Name – **Kunal Dhanawade**

PRN – **230944520039**

1. Code this:

define your exception class by name "MyArithException" (MyArithException.java)

define class "Calculator" (Calculator.java)

In this class define a function, which will accept an int and return double of it. e.g., int caldouble(int) this function will check an int which is passed to it

It will raise the exception if 0 is passed or negative passed. in case of 0 caller of this method should get error message "Zero not allowed"

in case of negative caller of this method should get error message "negative not allowed"

In case of positive value, it should simply return the double value. (This method shouldn't handle the exception)

Now define class "MyCalcApp" (MyCalcApp.java)

In this class you accept a number from user and pass that no. to caldouble() method which is mentioned above.

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **public** **class** MyArithException **extends** Exception {  **public** MyArithException(String message) {  **super**(message);  }  } | | **public** **class** Calculator {    **static** **int** caldouble(**int** num) **throws** MyArithException {    **if**(num==0)  **throw** **new** MyArithException("Zero not allowed");  **if**(num<0)  **throw** **new** MyArithException("Negative not allowed");    **return** num\*2;  }  } | | **import** java.util.Scanner;  **public** **class** MyCalcApp {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Scanner sc = **new** Scanner(System.***in***);    System.***out***.print("Enter a no. :");  **int** num = sc.nextInt();    Calculator c = **new** Calculator();  **try** {  **int** res = c.*caldouble*(num);  System.***out***.println(res);  } **catch** (MyArithException e) {  System.***out***.println(e);  }  System.***out***.println("Done");  }  } | |
|  |
| |  | | --- | | Enter a no. :24  48  Done | | Enter a no. :0  MyArithException: Zero not allowed  Done | | Enter a no. :-6  MyArithException: Negative not allowed  Done | |

1. Code this:

on the developer side create an exception:

NumberNotDivisibleBySevenException

as a checked exception

create necessary jar and documentation.

on client side

public class MyMath class with

public void disp(int num)

this disp() method will check if the number passed is not divisible by 7 , it will raise "NumberNotDivisibleBySevenException" or else it will simply display the number passed. (This method shouldn't handle the exception)

public class Demo

main function

invoke "disp()" of "MyMath" class.

|  |
| --- |
| /\*\*  \*  \*/  package devpack;  /\*\*  \* @author Kunal  \*  \*/  public class NumberNotDivisibleBySevenException extends Exception {    /\*\*  \* @param e  \*/  public NumberNotDivisibleBySevenException() {  super("Number Not Divisible By Seven Exception");  }  } |
| |  | | --- | | **package** clientpack;  **import** devpack.NumberNotDivisibleBySevenException;  **public** **class** MyMath {  **public** **void** disp(**int** num) **throws** NumberNotDivisibleBySevenException {  **if**(num%7!=0)  **throw** **new** NumberNotDivisibleBySevenException();  System.***out***.println("the entered no. is: "+num);  }  } | | **package** clientpack;  **import** devpack.NumberNotDivisibleBySevenException;  **public** **class** Demo {  **public** **static** **void** main(String[] args) {  MyMath m = **new** MyMath();    **try** {  m.disp(7);  } **catch** (NumberNotDivisibleBySevenException e) {  System.***out***.println(e);  }  System.***out***.println();  **try** {  m.disp(24);  } **catch** (NumberNotDivisibleBySevenException e) {  System.***out***.println(e);  }  System.***out***.println("\nDone");  }  } | |
| the entered no. is: 7  devpack.NumberNotDivisibleBySevenException: Number Not Divisible By Seven Exception  Done |

1. Code this:

define "MyException" as a checked exception.

define a class "Demo" with public void show1(), public void show2() , public void show3() and main functions.

inside "show3()" accept a number and if it is greater than 10 raise "MyException" else display the number. (this method shouldn't handle the exception)

main() function should call "show1()" ,

show1() function should call "show2()",

show2() function should call "show3()"

show2() should not handle the exception but show1() should handle.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **public** **class** MyException **extends** Exception {  **public** MyException(String message) {  **super**(message);  }  } | | **public** **class** Demo {    **public** **static** **void** show1() {  **try** {  *show2*();  } **catch** (MyException e) {  e.printStackTrace();  }  }    **public** **static** **void** show2() **throws** MyException {  *show3*(7);  *show3*(24);  }    **public** **static** **void** show3(**int** num) **throws** MyException {  **if**(num>10)  **throw** **new** MyException("Number is greater than 10");  System.***out***.println("the number is: "+num);  }  **public** **static** **void** main(String[] args) {  *show1*();  System.***out***.println("Done");  }  } | |
|  |
| the number is: 7  MyException: Number is greater than 10  at Demo.show3(Demo.java:19)  at Demo.show2(Demo.java:14)  at Demo.show1(Demo.java:6)  at Demo.main(Demo.java:24)  Done |

