
A. Outer try..  
   Inner finally of outer try  
   Outer catch of outer try..  
   abdh  
  
B. Outer try..  
   Inner finally of outer try  
   Inner catch of Outer catch  
   Inner finally of outer catch  
   Outer catch of outer try..  
   abdfgh  
  
C. Outer try..  
   Inner try of outer try  
   Inner catch of outer try  
   Inner finally of outer try  
   Inner try of Outer catch  
   Inner catch of Outer catch  
   Inner finally of outer catch  
   Outer catch of outer try..  
   abcdefgh  
  
D. Compile time error  
Ans :  
  
Q28)public class Test {  
    public static void parse(String str) {  
         try {  
         float f = Float.parseFloat(str);  
        } catch (NumberFormatException nfe) {  
         f = 0.0F;  
         } finally {  
         System.out.println(f);  
        }  
         }  
         public static void main(String[] args) {  
         parse("invalid");  
         }  
}  
A. 0.0  
B. Compilation fails.  
C. A ParseException is thrown by the parse method at runtime.  
D. A NumberFormatException is thrown by the parse method at runtime.  
Answer:  
  
Q29)public class Test {  
         static int[] a;  
        static { a[0]=2; }  
        public static void main( String[] args ) {  
            System.out.println("hi");  
        }  
    }  
A. java.lang.StackOverflowError  
B. java.lang.IllegalStateException  
C. java.lang.ExceptionInInitializerError  
D. java.lang.ArrayIndexOutOfBoundsException  
Answer:  
  
Q30)  
class Test{  
public static void main(String[] args) {  
String s = "-";  
try {  
doMath(args[0]);  
s += "t "; // line 6  
}  
finally { System.out.println(s += "f "); }  
}  
public static void doMath(String a) {  
int y = 7 / Integer.parseInt(a);  
} }  
And the command-line invocations:  
java Input  
java Input 0  
Which are true? (Choose all that apply.)  
A. Line 6 is executed exactly 0 times.  
B. Line 6 is executed exactly 1 time.  
C. Line 6 is executed exactly 2 times.  
D. The finally block is executed exactly 0 times.  
E. The finally block is executed exactly 1 time.  
F. The finally block is executed exactly 2 times.  
G. Both invocations produce the same exceptions.  
H. Each invocation produces a different exception.  
Answer:

https://lh3.googleusercontent.com/a/ACg8ocLxDQ2qtoCJ-Ttl9yvW0Lq_7j2d0r8PQYev8AWe3qVBjHw10L4b=s40-c

Add class comment…

Post

https://lh3.googleusercontent.com/a-/ALV-UjWYie25o0d6VRv7E2pD8LUPd_oNgGNrdrZJMhif9KxDr1kGiGw=s40-c

**Announcement: "=============== Exception Handling in…"**

Jagannath Bhandari

Created Apr 21Apr 21

=============== Exception Handling in Java==========================  
 - Exception is an event that occurs during the execution due to wrong  
   input and it leads to abnormal termination of program execution.  
     
 - Whenever an exception is raised, JVM creates an object of type  
   of Exception or sub type of Exception and then returns the exception  
   related message to the console.  
     
 - Exception is a predefined class available in java.  
 - Exception is a sub class of Throwable.  
 - Throwable is the super-most class for each Error and Exception and their  
   sub types.  
 - Error is a predefined class.  
 - Error has sub classes such as:  
       - AssertionError  
       - ThreadDeath  
       - VirtualMachineError  
       - LinkageError  
   \*AssertionError thrown to indicate that an assertion has failed.  
   \* It was introduced in java 1.4 version.  
   
  \* ThreadDeath is thrown in the victim thread when the (deprecated) Thread.stop()  
    method is invoked.  
    -It was introduced in Java 1.0 version.  
    -An application should catch instances of this class only if it  
     must clean up after being terminated asynchronously.If  
     ThreadDeath is caught by a method, it is important that it should  
     be re-thrown so that the thread actually dies.  
       
  \*VirtualMachineError object is thrown to indicate that the Java Virtual Machine  
   is broken or has run out of resources necessary for it to continue operating.  
   - It has multiple sub classes:  
        a. InternalError:  
          Its object thrown is to indicate some unexpected internal error that has occurred in  
            the Java Virtual Machine.  
        b. StackOverflowError: It is thrown when a stack overflow occurs because of an application  
            recursing too deeply.  
        c. UnknownError:Thrown when an unknown but serious exception has occurred in the  
           Java Virtual Machine.  
        d.OutOfMemoryError:It is thrown when the JVM cannot allocate an object  
           because it is out of memory, and no more memory could be made  
           available by the garbage collector.  
             
    \* LinkageError: Its  a subclass of Error.  
          Its subclasses indicate that a class has some dependency on another class;  
          however, the latter class incompatibly changed after the compilation  
          of the former class.  
          It has sub classes such as:  
               a. ClassFormatError  
               b. IncompatibleClassChangeError  
               c. ClassCirculatoryError  
               d. ExceptionInitializerError  
               e. VerifyError  
               f. UnsatisfiedLinkError  
               g. NoClassDefoundError  
              Some of the miscellaneous Error as sub class of IncompatibleClassChangeError  
               are:   NoSuchFieldError,NoSuchMethodError,InstanitationError,  
               AbstractMethodError etc.  
    Note:  
    NoSuchFieldError is thrown if an application tries to access or modify a specified  
    field of an object, and that object no longer has that field.  
    Generally, this error is caught by the compiler; this error can  
    only occur at run time if the definition of a class has  
    incompatibly changed.  
     
    NoSuchMethodError: It is thrown if an application tries to call a specified method of a  
    class (either static or instance), and that class no longer has a  
    definition of that method.  
    Normally, this error is caught by the compiler; this error can  
    only occur at run time if the definition of a class has  
    incompatibly changed.  
         
    IllegalAccessError: It  is thrown if an application attempts to access or modify  
    a field, or to call a method that it does not have access to.  
     
