Throw

Click to Continue

Throw clause is used to generate & raise exceptions.

How to throw?

throw <Exception Object>

```
throw new NullPointerException("Student cannot be null");
```

Throw needs to be used in conjunction with ***throws*** clause. Assume method M2 invokes M1. M2 should

- Catch the exception thrown by M1 **(or)**
- M2 should use ***throws*** clause in the methods signature and throw the exception.

When Throw?

Click to Continue

Throw clause will be used for throwing exception during undesirable circumstances.

Scenario 1: Throw the exception AS-IS:

Assume a program tries to add a element in an array of length 50 at index 100. Java will throw ***ArrayIndexOutOfBoundException***

- The exception can be caught and rethrown as a user defined exception.

(or)

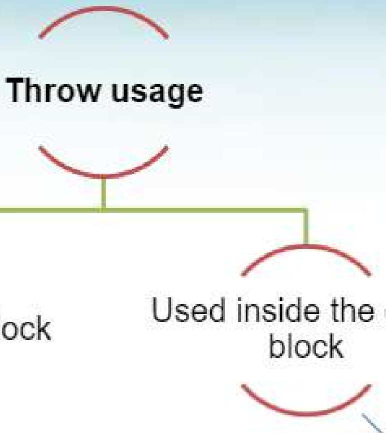
- Performs some logic and throw it as ***ArrayIndexOutOfBoundException*** as it is.

Scenario 2: Throw a user defined exception

User enters a invalid email id as "abc:gmail-com" the java program can check the format and thrown an user defined exception **InvalidEmailException**.

Throw usage

Click to Continue



```
try{
        If(age ==10){
                throw new  <Throwable-Object>;
        }
        catch(<Throwable-Object>)
        {
        // The thrown exception is caught here
        }
```

```
try{
        //some code here which throws
        an exception
}
catch(<Throwable-Object ){
        throw < Throwable-Object >;
}
```

Try it out –throws

Click to Continue

1: In the **ExceptionDemo** class **divide** method perform the following logic.

If Divisor is zero throw a ***ArithmeticException*** with message "Divisor cannot be zero"

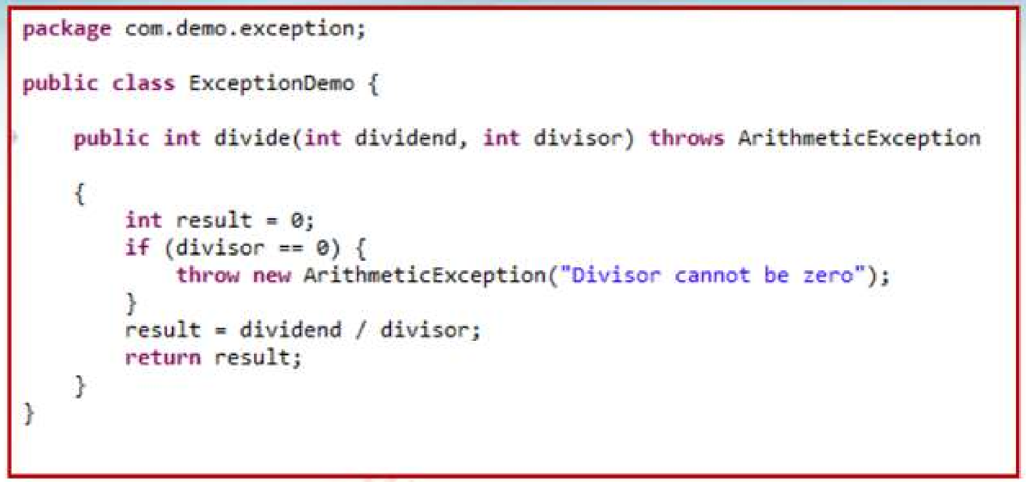
- This method should throw this ***ArithmeticException.***

