Criterion C: Development

Outline of Complex Techniques and Tools

File Organization

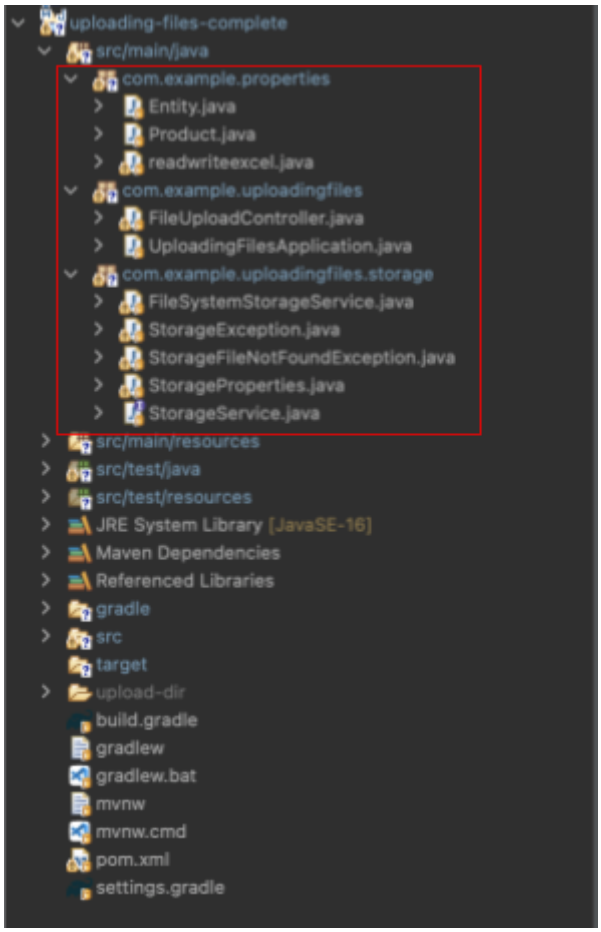


*Figure 1. File Organization (*Note: Red boxed files are the most important files)*

Package com.example.properties - Main package with main files “Entity.java”, “Product.java”, “readwriteexcel.java”

Entity.java - the superclass for Product.java

Readwriteexcel.java - the main class used to modify and read the excel file.

FileUploadController.java - Controls the upload of the file (com.example.uploadingfiles)

UploadingFilesApplication.java - Handles running the FileUploadController.java (com.example.uploadingfiles)

All files under uploadingfiles.storage package handles file uploading (more details below)

Imported Libraries

```
import java.io.FileInputStream;

import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.text.DecimalFormat;
import java.time.temporal.ChronoUnit;

import org.apache.poi.ss.usermodel.Comment;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.CellType;
import org.apache.poi.ss.usermodel.CreationHelper;
import org.apache.poi.ss.usermodel.Drawing;
import org.apache.poi.ss.usermodel.RichTextString;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;
import java.util.Comparator;
```

Figure 2. Imported Libraries (Spring.io)

time.LocalDate - Used to determine the date at any time and is formatted using the format.DateTimeFormatter into the standard time form used (MM/dd/yy)

Apache.poi - Used to modify and create the modified excel sheet.

List of Techniques

1. Inheritance
2. Object arrays w/ user defined objects
3. Sorting algorithm
4. Additional libraries (seen above)
5. Polymorphism

IDE Used

Spring Tool Framework IDE was used as I needed to implement Spring to connect with the java backend of my excel sheet modifier. I decided on this specific IDE because its features closely resembled and are linked to Eclipse which is the normal IDE that I use in class. In other words, it was convenient and a tool I was already familiar with.

Demonstration of Complex Techniques and Tools Used

Inheritance

Inheritance – A parent object/class holds data and attributes that can then be inherited from a child class/object. The usage of inheritance allows for more efficiency in the code as well as eliminating the repetition of certain attributes that needed to be used in other classes. In this project, inheritance was used in the *Entity* and *Product* classes. The variables “ID” (changed to RD in *Product* class) and “Country” were inherited from the *Entity* class.

