

# COSC 2P05 Assignment 2

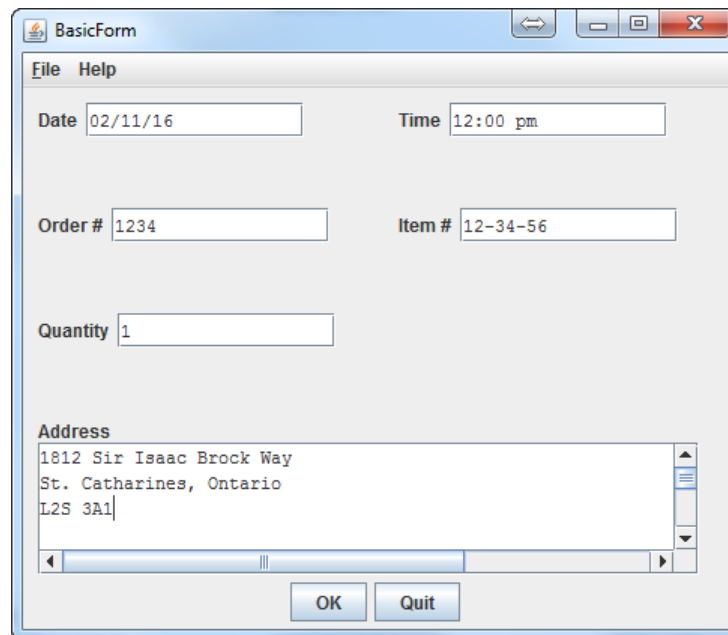
Winter 2015/16

**Due: Feb. 29 @ 10:00 am (Late Date: Mar. 3 @10:00 am).**

In preparation for this assignment, create a folder called `Assign_2` for the assignment. The objective of this assignment is to apply exceptions in a problem solution.

## Acme Distributing

Acme Distributing has a call center which takes orders from customers. The call center clerk answers a call and takes the order by filling in an order form such as:



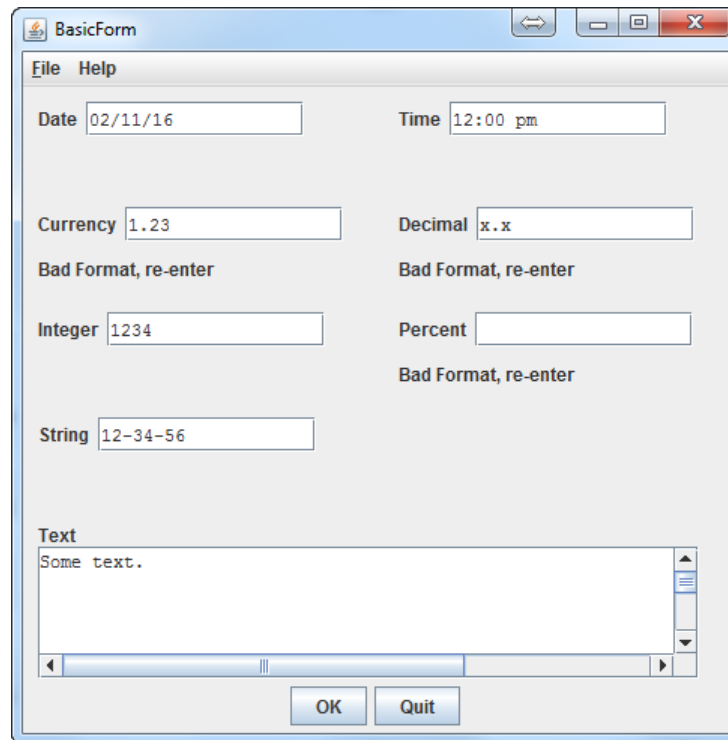
The screenshot shows a Java Swing window titled "BasicForm" with a standard Mac OS X-style title bar (red, yellow, green buttons). The window contains a menu bar with "File" and "Help". Below the menu bar are several text input fields: "Date" with the value "02/11/16", "Time" with the value "12:00 pm", "Order #" with the value "1234", and "Item #" with the value "12-34-56". Below these is a "Quantity" field with the value "1". At the bottom is a large text area labeled "Address" containing the text "1812 Sir Isaac Brock Way", "St. Catharines, Ontario", and "L2S 3A1". At the very bottom of the window are two buttons: "OK" and "Quit".

When the clerk presses OK, the order is placed into a queue to be handled by the warehouse.

## Part A

As part of a library package `Forms`, write a Java class based on `BasicIO` called `CheckedForm` that allows the client to create a form with a number of fields and associate a data type with each field. When the client code presents the form to the user (`accept`), control will not return to the client code until all fields have been filled with data of the correct type, or the user has pressed `Quit`. Whenever the user enters data of an incorrect type, the form is redisplayed and the erroneous field is flagged as requiring fixing.

