Lab Exercise: Exceptions

INTERTECH

Instructors Who Consult. | **Consultants** Who Teach.

About Intertech

Thank you for choosing Intertech for your training from Udemy. The next page of this document will get you started with your lab if you'd like to skip ahead. Below is a brief overview of Intertech as well as a promo code for you for future live Intertech trainings.

Intertech (www.intertech.com) is the largest combined software developer **training** and **consulting** firm in Minnesota. Our unique blend of training, consulting, and mentoring has empowered technology teams in small businesses, Fortune 500 corporations and government agencies since 1991.

Our training organization offers live in-classroom and online deliveries, private on-site deliveries, and on-demand options such as the course in which you've enrolled from Udemy. We cover a broad spectrum of .NET, Java, Agile/Scrum, Web Development, and Mobile technologies. See more information on our training and search for courses by **clicking here**.

As a Udemy customer, you can save on any live in-classroom, live online, or onsite training with Intertech as well. Just use promo code "Udemy_Labs" when enrolling **and** save 35% on the course price.

We appreciate you choosing Intertech on Udemy for this course!

Lab Exercise

Exceptions

Exception handling is a normal and required part of Java coding. In this lab, you retrofit parts of your application to use Java exceptions where they just use System.out to alert right now. System.out is not something that the application or a user can effectively use to take action when a problem occurs.

Specifically, in this lab you will:

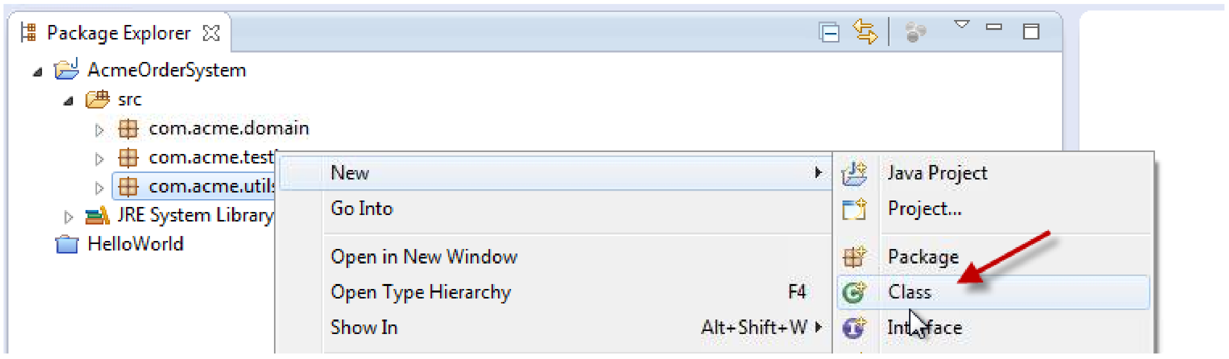
- Create an exception class and use it to create an exception object
- See how to throw the exception object when the application detects a problem or potential problem
- Learn how to use try-catch blocks to capture and react to problems
- Learn how to rethrow an exception as another way to handle an exception

Scenario

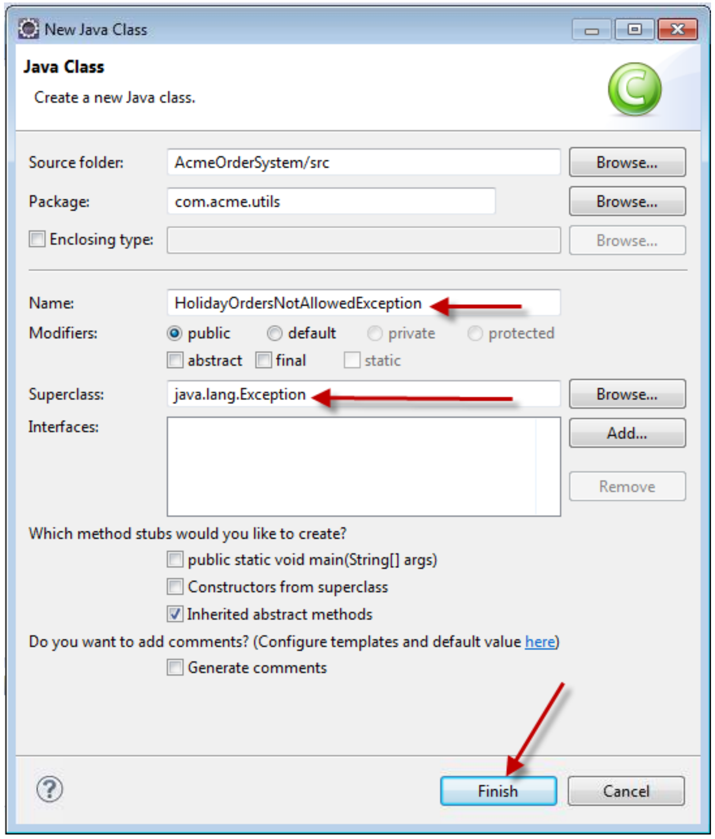
There are several places in the Acme Order System that check for valid data. For example, in the MyDate class, the valid( ) method checks to ensure proper day, month, and year data are supplied when creating or updating a MyDate instance. In the Order class, orders are not allowed to be created with order dates of holidays. However, in both of these examples, when bad data is provided, the application does not stop or otherwise return an indication of problem to the offending method caller. Instead, the data is just not accepted. This might be a worse bug than actually having the application fail! In this lab, you create a new Exception class and have an instance of this exception thrown to the caller if an Order is created with a holiday date as its order date.