     
  ---Exception class in java:  
   - Exception is a predefined class available in java.lang package.  
   - It is the sub class of Throwable.  
   - Exception and its subclasses indicates conditions that are reasonable for an  
     application might to catch.  
   - Exception has in general two categories of sub classes:  
         - Sub classes of RuntimeException  
         - Direct Sub classes of Exception  
       
      - Sub classes of Exception:  
          a. RunTimeException  
          b. ClassNotFoundException  
          c. NoSuchFieldException  
          d. NoSuchMethodException  
          e. InterruptedException  
          f. AccessorException  
          g. ActivationException  
          h. AlreadyBoundException  
          i. AnalyzerException  
          j. AWTEXception  
          k. BackingStoreException  
          l. BrokenBarrierException  
          m. CertificateException  
          n. CGIClientException  
          o. CompilerException  
          p. DataFomatException  
          q. DispatchException  
          r. DestroyFailedException  
          s. DuplicateException  
          t. ExceutionException  
          u. FilterException  
          v. IllegalClassFormatException  
          w. InvalidApplicationException  
          x. InvalidValueException  
          y. IOException  
          z. JAXBException  
          -------------  
          a. LambdaConversionException  
          b. LocaleSyntaxException  
          c. NamingException  
          d. NoSuchEventException  
          e. NoSuchProducerException  
          f. NoSuchRecordingException  
          g. NotBoundException  
          h. ParseException  
          i. PathException  
          j. ReflectiveCopyException  
          k. ReflectiveOperationException  
          l. ScriptException  
          m. ServerNotActiveException  
          n. ServerSideException  
          o. SOAPException  
          p. SQLException  
          q. TargetLostException  
          r. TimeOutException  
          s. WindowsException  
          t. XMLParseException  
   
  ===============================================================  
-Methods inherited from class java.lang.Throwable into Exception & Error  
    addSuppressed(),  
    fillInStackTrace(),  
    getCause(),  
    getLocalizedMessage(),  
    getMessage(),  
    getStackTrace(),  
    getSuppressed(),  
    initCause(),  
    printStackTrace(),  
    setStackTrace(),  
    toString()  
       
-Methods inherited from class java.lang.Object  
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait      
   
     
 - To overcome exception in program we have to use  
     try-catch block.  
     Syntax:  
   
      try  
      {    
      }  
      catch(Parameter list){    
      }  
       
   try catch block:  
   -It is  a block especially designed to handle exception  
     raised in java program.  
       
   - try is a keyword.  
   - It contains a block in which we provide risky statements  
     and its related code.  
   - When an exception is raised outside the block then even if  
     there is catch block with same type of Exception then also  
     the catch block will not execute.  
 ======================================================================  
 In try block we place the exception causing statements/ risky  
statements and its related statements.  
  
In catch block we provide exception handling statements.  
There is a chance to get an exception inside catch block.  
  
The catch block is executed only when an exception is raised  
in the associated try block and the exception object type matches  
with catch block's parameter type.  
  
An exception can't be caught by catch block if the type is not  
available as parameter in catch.In this case JVM  will  
terminate abnormally.  
  
If no execption is raised from try block then the catch block  
will not execute.  
Nested try catch block:  
If an exception is raised outside try then even if the catch has  
matching parameter , the exception will not be caught by catch  
block,here program terminates abnormally.  
  
Once exception is raised in try block then the catch will start  
exceuting, the remaining code after exception raising statement  
from try will not execute.  
  
If an exception raises in catch block and even if the exception  
matches with the parameter of same catch block then alos it will  
not be caught by catch block.  
  
If there is nested try catch:  
     case 1: Exception raised out side try catch then we get  
     abnormal termination even if the exception type matches  
     with parameter of catch block in inner try or outer try.  
  
     case 2: Exception raised in outer try block and there is  
     matching exception type as parametr in inner catch and outer  
     catch then ...  
     Inner catch will execute when exception raised in inner  
     try matches the param.  
     Outer catch will execute when exception raised in outer  
     try matches the param.  
       
     case 3: exception raised in inner catch and the type matches  
     with outer catch then....  
  
    case 4: Exception raised in inner try but exception type  
    not matched in inner catch but matches with outer catch  
    then ...  
  
    case 5: Exception raised in inner try but exception  
    type not matched in inner catch and with outer catch then ...  
  
    In an application we have some logic to be exceuted at  
    any cost even if exception is raised, and handled or not  
    handled. In such cases we must use a special block i.e.,  
    finally block.  
  
    Syntax:  
     try{  
     }  
     catch(Exceptiontype ev){  
     }  
     finally{  
     }  
//Case 1: No exception raised in try.finally wil execute  
//Case 2: Exception in try block.Catch parameter matches.  
          catch executes,finally executes.  
//CAse 3: Exception in try block.Catch parameter not matched.  
          catch executes,finally executes.  
//case 4: When exception raised and not matched with catch then  
          finally block executes first then only exception  
      message displayed;  
//case 5: if there is a return statement then also finally block  
          will execute.  
//case 6: find the o/p  
 class Test{  
 public static int m1(){  
  System.out.println("m1 starts");  
  try{  
   System.out.println("inside try");  
   return 100;  
  }  
  catch(ArithmeticException ae){  
  System.out.println("inside catch");  
  }  
  finally{  
  System.out.println("inside finally");  
  return 200;  
  }  
 }  
public static void main(String []args){  
 System.out.println("main starts");  
 System.out.println(m1());  
 System.out.println("main ends");  
}  
}  
If value is returned from try then JDH stores the value then  
control goes to finally block then returns to main method.  
  
//case 7: find the o/p  
 class Test{  
 public static int m1(){  
  System.out.println("m1 starts");  
  try{  
   System.out.println("inside try");  
   break;  
  }  
  catch(ArithmeticException ae){  
  System.out.println("inside catch");  
  }  
  finally{  
  System.out.println("inside finally");  
  }  
  return 200;  
  }  
 }  
public static void main(String []args){  
 System.out.println("main starts");  
 System.out.println(m1());  
 System.out.println("main ends");  
}  
}  
error because of break used without loop.  
  