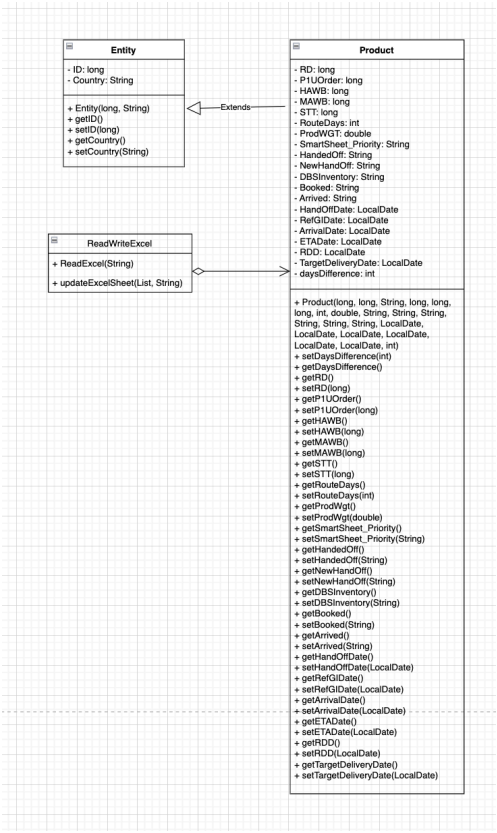


Figure 3. UML diagram illustrating the inheriting of entity from product

```
public class Product extends Entity{
    private long RD;
    private long P1UOrder;
    private long HAWB;
    private long MAWB;
    private long STT;
    private int RouteDays;
    private double ProdWgt;
    private String SmartSheet_Priority;
    private String HandedOff;
    private String NewHandOff;
    private String DBSInventory;
    private String Booked;
    private String Arrived;
    private LocalDate HandOffDate;
    private LocalDate RefGIDate;
    private LocalDate ArrivalDate;
    private LocalDate ETADate;
    private LocalDate RDD;
    private LocalDate TargetDeliveryDate;
    private int daysDifference;
```

The above code block demonstrates the usage of inheritance through the *Product* class inheriting some attributes and properties from *Entity*.

The *Entity* class was originally made to contain those two variables due to an *Employee* class, however the client decided against this *Employee* class later in the process. The *Entity* class was still kept in, in the event that the client did want to add in Employee security in an *Employee* class.

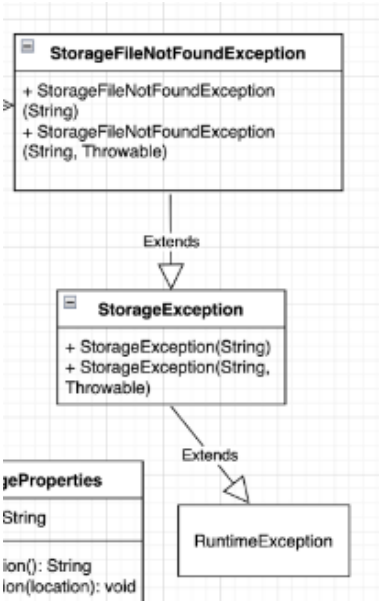
```
public Product(long RD, long P1UOrder, String country, long HAWB, long MAWB, long STT, int RouteDays,
        double ProdWgt, String SmartSheet_Priority, String HandedOff, String NewHandOff,
        String DBSInventory, String Booked, String Arrived, LocalDate HandOffDate,
        LocalDate RefGIDate, LocalDate ArrivalDate, LocalDate ETADate, LocalDate RDD,
        LocalDate TargetDeliveryDate, int daysDifference) {
    super(RD, country); // inheriting ID and country from Entity class
```

Inheritance was also used in classes that specifically looked at error exceptions. For example the *StorageException* class extends the *RuntimeException* from Java as seen below.

```java
public class StorageException extends RuntimeException { // RuntimeException comes from java library

    public StorageException(String message) {
        super(message); // error message
    }

    public StorageException(String message, Throwable cause) {
        super(message, cause); // error message and cause
    }
}
```

Another case of inheritance is used in another class of storage exceptions called *StorageFileNotFoundException*. This class extends the *StorageException* class. (“Spring Projects”)

```java
public class StorageFileNotFoundException extends StorageException {

    public StorageFileNotFoundException(String message) {
        super(message); // error message
    }

    public StorageFileNotFoundException(String message, Throwable cause) {
        super(message, cause); // error message and cause
    }
}
```



This image above provides an overview of the usage of inheritance used in this project.