Step 1: Create the HolidayOrdersNotAllowedException

1.1 Create HolidayOrdersNotAllowedException. Right-click on the com.acme.utils package and select ***New > Class***.



In the resulting New Java Class window, enter HolidayOrdersNotAllowedException, and change the Superclass from java.lang.Object to java.lang.Exception. Click the ***Finish*** button when complete.



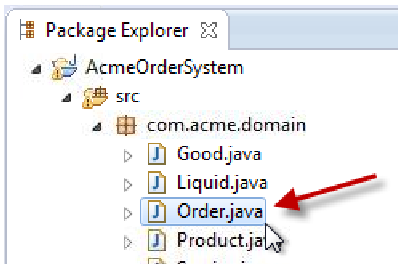
Source folder: AcmeOrderSystem/src
Package: com.acme.utils
☐ Enclosing type:
Name: HolidayOrdersNotAllowedException
Modifiers: [✓ public] default private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Exception
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

1.2 Add a constructor to the new exception class. In the HolidayOrdersNotAllowedException Java editor that opens after creating the class, add a constructor to the class. The constructor should be passed the offending date and use a call to the super constructor to create the appropriate message.

```
public HolidayOrdersNotAllowedException(MyDate date){
  super("Orders are not allowed to be created on: " + date);
}
```

Step 2: Use the new HolidayOrdersNotAllowedException

2.1 Modify the setOrderDate( ) method in Order. In the Package Explorer view, double-click on the Order.java file in the com.acme.domain package to open the file in a Java editor.



Modify the setOrderDate( ) method to throw a HolidayOrdersNotAllowedException when the suggested date is a holiday.

2.1.1 This requires that the method declaration have a "throws HolidayOrdersNotAllowedException".

```
public void setOrderDate(MyDate orderDate) throws
HolidayOrdersNotAllowedException {
```

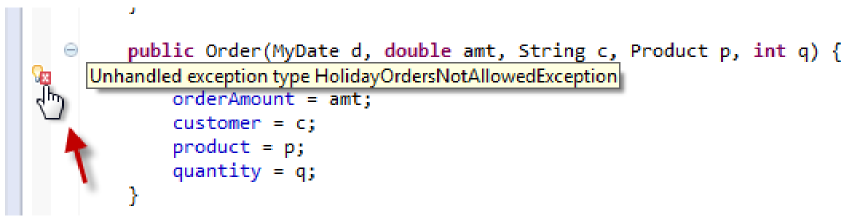
2.1.2 Throwing the exception will require that the Order class imports com.acme.utils.HolidayOrdersNotAllowedException.

```
import com.acme.utils.HolidayOrdersNotAllowedException;
```

2.1.3 Modify the body of the if-conditional to create and throw a new HolidayOrdersNotAllowedException when the new orderDate is determined to be a holiday.

```
    if (isHoliday(orderDate)) {
      System.out.println("Order date, " + orderDate
          + ", cannot be set to a holiday!");
      throw new HolidayOrdersNotAllowedException(orderDate);
    } else {
      this.orderDate = orderDate;
    }
```

2.1.4 Save the new HolidayOrdersNotAllowedException and the Order classes. This should result in a compiler error in the Order constructor. As HolidayOrdersNotAllowedException is a checked exception, it must be handled by all callers of setOrderDate( ). Currently, the constructor does not handle this exception.



```
public Order(MyDate d, double amt, String c, Product p, int q) {
Unhandled exception type HolidayOrdersNotAllowedException
        orderAmount = amt;
        customer = c;
        product = p;
        quantity = q;
    }
```

2.2 Update the constructor. The constructor calls on a method that may throw a checked exception, and so it must either rethrow or try-catch and handle the exception. Surround the offending part of the constructor with a try-catch block. In the catch block, display an appropriate error and exit the system with a call to System.exit(0).

```
public Order(MyDate d, double amt, String c, Product p, int q) {
  try {
    setOrderDate(d);
  } catch (HolidayOrdersNotAllowedException e){
    System.out.println("The order date for an order cannot be a
      holiday!  Application closing.");
    System.exit(0);
  }
  orderAmount = amt;
  customer = c;
  product = p;
  quantity = q;
}
```

Note: Calling System.exit(0) is rarely an appropriate way to handle an exception. Handling an exception usually requires the exception to be logged, informing the user of the issue, and then seeking appropriate action from the user (in this case perhaps picking a new order date).