For example, during an interaction with the user, the form might look like:

The image shows a Java Swing window titled "BasicForm". It has a menu bar with "File" and "Help". The main area contains several input fields arranged in a grid. The first row has "Date" with the value "02/11/16" and "Time" with the value "12:00 pm". The second row has "Currency" with the value "1.23" and "Decimal" with the value "x.x". Below these are two labels: "Bad Format, re-enter" for both the currency and decimal fields. The third row has "Integer" with the value "1234" and "Percent" which is empty. Below the percent field is another "Bad Format, re-enter" label. The fourth row has a "String" field with the value "12-34-56". At the bottom of the main area is a "Text" area containing the text "Some text.". At the very bottom of the window are two buttons: "OK" and "Quit".

CheckedForm should use BasicForm from the Brock BasicIO library as underlying support. The fields subject to checking are implemented as TextFields and read from the BasicForm as String. In addition, a text area is implemented as a TextArea and is not subject to checking. The supported formats are 1) DATE in SHORT format, 2) TIME in SHORT format, 3) CURRENCY, 4) DECIMAL, 5) INTEGER, 6) PERCENT, and 7) STRING. Data entered in each must match the appropriate format for the type as defined by the java.text format classes DateFormat and NumberFormat with the exception of STRING which is unchecked.

The method:

```
public void addField ( String name, String label, Type kind )
```

adds a field of the specified Type (an enumerated type) to the form.

The method:

```
public void addText ( String name, String label )
```

adds a (unchecked) text area. The layout is under control of the CheckedForm and fields are added left to right, top to bottom with the specified name and label (see BasicForm).

The form is presented to the user via the accept method. When the user presses OK, each of the **checked** fields is verified as having the correct format (note: all fields must be filled). If not, a message is displayed beneath the field to prompt the user to fix it. Once the user presses OK with valid fields or presses Quit, the accept method returns with the button number.

The data entered in the fields may be read via the method:

```
public Object readField ( String name )
```

which guarantees to return a valid object of the appropriate type (`Date` for date and time, `Number` for numeric values and `String` for strings.) if the OK button is pressed. The implementation should use `Format` objects to check the format and interpret the values of the fields and catch `ParseException` to detect data errors.

Texts are read via the method:

```
public String readText ( String name )
```

which reads all the text in the text area and returns it as a single string. (Note: use `readC` from `BasicForm`).

Test your implementation of `CheckedForm`.

## Part B

Write a prototype implementation of order handling for Acme Distributing using `CheckedForm`. The program should read the current order queue from disk, repeatedly (until the user hits `Quit`), present the order form and log the order into the queue. The queue should be written to disk before the program shuts down.

You may use a library class for the queue and you should read/write the queue as a persistent object.

The program should handle exceptions to make it foolproof and failsoft. In particular, it should handle the situation where the file does not contain objects of the correct type, prompting for a new file selection; where the queue becomes full and cannot handle a new request alerting the clerk and saving the log queue and for any other exception it should alert the clerk, save the queue if possible, and shut down gracefully. Data entered into the form should be validated.

## Submission

Write a simple program to read the order queue file and print the entries to verify that the data is being added. Test your program in a variety of situations to ensure exception handling is working properly. Introduce an unexpected exception (e.g. do a divide by zero somewhere) to verify that it is failsoft.

Your submission will be in two parts: electronic and paper. The paper submission should include the listings of all Java files you have written. Print the contents of the queue file before and after execution as part of the paper submission.

In addition to your paper submission, you must make an electronic submission of the assignment. You should have a folder for the assignment (`Assign_2`). This folder should include what is needed for the marker to run your program (as indicated below). Zip this folder as `Assign_2.zip`. Log on to Sakai and select the COSC 2P05 site. On the `Assignments` page

select Assignment 2. Attach your .zip file (e.g. Assign\_2.zip) to the assignment submission (use the Add Attachments button and select Browse. Navigate to where you stored your assignment and select the .zip file (e.g. Assign\_2.zip). The file will be added to your submission. Be sure to read and check the Honor Pledge checkbox. Press Submit to submit the assignment. You should receive a confirmation email.

The complete paper submission should be placed in an envelope to which a completed coversheet (<http://www.cosc.brocku.ca/coverpage>) is attached. The submission should be made to the COSC 2P05 submission box outside J328 in accordance with the assignment guidelines posted on the 2P05 Sakai site and not in violation of the regulations on plagiarism

(<http://www.brocku.ca/webcal/2015/undergrad/areg.html#sec67>, <http://www.cosc.brocku.ca/about/policies/plagiarism>). **Note:** Assignments not including a coversheet will **not** be marked.

## DrJava

The .zip folder you submit should contain the project folder including all files relevant to the project—the .drjava, .java and .class files for the assignment.

## Other Platforms

If you are using an IDE other than DrJava to prepare your assignment, you must include the .java source files and the .pdf files described above as well as a file (likely .class or .jar) that will execute on the lab machines. It is your responsibility to ensure that the marker can execute your program.