2: The exception thrown needs to be handled in ***ThrowsExceptionDemo***

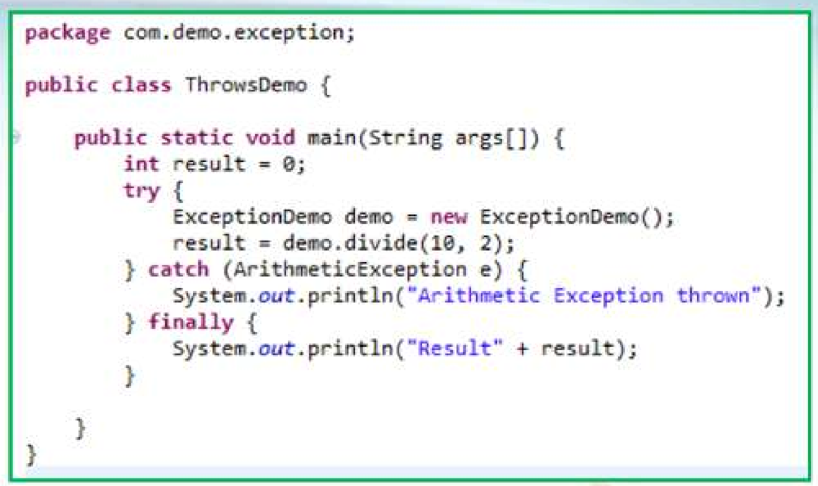
- The main method should ***catch*** the ***ArithmeticException*** thrown by the divide method and print the message in the exception Object.
- The try/catch block should also have a finally block which prints a message "The result is" <Result>

Try it out Solution

Click to Continue



```java
package com.demo.exception;

public class ExceptionDemo {

    public int divide(int dividend, int divisor) throws ArithmeticException

    {
        int result = 0;
        if (divisor == 0) {
            throw new ArithmeticException("Divisor cannot be zero");
        }
        result = dividend / divisor;
        return result;
    }
}
```



```java
package com.demo.exception;

public class ThrowsDemo {

    public static void main(String args[]) {
        int result = 0;
        try {
            ExceptionDemo demo = new ExceptionDemo();
            result = demo.divide(10, 2);
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception thrown");
        } finally {
            System.out.println("Result" + result);
        }

    }
}
```

Test the programs for two scenarios and see which blocks are getting invoked,

Scenario 1: Pass divisor as 2 , you should notice only the finally block message printed.

Scenario 2: Pass the divisor as 0 , you should notice the catch and finally block message being printed.

Try it out - Print Stack Trace Method

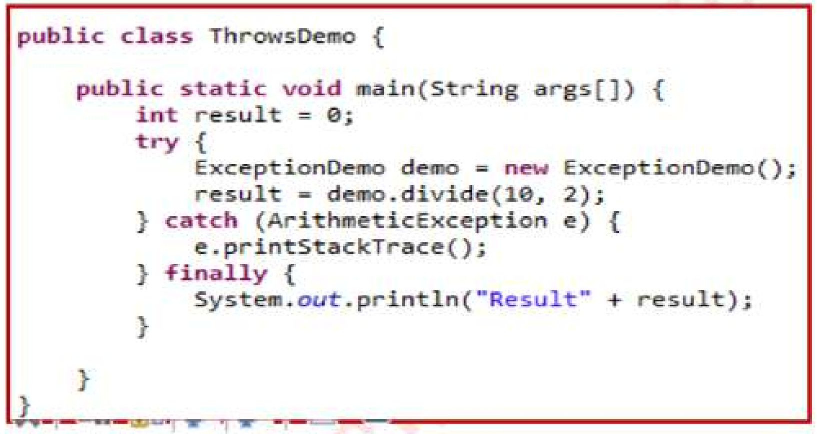
Click to Continue

Print Stack Trace is a method used for printing the exception stack, which maintains the invocation sequence of methods.

This method prints the error message of exception object and also the line number where the exception occurred, thus helps in identifying root cause of a problem.

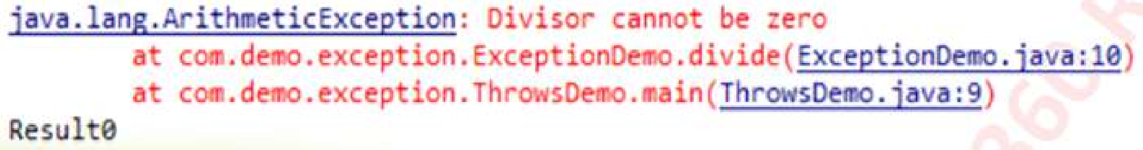
IMPORTANT: It is not a good practice to use **printStackTrace** in application, since it consumes a lot of memory and cpu cycles.

In the previous example, replace the print statement in the catch block with print stack trace and observe the output.



```
public class ThrowsDemo {

    public static void main(String args[]) {
        int result = 0;
        try {
            ExceptionDemo demo = new ExceptionDemo();
            result = demo.divide(10, 2);
        } catch (ArithmeticException e) {
            e.printStackTrace();
        } finally {
            System.out.println("Result" + result);
        }

    }
}
```

Stack Trace:



```
java.lang.ArithmeticException: Divisor cannot be zero
	at com.demo.exception.ExceptionDemo.divide(ExceptionDemo.java:10)
	at com.demo.exception.ThrowsDemo.main(ThrowsDemo.java:9)
Result0
```

A Problem statement

Click to Continue

An application has three classes ***C1,C2 and C3*** with methods ***M1,M2 and M3*** respectively**,** where each class invokes the method in the other class in this order **M1→ M2→ M3**.