2.3 Save your changes to the Order class. Make sure there are no compiler errors in any class. Run TestOrders. The application should inform you when an illegal order is created and then stop.

```
...
The tax for this order is: 60.0
The total bill for: 125 ea. Acme Balloon-1401 that is 375.0
CUBIC_FEET in size for Bugs Bunny is 1040.0
Order date, 1/1/2012, cannot be set to a holiday!
The order date for an order cannot be a holiday!  Application
closing.
```

Lab Solutions

Order.java

```
package com.acme.domain;

import com.acme.utils.HolidayOrdersNotAllowedException;
import com.acme.utils.MyDate;

public class Order {
  private MyDate orderDate;
  private double orderAmount = 0.00;
  private String customer;
  private Product product;
  private int quantity;

  public MyDate getOrderDate() {
    return orderDate;
  }

  public void setOrderDate(MyDate orderDate)
      throws HolidayOrdersNotAllowedException {
    if (isHoliday(orderDate)) {
      System.out.println("Order date, " + orderDate
          + ", cannot be set to a holiday!");
      throw new HolidayOrdersNotAllowedException(orderDate);
    } else {
      this.orderDate = orderDate;
    }
  }

  public double getOrderAmount() {
    return orderAmount;
  }

  public void setOrderAmount(double orderAmount) {
    if (orderAmount > 0) {
      this.orderAmount = orderAmount;
    } else {
      System.out
          .println("Attempting to set the orderAmount to a value
less
             than or equal to zero");
    }
  }

  public String getCustomer() {
    return customer;
  }

  public void setCustomer(String customer) {
    this.customer = customer;
  }

  public Product getProduct() {
    return product;
  }

  public void setProduct(Product product) {
    this.product = product;
  }

  public int getQuantity() {
    return quantity;
  }

  public void setQuantity(int quantity) {
    if (quantity > 0) {
      this.quantity = quantity;
    } else {
      System.out
          .println("Attempting to set the quantity to a value
less
              than or equal to zero");
    }
  }

  public static double getTaxRate() {
    return taxRate;
  }

  public static double taxRate = 0.05;

  public static void setTaxRate(double newRate) {
    taxRate = newRate;
  }

  public static void computeTaxOn(double anAmount) {
    System.out.println("The tax for " + anAmount + " is: " +
anAmount
        * Order.taxRate);
  }

  public Order(MyDate d, double amt, String c, Product p, int q)
{
    try {
      setOrderDate(d);
    } catch (HolidayOrdersNotAllowedException e) {
      System.out
          .println("The order date for an order cannot be a
holiday!
             Application closing.");
      System.exit(0);
    }
    orderAmount = amt;
    customer = c;
    product = p;
    quantity = q;
  }

  public String toString() {
    return quantity + " ea. " + product + " for " + customer;
  }

  public double computeTax() {
    System.out.println("The tax for this order is: " +
orderAmount
        * Order.taxRate);
    return orderAmount * Order.taxRate;
  }

  public char jobSize() {
    if (quantity <= 25) {
      return 'S';
    } else if (quantity <= 75) {
      return 'M';
    } else if (quantity <= 150) {
      return 'L';
    }
    return 'X';
  }

  public double computeTotal() {
    double total = orderAmount;
    switch (jobSize()) {
    case 'M':
      total = total - (orderAmount * 0.01);
      break;
    case 'L':
      total = total - (orderAmount * 0.02);
      break;
    case 'X':
      total = total - (orderAmount * 0.03);
      break;
    }
    if (orderAmount <= 1500) {
      total = total + computeTax();
    }
    return total;
  }

  private boolean isHoliday(MyDate proposedDate) {
    for (int x = 0; x < MyDate.getHolidays().length; x++) {
      MyDate holiday = (MyDate) MyDate.getHolidays()[x];
      if ((holiday.getDay() == proposedDate.getDay())
          && (holiday.getMonth() == proposedDate.getMonth())
          && (holiday.getYear() == proposedDate.getYear())) {
        return true;
      }
    }
    return false;
  }
}
```

HolidayOrderNotAllowedException.java

```
package com.acme.utils;

public class HolidayOrdersNotAllowedException extends Exception
{

  public HolidayOrdersNotAllowedException(MyDate date){
    super("Orders are not allowed to be created on: " + date);
  }
}
```

Bonus Lab

Step 3: Handle invalid MyDates

You added a valid( ) method to the MyDate class, but in the event that someone tries to create an invalid date by setting an inappropriate day, month, or year, the new information is just ignored. Instead, an exception should be thrown so that the user of MyDate is aware of the potential problem and handles it.

3.1 Modify the valid( ) method. Modify the valid( ) method on MyDate to return void (instead of a boolean) and to throw an exception whenever an invalid date is created.

3.2 Update the callers of MyDate to rethrow the exception whenever a user tries to create an inappropriate MyDate. What ripple effect does this have on your code? Does this exercise help you to understand why it is usually a good idea to handle exceptions as locally as possible?

3.3 Test your changes. Test your changes by running each of the affected test classes (TestMyDate, TestOrders, and PassByExperiment). You may have to change some data in order for the tests to complete successfully.

Give Intertech a call at **1.800.866.9884** or visit Intertech's website.