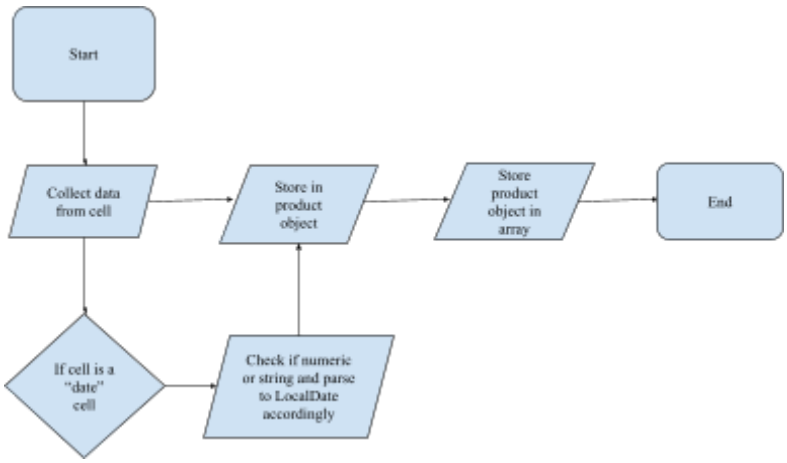
Object arraylist

Object arraylist – A data structure consisting of objects that are a part of a certain class.

```
public List<Product> ReadExcel(String filePath) {
    List<Product> products = new ArrayList<>(); // creating array for objects: type product
    try {
        FileInputStream fis = new FileInputStream(filePath); // initialise input stream
        Workbook wb = WorkbookFactory.create(fis); // create spreadsheet maker
        Sheet sheet = wb.getSheet("Sheet1"); // find "Sheet1" in excel sheet to input information into
```

Figure 4. Object arraylist and creating excel sheet

Arraylist of objects that represent the products being put into the excel sheet. Each product object carries attributes that modify the excel sheet when sorted. The storage of the objects into the arraylist works as follows:



As seen above, the data is collected from each cell and stored into the product object which is then stored into the product arraylist. There is a special case however if there is a date variable as the date variable needs to be parsed into the correct format.

```
int row_index = 1; // start from row 1
Row row;
while ((row = sheet.getRow(row_index)) != null) { // check that the row is not empty before collecting data
    long RD = (long) row.getCell(0).getNumericCellValue(); // store data in variable from that specific cell
    long PlUOrder = (long) row.getCell(1).getNumericCellValue();
    String Country = row.getCell(2).getStringCellValue();
    long HAWB = (long) row.getCell(3).getNumericCellValue();
    long MAWB = (long) row.getCell(4).getNumericCellValue();
    long STT = (long) row.getCell(5).getNumericCellValue();
    int RouteDays = (int) row.getCell(6).getNumericCellValue();
    double ProdWgt = (double) row.getCell(7).getNumericCellValue();
    String SmartSheet_Priority = row.getCell(8).getStringCellValue();
    String HandedOff = row.getCell(9).getStringCellValue();
    String NewHandOff = row.getCell(10).getStringCellValue();
    String DBSInventory = row.getCell(11).getStringCellValue();
    String Booked = row.getCell(12).getStringCellValue();
    String Arrived = row.getCell(13).getStringCellValue();
```

The above image shows the retrieving of cell values from the original spreadsheet.

```
Cell HandOffDateCell = row.getCell(14);
LocalDate HandOffDate = null;
// Date cells need to be checked if they are type string or number in order to parse correctly
if (HandOffDateCell.getCellType() == CellType.STRING) {
    String HandOffDateStr = HandOffDateCell.getStringCellValue();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy"); //parsing
    HandOffDate = LocalDate.parse(HandOffDateStr, formatter);
} else if (HandOffDateCell.getCellType() == CellType.NUMERIC) {
    HandOffDate = HandOffDateCell.getLocalDateTimeCellValue().toLocalDate();
}
```

Figure 5. Parsing of date variable

Figure 5 is repeated for each of the date variables that need to be parsed in the excel sheet. For the different variable types (String or numeric), the cells are modified in a different way. In figure 5, a date type value was formatted but an *if statement* was implemented for checking if the cell held a string or date value so date-sorting could be used later on.