Class C3 method **M3** needs to check if the student age is empty or not.

If the age is empty the class C3 needs to handle it by throwing an custom error "**Invalid Age**".

How can we solve this problem?

We need to create **user defined exceptions (or) custom exceptions.**

User Defined Exception

Click to Continue

Custom exceptions are user defined exception which are created by programmers.

Illustration: In a Retail application the developers can create the following exceptions,

- ***InvalidProductException-*** Thrown when the product entered by the user is invalid.
- ***ServiceTaxException –*** Exception thrown when service tax is zero.

Creating User Defined Exceptions

Click to Continue

Scenario: Assume an application has a business logic of validating the Product Price entered by the user, the programmer may create a user defined exception named “**ProductPriceException”** and throw it when the validation fails. Let us create the exception now.

Steps to create a user defined exception:

1: Create a Java class which extends the Exception class.

2: Override the necessary constructors.

Create a java class extending Exception

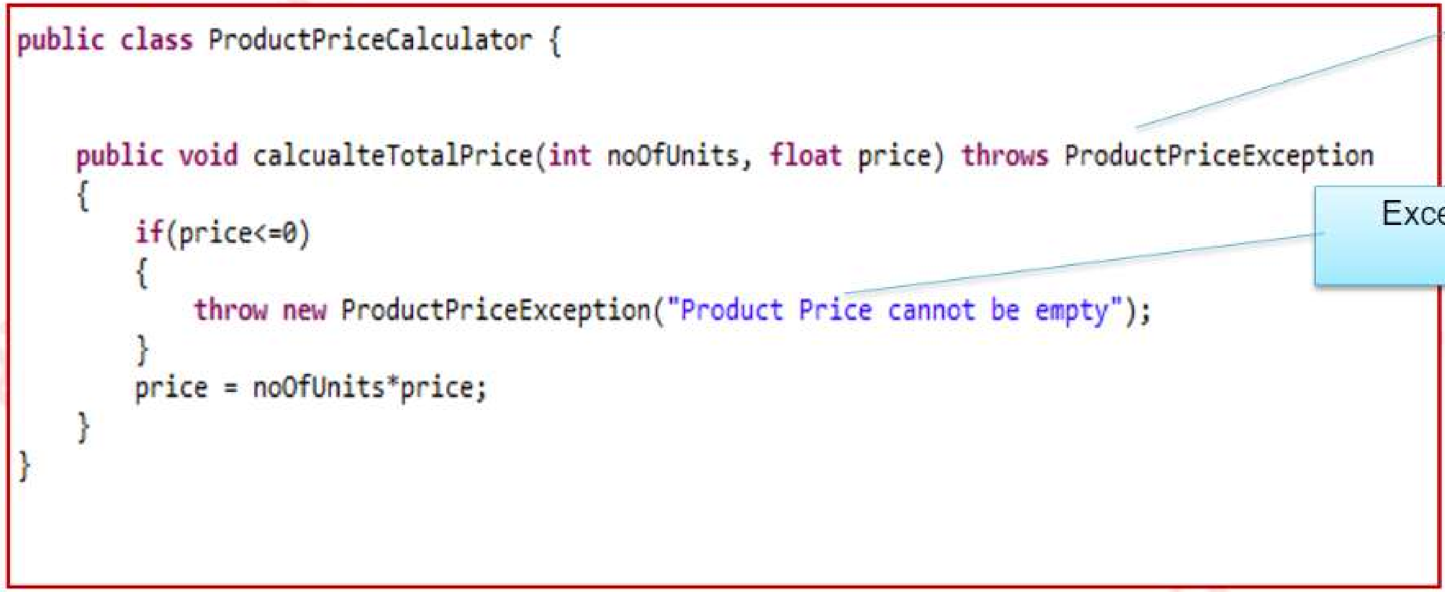
```java
public class ProductPriceException extends Exception{

    public ProductPriceException()
    {
        super();
    }

    public ProductPriceException(String message)
    {
        super(message);
    }

    public ProductPriceException(Throwable throwable)
    {
        super(throwable);
    }
}
```

Override the required constructors.

Throw the exception..

Click to Continue

Develop a **ProductPriceCalculator** which will check the product price is <=0 , if yes will throw the **ProductPriceException**.



```
public class ProductPriceCalculator {


    public void calcualteTotalPrice(int noOfUnits, float price) throws ProductPriceException
    {
        if(price<=0)
        {
            throw new ProductPriceException("Product Price cannot be empty");
        }
        price = noOfUnits*price;
    }
}
```

Method should declare the exception to be thrown.