1. Code this:

on the developer side create following checked exception:

InvalidLengthException

create necessary jar file and documentation.

on client side

public class Authenticator

with a parameterized constructor which takes String as a password.

this class also will have "done()" method with "successful authentication" message.

Parameterized constructor should check the length of the password passed if it is less than 5 or more that 9, it should raise "InvalidLengthException" (constructor shouldn't handle the exception)

create a class "Demo" with main

inside main function create the object of "Authenticator" class and invoke "done()" method.

|  |
| --- |
| /\*\*  \*  \*/  package devpack;  /\*\*  \* @author Kunal  \*  \*/  public class InvalidLengthException extends Exception {    public InvalidLengthException() {  super("Invalid Length Exception");  }  } |
| |  | | --- | | **package** clientpack;  **import** devpack.InvalidLengthException;  **public** **class** Authenticator {  **public** Authenticator(String pwd) **throws** InvalidLengthException {  **int** len = pwd.length();  **if**(len<5 || len>9)  **throw** **new** InvalidLengthException();  }    **public** **void** done() {  System.***out***.println("sucessful authentication");  }  } | | **package** clientpack;  **import** devpack.InvalidLengthException;  **public** **class** Demo {  **public** **static** **void** main(String[] args) {  **try** {  Authenticator m = **new** Authenticator("kunal");  // Authenticator m = new Authenticator("kd");  // Authenticator m = new Authenticator("kunaldhanawade");  m.done();  } **catch** (InvalidLengthException e) {  e.printStackTrace();  }    System.***out***.println("\nDone");  }  } | |
| |  | | --- | | sucessful authentication  Done | | devpack.InvalidLengthException: Invalid Length Exception  at clientpack.Authenticator.<init>(Authenticator.java:10)  at clientpack.Demo.main(Demo.java:10)  Done | | devpack.InvalidLengthException: Invalid Length Exception  at clientpack.Authenticator.<init>(Authenticator.java:10)  at clientpack.Demo.main(Demo.java:11)  Done | |

1. Code this:

on the developer side

create checked exception "ResourceNotAllocatedException"

create a class:

public class MyResource implements AutoCloseable

{

public MyResource(int capacity) throws ResourceNotAllocatedException

{

if(capacity>100)

{

throw new ResourceNotAllocatedException("not sufficient space");

}

}

void disp()

{

System.out.println("successful");

}

@Override

public void close() {

System.out.println("resource is closed");

}

}

create necessary jar file and documentation

on the client side

public class Demo with main function

inside main function, create object of "MyResource" and invoke "disp()" method.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | /\*\*  \*  \*/  package devpack;  /\*\*  \* @author Kunal  \*  \*/  public class ResourceNotAllocatedException extends Exception {    public ResourceNotAllocatedException(String message) {  super(message);  }  } | | /\*\*  \*  \*/  package devpack;  /\*\*  \* @author Kunal  \*  \*/  public class MyResource implements AutoCloseable {    public MyResource(int capacity) throws ResourceNotAllocatedException {  if(capacity>100)  {  throw new ResourceNotAllocatedException("not sufficient space");  }  }    public void disp() {  System.out.println("successful");  }  public void close() {  System.out.println("resource is closed");  }  } | |
| **package** clientpack;  **import** devpack.MyResource;  **import** devpack.ResourceNotAllocatedException;  **public** **class** Demo {  **public** **static** **void** main(String[] args) {  **try** {  MyResource res = **new** MyResource(124);  res.disp();  } **catch** (ResourceNotAllocatedException e) {  e.printStackTrace();  }    System.***out***.println("\nDone");  }  } |
| devpack.ResourceNotAllocatedException: not sufficient space  at devpack.MyResource.<init>(MyResource.java:15)  at clientpack.Demo.main(Demo.java:10)  Done |