  
//case 8: find the o/p  
 class Test{  
 public static int m1(){  
  System.out.println("m1 starts");  
  int i = 1;  
  while(i++ <=5){  
  try{  
   System.out.println("inside try");  
   break;  
  }  
  catch(ArithmeticException ae){  
  System.out.println("inside catch");  
  }  
  finally{  
  System.out.println("inside finally");  
  }  
  return 200;  
  }  
 }  
public static void main(String []args){  
 System.out.println("main starts");  
 System.out.println(m1());  
 System.out.println("main ends");  
}  
}  
  
  
//case 9:  
class Test{  
public static int m1(){  
 int i = 1;  
 while(i++ <=5){  
  try{  
  System.out.println("inside try");  
  continue;  
  }  
  catch(ArithmeticException ae){  
   System.out.println("inside catch");  
  }  
  finally{  
  System.out.println("inside finally");  
  }  
 }  
 return 200;  
}  
public static void main(String []args){  
System.out.println("main starts");  
System.out.println(m1());  
System.out.println("main ends");  
}  
}  
Whenever control comes out from try block ut must execute finally block.  
  
When control is coming out of try block is it possible to not execute fianlly block?  
Answer:Yes, by shutting down JVM.But the other codes are also not going to execute.  
You must use System.exit(0);  
  
Can  we provide return statement in  finally?  
Answer: yes.  
  
  
multiple combinations of try catch finally:  
1. try  
  {  
    logic;  
  }  
  
2.catch(){  
   logic;  
   }  
3. finally{  
    logic;  
   }  
4. try{  
   logic1;  
   }  
   catch(Exception a){  
   logic2;  
   }  
   finally{  
   logic 3;  
   }  
5. try{  
   logic1;  
   }  
   finally{  
   logic 3;  
   }  
   catch(Exception a){  
   logic2;  
   }  
6. try{  
   logic1;  
   }  
   finally{  
   logic 3;  
   }  
7.  try  
  {  
    logic;  
  }  
  catch(Exception1 a){  
   logic2;  
   }  
   catch(Exception2 a){  
   logic2;  
   }  
   catch(Exception3 a){  
   logic2;  
   }  
 8.   try  
     {  
    logic;  
    }  
    finally{  
   logic 1;  
   }  
   finally{  
   logic 2;  
   }  
 9.  try  
  {  
    logic;  
  }  
  catch(Exception1 a){  
   logic2;  
   }  
   catch(Exception2 a){  
   logic2;  
   }  
   catch(Exception3 a){  
   logic2;  
   }  
   finally{  
   logic 4;  
   }  
case 10.  
   try{  
   }  
   catch(){  
   }  
   statement;  
   finally{  
   }  
  
  
  
Solutions:  
1. try with no catch, not allowed  
2. catch without try not allowed  
3. only finally not allowed  
4. allowed  
5. not allowed  
6. allowed  
7. allowed  
8. not allowed  
9. allowed  
10. not allowed  
  
  
Rules:  
1. try catch finally -- allowed  
2. try catch -- allowed  
3. try finally ---allowed  
4. single try multiple catch-- allowed  
5. try catch catch finally -- allowed  
6. try catch multiple finally -- not allowed  
7. single try multiple catch with same type parameter---not allowed  
8. try catch finally at class level--not allowed  
9. return statement in finally---allowed (no further statement allowed  
10. return statement in finally block inside if condition ,we dont get get CTE for futher statement  
    after finally block  
10.inner catch parameter of try catch in outer try and outer catch parameter  
   can  be same type  
11.inner catch parameter of try catch in outer catch and outer catch  
   parameter must not be same type  
12. variable declared in try block  cant be accessed in catch block and finally block  
13. variable declared in method cant be accessed inside try,catch as well as finally in same  
    method if its not initialised in declaration level.  
14.variable declared in method but initialised in try block cant be accessed in other block.  
    Because statements inside try block may not get executed as there may be exception at any  
    line of instruction in try block.  
15. variable declared in method but initialised in try block cant be accessed in other block.  
    But if reinitailised in catch block1 then no CTE.but in another catch block2 it cant be  
    accessed.  
16. If variable is declared in method then it must be initialised in each block where we want  
    to access.  
17. In real time, always initialise variable in method with compatible value,then you will not  
   get error while accessing from try catch finally block.  
   Here we gurantee to compiler that x value is provided no need to worry , access the value  
   initialised.

Garbage collection (GC) is a part of the Java Virtual Machine (JVM)

that automatically destroys unused objects to reclaim runtime memory.

This process helps to maintain free memory for running Java applications.

GC works in two steps:

Mark: Identifies which memory is in use and which isn't.

Sweep: Removes objects identified during the mark phase.

GC is performed by a daemon thread called Garbage Collector (GC). The GC

only collects objects that are created by the new keyword. An object can

become unreferenced in several ways, including: Nulling the reference,

Assigning a reference to another, and Anonymous object.

GC is important because it: Prevents memory-related errors like

OutOfMemoryError, Eliminates dangling pointer bugs, and Eliminates

double free bugs.

However, GC can impact application performance in several ways, including:

Application pauses

During GC cycles, the application may experience pauses causing the application

to hang or lag.

High CPU usage

GC can consume a significant amount of CPU resources, impacting application

performance.

You can request JVM to run Garbage Collector using two methods:

System.gc(): This method is contained in the System class.

Runtime.getRuntime().gc(): This method is contained in the Runtime class.

The call System.gc() is effectively equivalent to the call

Runtime.getRuntime().gc().

Every JVM can implement its own version of garbage collection, however,

it should meet the standard JVM specification.

Important Concepts Related to Garbage Collection in Java

1. Unreachable objects: An object is said to be unreachable if it doesn’t

contain any reference to it. Also, note that objects which are part of the

island of isolation are also unreachable.