Exception thrown.

Try it out – User Defined Exception

Click to Continue

Problem Statement: A pharmaceutical company is developing an application where it allows the user to search for medicines and order medicines online. As a developer you are supposed to develop the orderMedicines API. This method accepts MedicineName, Quantity.

Business Rule: If the medicine name is null or quantity is <0 the method should throw an error

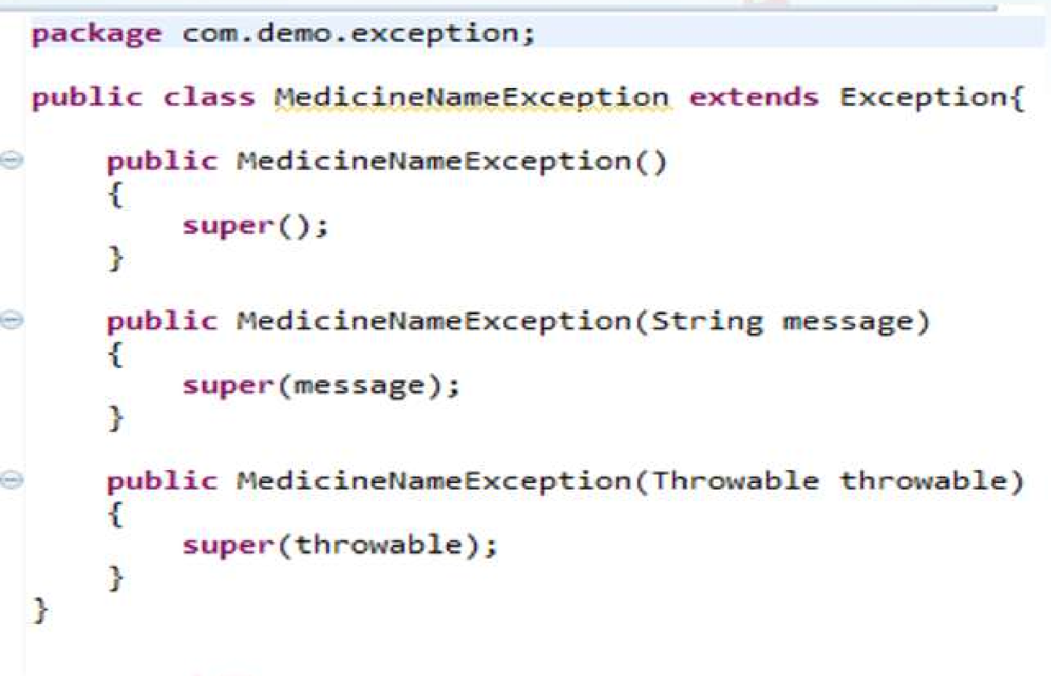
1. Create a user defined exception classes named "***MedicineNameException***" & "***MedicineQuantityException***"
2. Overload the respective constructors.
3. Create a main class "Pharmacy" , add the following method,
   - ***orderMedicine -*** The parameter are String medicineName, int age, int quantity. Add the following logic
     - *if quantity < =0 throw a **MedicineQuantityException** with error message "quantity cannot be less than zero"*
     - *If name is null or empty throw a **MedicineNameException** with error message "Medicine name cannot be empty"*

   - *Invoke the method **orderMedicine*** from the main method with the data specified and see how the program behaves,

