1) 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 messege) {

super(messege);

}

}

public class Calculator {

public double caldouble(int a) throws MyArithException{

if (a < 0) {

throw new MyArithException("negative not allowed");

}

if (a == 0) {

throw new Error("Zero not allowed");

}

return (double) a;

}

}

public class MyCalcApp {

public static void main(String[] args) {

int a = -1;

Calculator c = new Calculator();

try {

System.***out***.println(c.caldouble(a));

} catch (MyArithException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}}}

2) 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 dev;

public class NumberNotDivisibleBySevenException extends Exception {

public NumberNotDivisibleBySevenException(String msg) {

super(msg);

}

}

package client;

import dev.NumberNotDivisibleBySevenException;

public class MyMath {

public void disp(int num) throws NumberNotDivisibleBySevenException {

if (num % 7 != 0) {

throw new NumberNotDivisibleBySevenException("Number not divisible by 7: " + num);

}

System.***out***.println(num);

}

}

package client;

public class Demo {

public static void main(String[] args) {

MyMath m = new MyMath();

try {

m.disp(10);

} catch (Exception e) {

e.printStackTrace();

}

}

}

3) 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 messege) {

super(messege);

}

}

public class Demo {

public void show1(int a) {

try {

show2(a);

} catch (MyException e) {

e.printStackTrace();

}

}

public void show2(int a) throws MyException {

show3(a);

}

public void show3(int a ) throws MyException {

if(a > 10) {

throw new MyException("Number greater than 10");

}

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Demo d = new Demo();

d.show1(110);

}

}

4) 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 developer;

public class InvalidLengthException extends Exception {

public InvalidLengthException(String messege) {

super(messege);

}

}

package client;

public class Authenticator {

public Authenticator(String pass) throws InvalidLengthException

{

if (pass.length() < 5 || pass.length() > 9) {

throw new InvalidLengthException("Length Invalid");

}

}

}

package client;

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

try {

new Authenticator("Hii");

}

catch(Exception e) {

e.printStackTrace();

}

}

}

5)

Create a custom checked exception InsufficientFundsException, which extends Exception.

Create a class BankAccount with:

Fields: int accountNumber, String accountHolder, double balance.

Method withdraw(double amount), which:

raises InsufficientFundsException if balance < amount.

Deducts the amount from balance if funds are sufficient.

Method deposit(double amount), which adds money to the balance.

Create a main() method to:

Create a BankAccount object with an initial balance.

Try withdrawing an amount greater than the balance (handle the exception using try-catch).

Deposit money and retry the withdrawal.

public class InsufficientFundsException extends Exception {

public InsufficientFundsException(String messege){

super(messege);

}

}

public class BankAccount {

int accountNumber;

String accountHolder;

double balance;

BankAccount(int amount){

balance=amount;

System.***out***.println("balance : "+balance);

}

void withdraw(double amount) throws InsufficientFundsException {

if(balance < amount) {

throw new InsufficientFundsException("Insufficient Funds");

}

balance-=amount;

System.***out***.println("balance : "+balance);

}

void deposit(double amount) {

balance+=amount;

System.***out***.println("balance : "+balance);

}

public static void main(String[] args) {

BankAccount ba = new BankAccount(100);

try {

ba.withdraw(200);

}

catch(Exception e){

e.printStackTrace();

}

ba.deposit(10000);

try {

ba.withdraw(10);

} catch (InsufficientFundsException e) {

e.printStackTrace();

}

}

}

6) 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 developer;

public class ResourceNotAllocatedException extends Exception {

public ResourceNotAllocatedException(String messege) {

super(messege);

}

}

package developer;

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 client;

import developer.MyResource;

import developer.ResourceNotAllocatedException;

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

try {

new MyResource(200).disp();

} catch (ResourceNotAllocatedException e) {

e.printStackTrace();

}

System.***out***.println("Done");

}

}