Integer i = new Integer(4);

//the new Integer object is reachable via the reference in 'i'

i = null;

//the Integer object is no longer reachable.

2. Eligibility for garbage collection: An object is said to be eligible for

GC(garbage collection) if it is unreachable. After i = null, integer object

4 in the heap area is suitable for garbage collection in the above image.

Ways to make an object eligible for Garbage Collector:

Even though the programmer is not responsible for destroying useless objects

but it is highly recommended to make an object unreachable(thus eligible for GC)

if it is no longer required.

There are generally four ways to make an object eligible for garbage collection.

- Nullifying the reference variable

- Re-assigning the reference variable

- An object created inside the method

- Island of Isolation

Ways for requesting JVM to run Garbage Collector

Once we make an object eligible for garbage collection, it may not destroy

immediately by the garbage collector. Whenever JVM runs the Garbage Collector

program, then only the object will be destroyed. But when JVM runs Garbage

Collector, we can not expect.

We can also request JVM to run Garbage Collector.

There are two ways to do it :

Using System.gc() method: System class contain static method gc() for requesting

JVM to run Garbage Collector.

Using Runtime.getRuntime().gc() method: Runtime class allows the application

to interface with the JVM in which the application is running. Hence by using

its gc() method, we can request JVM to run Garbage Collector.

There is no guarantee that any of the above two methods will run Garbage

Collector.

The call System.gc() is effectively equivalent to the call : Runtime.getRuntime().gc()

Finalization

Just before destroying an object, Garbage Collector calls finalize() method on the object to perform cleanup activities. Once finalize() method completes, Garbage Collector destroys that object.

finalize() method is present in Object class with the following prototype.

protected void finalize() throws Throwable

Based on our requirement, we can override finalize() method for performing

our cleanup activities like closing connection from the database.

The finalize() method is called by Garbage Collector, not JVM. However,

Garbage Collector is one of the modules of JVM.

Object class finalize() method has an empty implementation. Thus, it is

recommended to override the finalize() method to dispose of system resources

or perform other cleanups.

The finalize() method is never invoked more than once for any object.

If an uncaught exception is thrown by the finalize() method, the exception

is ignored, and the finalization of that object terminates.

Advantages of Garbage Collection in Java:

The advantages of Garbage Collection in Java are:

It makes java memory-efficient because the garbage collector removes the

unreferenced objects from heap memory.

It is automatically done by the garbage collector(a part of JVM), so we

don’t need extra effort.

Rules:  
1. try with no catch, not allowed  
2. catch without try not allowed  
3. only finally not allowed  
4. duplicate exceptions type in catch not allowed.  
5. multiple finally block not allowed for one try  
-----------------------------------------------------------------------  
  
1. try catch finally -- allowed  
2. try catch -- allowed  
3. try finally ---allowed  
4. single try multiple catch-- allowed  
5. try catch catch finally -- allowed  
6. try catch multiple finally -- not allowed  
7. single try multiple catch with same type parameter---not allowed  
8. try catch finally at class level--not allowed  
9. return statement in finally---allowed (no further statement allowed  
  If return statement is in finally block inside if condition ,we don't get get CTE for  
further statement after finally block  
10.inner catch parameter of try catch in outer try and outer catch parameter  
   can  be same type  
11.inner catch parameter of try catch in outer catch and outer catch  
   parameter must not be same type  
12. variable declared in try block can't be accessed in  
    catch block and finally block  
13. variable declared in method can't be accessed inside try,catch as  
    well as finally in same method if its not initialized  
    in declaration level.  
14.variable declared in method but initialized in try block  
    can't be accessed in other block.  
    Because statements inside try block may not get  
    executed as there may be exception at any  
    line of instruction in try block.  
15. variable declared in method but initialized in try block  
    can't be accessed in other block.  
    But if re-initialized in catch block1 then no CTE.but in  
     another catch block2 it can't be  
    accessed.  
16. If variable is declared in method then it must be initialized  
    in each block where we want to access.  
17. In real time, always initialize variable in method with  
    compatible value,then you will not get error while  
    accessing from try catch finally block.  
   Here we guarantee to compiler that x value is provided no  
   need to worry , access the value i.e, initialized.  
=======================================================================================  
18. in a non-void method, return statement only in try block  
    will generate CTE.  
    value must be returned just before from end of method also.  
19. in a non void method, return statement only in try block  
will generate CTE.  
    value must be returned from catch block even if from  
    not returned just before  end of method.  
20. if there are multiple catch block and value not returned  
    from all the catch block  
   then there will be again CTE.But make sure not to provide any statement after last catch  
   returning value,or else we will get CTE.If you have to execute the statements later  
   then in this case you must provide statements in finally  
   block so that there will be no CTE.  
   =================================================================================  
  All about Throwable:  
   Point to remember:  
 ---------------------------------------------------|  
For any exception message, the format is :            |  
   "Exception class name" + "reason of exception"    |  
     "position of exception generated in code"        |  
--------------------------------------------------------|  
The Throwable class in Java is the root class of all errors and exceptions in the  
Java language. It provides methods for getting the stack trace, the cause of the  
throwable, and the message.  
Throwable has two direct subclasses:  
         Error and Exception.  
  
Methods of Throwable class:  
getMessage(): Returns the message of the throwable.  
getStackTrace(): Returns an array of stack trace elements  
             representing the stack trace related to current  
             Throwable.  
getCause(): Returns the cause of the throwable.  
printStackTrace(): Prints this throwable and its backtrace to  
           the standard error stream.  
addSuppressed(): Appends the specified exception to the list of exceptions that were  
                suppressed in order to deliver this exception.  
fillInStackTrace(): Fills in the stack trace of this throwable.  
initCause(): Initializes the cause of this throwable.  
======================================================================================  
  
throw:  
It is a keyword.  
It is used for throwing an exception from a method or constructor  
 based on conditions.  
Only one exception can be thrown at a time using throw keyword.  
  