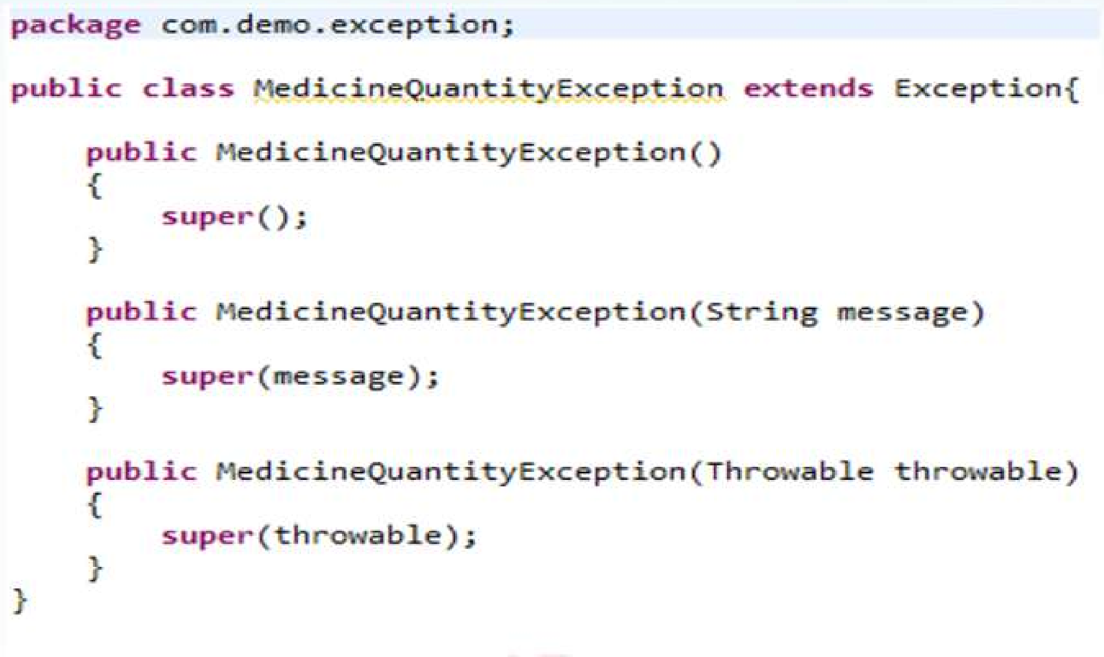
Try it out Solution – User Defined Exception

Click to Continue

Step 1: Create Exception classes



```
package com.demo.exception;

public class MedicineNameException extends Exception{

    public MedicineNameException()
    {
        super();
    }

    public MedicineNameException(String message)
    {
        super(message);
    }

    public MedicineNameException(Throwable throwable)
    {
        super(throwable);
    }
}
```

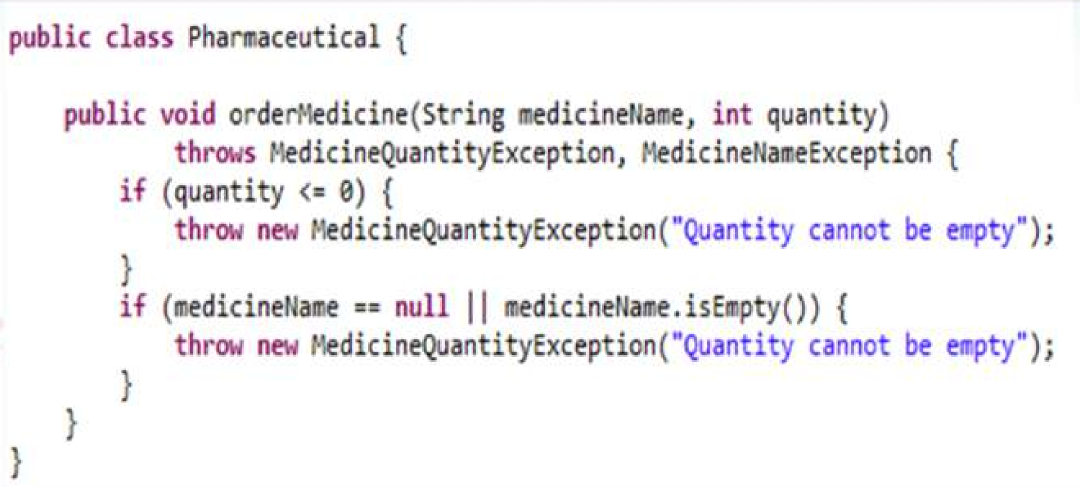


```
package com.demo.exception;

public class MedicineQuantityException extends Exception{

    public MedicineQuantityException()
    {
        super();
    }

    public MedicineQuantityException(String message)
    {
        super(message);
    }

    public MedicineQuantityException(Throwable throwable)
    {
        super(throwable);
    }
}
```

Try it out Solution – User Defined Exception

Click to Continue

Step 2: Create a Pharmaceutical class with orderMedicine method. Add a main method and invoke the orderMedicine method.



```
public class Pharmaceutical {

    public void orderMedicine(String medicineName, int quantity)
            throws MedicineQuantityException, MedicineNameException {
        if (quantity <= 0) {
            throw new MedicineQuantityException("Quantity cannot be empty");
        }
        if (medicineName == null || medicineName.isEmpty()) {
            throw new MedicineQuantityException("Quantity cannot be empty");
        }
    }
}
```

Execute the Pharmaceutical program for the following scenarios.

Name	Quantity	Output
Crocin	0	*MedicineQuantityException* should be thrown.
Pass a null string object	10	*MedicineNameException* should be thrown.

Country