Syntax:  
   throw new Throwable or any of its sub class object();  
     // use object directly or use its RV  
For e.g:  
whenever we use any login page we provide data as input,  
background input is verified whether its avaialble in DB or not.  
If not then system terminates the execution and we get Exception message on console.  
  
By throwing an exception we are actually terminating the execution and passing  
Readable instructions to user so that user will not be worried about the reason of  
exception.  
  
Points to remember:  
   - It must be used only inside a method,constructor,  
   - It can't throw multiple type of exception object at a time.  
   - If you have to throw multiple exception object then each throw exception object  
     statement must be placed in different conditional statement  
   - It can't be used at prototype level of method or constructor.  
   - It can't be used inside static block, instance block.  
   - AFTER throw exception object statement you must not provide any other statement,  
     if you have to execute other statement then put throw exception statement inside  
     Conditional statement.  
  
  
throws:  
It is a keyword.  
It is used for informing the caller method about the exception.  
throws keyword doesn't have any importance at execution time  
Whether throws is used or not used JVM will send the exception to the calling method.  
It can report multiple exceptions at a time.We must place comma when we have to report  
multiple exceptions.  
  
Points to remember:  
  - It must be placed only after method parenthesis or constructor parenthesis.  
  - Its mandatory to be used for checked exceptions  
  - Its optional to be used for unchecked exceptions  
  - It is not allowed in static block , instance block or any other element  
    other than after method or constructor parenthesis  
  - We can use multiple exception types after throws by using comma between types.  
  - If a method in super class doesn't contain throws checked exception then  
    in sub class overridden method also we should not provide throws.  
  
e.g:  
   public static void m1(int a) throws AE,SIOBE,AIOBE{  
      if()  
      throw new AE("message 1");  
      if()  
      throw new SIOBE("message 2");  
      if()  
      throw new AIOBE("message 3");  
   }  
  
  
If a called method is throwing exception to caller then it must be caught by using  
try catch in caller method.  
  
Q. What is the difference between throw and throws?  
Answer:  
throw is used for throwing an exception manually by the developer based on conditions.  
throws is used to report the exceptions thrown in called method to the caller method so  
that the exception is handled in the caller method by using try catch block.  
  
Only one exception object is allowed after throw keyword.  
Multiple exceptions types are allowed after throws keyword.  
  
When we use throw keyword, we must place exception object after it,  
When we use throws keyword we must place exception class name after it.  
  
throw must be used inside the body of method as part of logic.  
throws must be used in method header after parameter list i.e., after '()'  
  
throw keyword is generally used for throwing checked and customized exceptions.  
throws keyword is used for reporting checked exceptions and it is mandatory.  
  
  
Q. When shall we use throw, throws , try catch?  
Answer: Whenever we have to develop a method which takes input from user  
and based on any condition we have to use throw and throws for throwing  
exception and report the exception to the caller respectively.  
  
When we have to suppress exception and continue our program execution or  
provide readable/understandable message to the user regarding the cause of  
exception then we must use try catch.  
  
------------------------------------------------------------------------------  
=================================================================================  
Types of Exceptions:  
    In Java, there are two types of exceptions: checked and unchecked.  
    Checked exceptions are exceptions that the compiler forces you to handle,  
     while unchecked exceptions are exceptions that you are not required to  
      handle.  
Here are some examples of checked exceptions:  
FileNotFoundException: - This exception is thrown when a file that  
                        you are trying to access does not exist.  
IOException: - This exception is thrown when an input or output  
                  operation fails.  
SQLException: - This exception is thrown when a database  
                 operation fails.  
Here are some examples of unchecked exceptions:  
ArithmeticException: - This exception is thrown when an arithmetic  
                operation, such as division by zero, fails.  
NullPointerException: - This exception is thrown when you try to access  
                  a null object.  
ArrayIndexOutOfBoundsException: - This exception is thrown when you  
              try to access an element of an array that is out of bounds.  
You can tell if an exception is checked or unchecked by looking at its  
 superclass. If the superclass is Exception or one of its subclasses, then  
  the exception is checked. If the superclass is RuntimeException or one  
   of its subclasses, then the exception is unchecked.  
     
It is generally considered good practice to handle checked exceptions,  
 even though you are not required to do so. This is because checked  
  exceptions are typically indicative of a problem with your code  
   that you should fix. Unchecked exceptions, on the other hand, are  
    typically indicative of a problem with the user input or the  
     environment in which your code is running.  
       
Here are some tips for handling checked exceptions:  
Use a try-catch block to catch the exception and handle it appropriately.  
Re-throw the exception to the calling method.  
Declare the exception in the throws clause of the method signature.  
Here are some tips for handling unchecked exceptions:  
Use a try-catch block to catch the exception and handle it appropriately.  
Log the exception and continue running.  
Ignore the exception.  
It is important to note that there are some exceptions to the general  
rules for handling checked and unchecked exceptions.  
For example, the InterruptedException exception is a checked exception,  
 but it is often ignored because it is typically thrown when a thread  
 is interrupted.  
Ultimately, the decision of whether or not to handle a checked exception  
 is up to you. However, it is important to be aware of the potential  
  consequences of not handling a checked exception.  
   
    --------------------Custom Exception--------------  
  A custom exception in Java is a class that extends the Exception class.  
It is used to handle specific error scenarios in a more precise manner.  
Custom exceptions are typically used to provide more detailed information  
about the error, such as the cause of the error and the steps that can be  
taken to resolve it.  
To create a custom exception, you need to create a class that extends the  
Exception class. The class should have a constructor that takes a String as  
the error message. The class should also have methods to get the error  
message and the cause of the error.  
Here is an example of a custom exception:  
public class CustomException extends Exception {  
  
    private String errorMessage;  
    private Throwable cause;  
  
    public CustomException(String errorMessage) {  
        super(errorMessage);  
        this.errorMessage = errorMessage;  
    }  
  
    public CustomException(String errorMessage, Throwable cause) {  
        super(errorMessage, cause);  
        this.errorMessage = errorMessage;  
        this.cause = cause;  
    }  
  
    public String getErrorMessage() {  
        return errorMessage;  
    }  
  
    public Throwable getCause() {  
        return cause;  
    }  
}  
Once you have created a custom exception, you can use it in your code by  
throwing an instance of the exception. The following code shows how to  
throw an instance of the CustomException class:  
  
try {  
    // Code that may throw an exception  
} catch (CustomException e) {  
    // Handle the exception  
}  
You can also use custom exceptions to catch specific error scenarios.  
The following code shows how to catch an instance of the CustomException class:  
Java  
  
try {  
    // Code that may throw an exception  
} catch (CustomException e) {  
    // Handle the exception  
} catch (Exception e) {  
    // Handle other exceptions  
}  
Custom exceptions are a powerful tool that can be used to improve  
the quality of your Java code. By using custom exceptions, you can  
provide more detailed information about errors and handle specific  
error scenarios in a more precise manner.  
//Custom Exception Class  
@SuppressWarnings("serial")  
class InvalidTransactionException extends Exception {  
 public InvalidTransactionException(String message) {  
     super(message);  
 }  
}  
  
//Bank Account Class  
class BankAccount {  
 private String accountNumber;  
 private double balance;  
  
 public BankAccount(String accountNumber, double initialBalance) {  
     this.accountNumber = accountNumber;  
     this.balance = initialBalance;  
 }  
  
 public void deposit(double amount) {  
     balance += amount;  
     System.out.println("Deposited: " + amount);  
 }  
  
 public void withdraw(double amount) throws InvalidTransactionException {  
     if (amount <= 0) {  
         throw new InvalidTransactionException("Invalid amount for withdrawal");  
     }  
     if (amount > balance) {  
         throw new InvalidTransactionException("Insufficient funds for withdrawal");  
     }  
     balance -= amount;  
     System.out.println("Withdrawn: " + amount);  
 }  
  
 public double getBalance() {  
     return balance;  
 }  
}  
  
//Main Class to Demonstrate Bank Account Operations  
public class BankDemo {  
 public static void main(String[] args) throws InterruptedException {  
     // Create a bank account with initial balance  
     BankAccount account = new BankAccount("123456789", 1000);  
  
     // Perform transactions  
     try {  
         account.deposit(500);  
         System.out.println("Current Balance: " + account.getBalance());  
         Thread.sleep(5000);  
         account.withdraw(200);  
         System.out.println("Current Balance: " + account.getBalance());  
         Thread.sleep(5000);  
         account.withdraw(1500); // This will throw an exception  
         System.out.println("Current Balance: " + account.getBalance()); // This will not execute  
     } catch (InvalidTransactionException e) {  
         Thread.sleep(5000);  
         System.err.println("Transaction Error: " + e.getMessage());  
     }  
 }  
}

Exception propagation in Java is the process by which an exception

is thrown from a method and is not caught by that method, so it is

propagated to the calling method. This process continues up the call

stack until the exception is caught or it reaches the main() method,

which terminates the program.

There are two types of exceptions in Java: checked exceptions and

unchecked exceptions. Checked exceptions are those that must be

handled by the programmer, while unchecked exceptions are those

that do not have to be handled.

When an exception is thrown, the Java Virtual Machine (JVM) searches

for a catch block that can handle the exception. If a catch block is

found, the JVM transfers control to that block. If no catch block is

found, the exception is propagated to the calling method.

This process continues up the call stack until a catch block is found

or the main() method is reached. If the main() method does not have a

catch block for the exception, the program terminates.

Here is an example of exception propagation in Java:

public class ExceptionPropagation {

public static void main(String[] args) {

try {

method1();

} catch (Exception e) {

System.out.println("Exception caught in main(): " + e.getMessage());

}

}

public static void method1() {

try {

method2();

} catch (Exception e) {

System.out.println("Exception caught in method1(): " + e.getMessage());

}

}

public static void method2() throws Exception {

throw new Exception("This is an exception");

}

}

In this example, the method2() method throws an exception. The method1()

method does not have a catch block for the exception, so the exception is

propagated to the main() method. The main() method has a catch block for the

exception, so the exception is caught and the program does not terminate.

Exception propagation can be a useful tool for handling errors in Java programs.

By propagating exceptions up the call stack, you can ensure that errors are

handled by the appropriate code.

==================Modifiers=====================  
/\*  
 Modifiers are the keywords which are used to change the default  
 functionality of any programming element.  
 We have access modifiers as well as non access modifiers:  
     Access Modifiers:The modifiers which are used to set the limitation  
     regarding accessing of any programming element at project level are  
     called as access modifiers.They are:  
     1.public  
     2.private  
     3.protected  
     4. <default>  
       
Q1. Can we declare a class as public?  
Answer: Yes.  
  
Q2. How many classes can be declared as public in a program?  
Answer:At most one.recommended for class containing main  
method.  
  
Q3. Can we declare a constructor as public?  
Answer:Yes  
  
Q4. Can we declare constructor as private?  
Answer: Yes.  
  
Q5. Can we declare a global variable as public?  
Answer: Yes  
  
Q6. Can we declare a local variable as public?  
Answer: No.  
  
Q7. Can we declare method as public?  
Answer: Yes.  
  
Q8. Can we declare a block as public?  
Answer:No, because there is no identifier for a block.  
  
Q9. Can we declare an interface as public?  
Answer: Yes.In the case when class name and file name  
are same, then interface can't be public.  
  
Q10. Can we declare inner class as public if outer class  
is public?  
Answer: Yes  
  
Q11. Can we declare inner class as public of outer class  
is not public?  
Answer:Yes  
  
Q12. Can we declare inner class as private?  
Answer:Yes  
   
Q13. Why do we declare any programming element as public?  
Answer: Access modifiers lets us know about where to access  
the element in a project.If any element is declared as public  
then the element can be accessed throughout the project.  
  
Q14. Can we declare a class as private?  
Answer:Outer class is not allowed to be private  
but inner class can be declared as private.  
  
Q15. Can we declare main method as private?  
Answer:No, but if the main method is overloaded then  
we can have a private main method.  
  
Q. Can we declare user defined method as private?  
Answer: Yes.  
  
Q16. Can we declare block as private?  
Answer: No.  
  
Q17. Can we declare global variables as private?  
Answer: Yes.  
  
Q18. Can we declare local variables as private?  
Answer: No.  
  
Q19. Can we declare constructor as private?  
Answer: Yes, after this we can't create object outside the class.  
-----------------------------------------------------------  
Q20. Can we access public global variable from another class  
 in same package?  
  
Q21. Can we access public global variable from sub class  
 in same package?  
   
Q22. Can we access public global variable from another class  
 in different package?...............to be continued

How do we create custom thread?  
 Answer:  
 Steps to create custom thread using Thread class:  
 Step 1: Create a class deriving from Thread  
 Step 2: override run() and add logic to execute  
 Step 3: create sub class object and invoke start()  
         so that we make run() ready to run  
     
Step 1: Create class implementing Runnable  
Step 2: Implement run() with the logic to execute  
Step 3: Create sub class object  
Step 4 : Create Thread class object by passing SC rv  
  
While extending from Thread we create only sub class  
object.  
While implementing Runnable we create sub class  
object as well as Thread class object.  
run(): we provide logic  
start() : we execute the logic, it causes the custom thread to start  
  
We cant't guarantee the output for threaded application.  
  
What happens when we create Thread class or its sub class object?  
Answer: When we create Thread class object or its sub class object  
  - The Thread or its sub classes loaded  
  - Thread class SV,NSV,Blocks are executed  
  - A new instance is created with Thread and CurrentClass Thread instance variables  
  - WHatever logic is given in Thread class no param constructor,  
    the instance variables of Thread class are initialized with  
    values.  
    default values for important variables:  
    target: null  means execute run() from current thread i.e., current class  
    group: null, changes to current class object reference 1010  
    daemon: false, changes to daemon with parent thread value  'false'  
    priority: 0 , changes to parent thread priority '5'  
    name: null, changes to Thread-0  
    threadStatus: 0  
   - Once the Thread object is created, when we call the start(),  
     its logic is loaded and executed in current thread that is main thread,  
     and then it requests JVM to start custom thread for the current thread object "rv"  
     by loading the run() of current custom thread class.  
   - once custom thread is made ready to execute,start() changes the  
     threadStatus values and return to calling method.  
   - when main method logic execution is paused or continued, the custom thread  
   execution is started, and run concurrently with other threads.  
     
       
Parent Thread : The thread(method) in which thread is created.  
   
  
Important Variables of Thread:  
target,group,daemon,priority,threadStatus  
  
name: The name of the thread.    Thread-0 its default name.  
priority: The priority of the thread.  
state: The state of the thread.  
daemon: Whether the thread is a daemon thread.  
interrupted: Whether the thread has been interrupted.  
stackTrace: The stack trace of the thread.  
contextClassLoader: The context class loader of the thread.  
group:    main  
  
These variables can be accessed using the following methods:  
getName(): Returns the name of the thread.  
getPriority(): Returns the priority of the thread.  
getState(): Returns the state of the thread.  
isDaemon(): Returns whether the thread is a daemon thread.  
isInterrupted(): Returns whether the thread has been  
             interrupted.  
getStackTrace(): Returns the stack trace of the thread.  
getContextClassLoader(): Returns the context class loader  
              of the thread.  
Important methods of Thread:  
    1. start():  
    2. run():  
public class ThreadExample {  
    public static void main(String[] args) {  
        Thread thread = new Thread();  
  
        // Get the name of the thread.  
        String name = thread.getName();  
        System.out.println("The name of the thread is: " + name);  
  
        // Get the priority of the thread.  
        int priority = thread.getPriority();  
        System.out.println("The priority of the thread is: " + priority);  
  
        // Get the state of the thread.  
        Thread.State state = thread.getState();  
        System.out.println("The state of the thread is: " + state);  
  
        // Check if the thread is a daemon thread.  
        boolean isDaemon = thread.isDaemon();  
        System.out.println("Is the thread a daemon thread? " + isDaemon);  
  
        // Check if the thread has been interrupted.  
        boolean isInterrupted = thread.isInterrupted();  
        System.out.println("Has the thread been interrupted? " + isInterrupted);  
  
        // Get the stack trace of the thread.  
        StackTraceElement[] stackTrace = thread.getStackTrace();  
        System.out.println("The stack trace of the thread is:");  
        for (StackTraceElement element : stackTrace) {  
            System.out.println(element);  
        }  
  
        // Get the context class loader of the thread.  
        ClassLoader contextClassLoader = thread.getContextClassLoader();  
        System.out.println("The context class loader of the thread is: " + contextClassLoader);  
    }  
}

Q1.  
Implement two custom exceptions, InvalidAmountException and  
InsufficientFundException, to handle different error scenarios  
during account transactions. Additionally, create the necessary  
classes and methods to simulate this banking scenario.  
  
InvalidAmountException:  
Create a custom checked exception named InvalidAmountException that  
extends Exception.  
Implement a constructor that takes a String message parameter and  
passes it to the superclass constructor.  
  
InsufficientFundException:  
Create a custom checked exception named InsufficientFundException that  
extends Exception.  
Implement a constructor that takes a String message parameter and passes  
it to the superclass constructor.  
  
BankAccount Class:  
Create a class named BankAccount representing a customer's bank  
account with the following attributes:  
accountNumber (String): Unique account number.  
balance (double): Current account balance.  
  
Implement a constructor to initialize the accountNumber and balance.  
  
Implement a method deposit(double amount) that allows a customer to  
deposit money in their account.  
If the amount is negative or zero, throw an InvalidAmountException.  
  
Implement a method withdraw(double amount) that allows a customer to  
withdraw money from their account.  
If the amount is negative or zero, throw an InvalidAmountException.  
If the amount exceeds the available balance, throw an InsufficientFundException.  
  
BankingApplication Class:  
Create a class named BankingApplication to simulate account operations.  
Instantiate a BankAccount object for a customer.  
Demonstrate withdrawal operations with different amounts, catching  
and handling appropriate exceptions.

**//------------------------ InvalidAmountException------------------------------------**

**class InvalidAmountException extends Exception**

**{**

**InvalidAmountException(String msg)**

**{**

**super(msg);**

**}**

**}**

**//--------------------------------InsufficientFundException--------------------------**

**class InsufficientFundException extends Exception**

**{**

**InsufficientFundException(String msg)**

**{**

**super(msg);**

**}**

**}**

**//---------------------------------BankAccount -----------------**

**class BankAccount**

**{**

**String accountNumber;// Unique account number.**

**double balance; //Current account balance**

**public BankAccount(String accountNumber, double balance)**

**{**

**super();**

**this.accountNumber = accountNumber;**

**this.balance = balance;**

**}**

**void deposite(double amount)**

**{**

**balance+=amount;**

**}**

**void withdraw(double amount) throws InvalidAmountException, InsufficientFundException**

**{**

**if(amount<=0)**

**{**

**throw new InvalidAmountException("Withdraw amount be grater than zero");**

**}**

**if(amount>balance)**

**{**

**throw new InsufficientFundException("Insufficient fund to withdraw");**

**}**

**}**

**}**

**//--------------------------------------------------------------------------BankingApplication---------**

**public class BankingApplication**

**{**

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

**{**

**BankAccount ba=new BankAccount("10105789",1000);**

**try**

**{**

**ba.deposite(1000);**

**ba.withdraw(0);**

**}**

**catch(InvalidAmountException | InsufficientFundException e)**

**{**

**System.*err*.println("Transaction error : "+e.getMessage());**

**}**

**}**

**}**

Q2.Programming Question: Custom Exceptions in Car Simulation  
  
Scenario:  
  
You are developing a car simulation program that models various driving scenarios. Implement custom exceptions to handle specific error conditions encountered during the simulation.  
  
Tasks:  
  
CarOverheatException:  
Create a custom checked exception named CarOverheatException to handle situations where the car's engine temperature exceeds a safe limit during driving.  
Implement a constructor that accepts a String message describing the exception.  
  
LowFuelException:  
Create a custom checked exception named LowFuelException to handle situations where the car's fuel level is insufficient for a driving distance.  
Implement a constructor that accepts a String message describing the exception.  
  
CarBreakdownException:  
Create a custom checked exception named CarBreakdownException to handle random breakdowns that may occur during driving.  
Implement a constructor that accepts a String message describing the exception.  
  
SpeedLimitExceededException:  
Create a custom checked exception named SpeedLimitExceededException to handle situations where the car's speed exceeds the legal limit.  
Implement a constructor that accepts a String message describing the exception.  
  
Car Class:  
Create a class named Car with the following attributes:  
fuelLevel (double): Current fuel level of the car.  
engineTemperature (double): Current temperature of the car's engine.  
speed (double): Current speed of the car.  
  
Implement a constructor to initialize the fuelLevel, engineTemperature, and speed.  
  
Implement a method drive(double distance, double desiredSpeed) that simulates driving the car for a  
specified distance at a desired speed.  
If the fuelLevel is insufficient for the distance, throw a LowFuelException.  
If the engineTemperature exceeds 100 degrees Celsius during the drive, throw a CarOverheatException.  
If the car's speed exceeds 120 km/h, throw a SpeedLimitExceededException.  
Simulate a breakdown with a 5% probability during the drive, throwing a CarBreakdownException.  
  
DrivingSimulation Class:  
Create a class named DrivingSimulation to simulate driving scenarios using the Car class.  
Instantiate a Car object with initial parameters (e.g., fuel level, engine temperature, speed).  
Demonstrate driving operations by invoking the drive method with different scenarios, catching and  
handling the appropriate exceptions.  
  
  
Q3)  
Programming Scenario-Based Question: Instagram Account Authentication  
  
Scenario:  
  
You are tasked with developing an authentication system for Instagram accounts in Java.  
The system will validate usernames and passwords against predefined lists and throw custom  
exceptions for invalid credentials.  
  
Tasks:  
  
Custom Exceptions:  
Create two custom checked exceptions:  
InvalidUserNameException: Thrown when the provided username is not found in the list of valid  
usernames.  
InvalidPasswordException: Thrown when the provided password does not match the corresponding  
username's password.  
  
Authentication System:  
Implement a class named InstagramAuthenticator with the following methods and attributes :  
Instance Variables:  
validUsernames (String Array): An array containing the valid usernames for Instagram accounts.  
passwords (int Array): An array containing the corresponding passwords for the valid usernames.  
  
Constructor:  
public InstagramAuthenticator(String[] validUsernames, int[] passwords):  
Initializes the validUsernames and passwords arrays with the provided values.  
  
Methods:  
public boolean authenticate(String username, int password) throws InvalidUserNameException, InvalidPasswordException:  
  
This method takes a username (String) and password (int) as parameters.  
It checks if the username exists in the validUsernames array.  
If the username is not found, throw an InvalidUserNameException with an appropriate error message.  
If the username is found, retrieve the corresponding password from the passwords array.  
If the retrieved password does not match the provided password, throw an InvalidPasswordException  
with an appropriate error message.  
If both checks pass, return true to indicate successful authentication.  
Instructions:  
  
Implement the InstagramAuthenticator class according to the above specifications.  
Define and use the custom exceptions (InvalidUserNameException, InvalidPasswordException) to handle authentication errors.  
Demonstrate the usage of the InstagramAuthenticator class by creating instances with predefined usernames and passwords, and testing authentication with both valid and invalid credentials.  
Ensure proper exception handling to catch and print error messages when authentication fails due to invalid username or password.