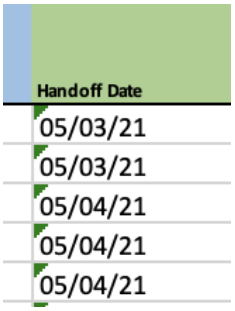


Figure 5.1 Implementation of modification of cell as shown in figure 5

```
// Check if HandOffDate and TargetDeliveryDate are less than 3 days from today
Product product = new Product(RD, PlUOrder, Country, HAWB, MAWB, STT, RouteDays,
        ProdWgt, SmartSheet_Priority, HandedOff, NewHandOff,
        DBSInventory, Booked, Arrived, HandOffDate,
        RefGIDate, ArrivalDate, ETADate, RDDDate,
        TargetDeliveryDate, daysDifference); // creating new object
products.add(product); // adding product to the list of products
```

The above code saves the product objects into the product arraylist for later usage.

```
catch (Exception e) { // exception handling of errors
    if (e instanceof java.lang.IllegalStateException) {
        throw new IllegalStateException("Wrong Variable Type Error", e);
    } else {
        System.out.println("ReadExcel catch block");
        e.printStackTrace();
    }
}
```

Exception handling above is used to make sure that the right variable is in the right column and thus can be parsed and stored correctly into the Product arraylist.

	P1U Order#	Co
27	HELLO	Pe
52	6213204455	Inc
42	6213219214	Ho
52	6213216855	Gu
34	6213219334	Co
71	6213221385	Ko
93	6213230475	Pe
34	6213219117	Jan

For example, the P1UOrder # is supposed to have longs as the variable type. However, if someone were to type a string, a variable type mismatch error would be thrown and thus the sheet would not be modified.

```
/* ------------------------------------------------
 * PURPOSE: to deal with errors in columns and rows
 * ------------------------------------------------
 */
catch (IllegalStateException e) { // redirect to error page with variable type mismatch error
    redirectAttributes.addFlashAttribute("errorTitle", "Variable Type Mismatch Error");
    redirectAttributes.addFlashAttribute("errorMessage", "An error occurred while processing the request.");
    redirectAttributes.addFlashAttribute("errorDetails", e.getMessage());

    return "redirect:/error";
}
catch (Exception e) {
    return "redirect:/"; // You can redirect to a specific error page if necessary
}
```

Variable Type Mismatch Error

An error occurred while processing the request.

Error Details:

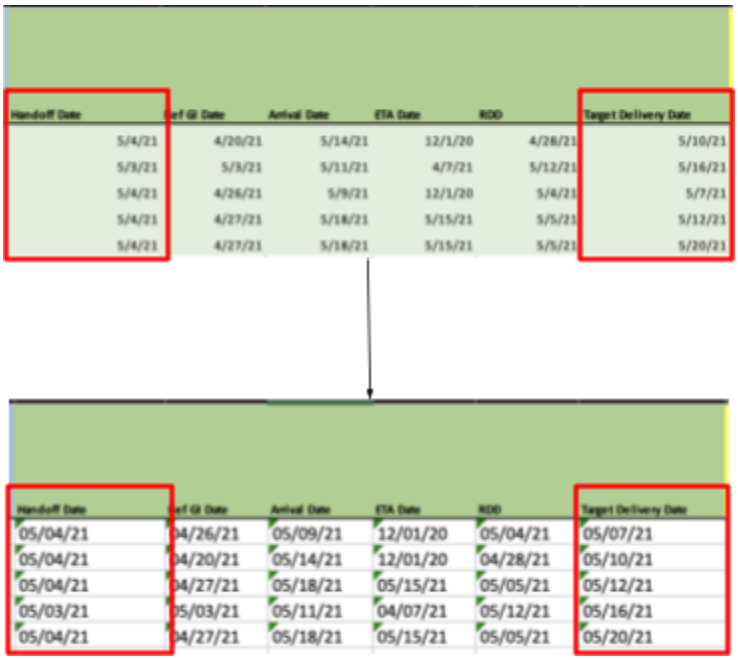
Wrong Variable Type Error

Sorting algorithm

```
            int daysDifference = Math.abs((int) ChronoUnit.DAYS.between(HandOffDate, TargetDeliveryDate)

            // Check if HandOffDate and TargetDeliveryDate are less than 3 days from today
            Product product = new Product(RD, P1UOrder, Country, HAWB, MAWB, STT, RouteDays,
                    ProdWgt, SmartSheet_Priority, HandedOff, NewHandOff,
                    DBSInventory, Booked, Arrived, HandOffDate,
                    RefGIDate, ArrivalDate, ETADate, RDDDate,
                    TargetDeliveryDate, daysDifference); // creating new object
            products.add(product); // adding product to the list of products

            row_index++;
        }

        Collections.sort(products, Comparator
                .comparing(Product::getHandOffDate).thenComparing(Product::getDaysDifference));
        // compare and sort the products based off of dates from top priority to least

        FileOutputStream fos = new FileOutputStream(filePath); // opening the file stream
        wb.write(fos); // writing to the excel file
        fis.close();
        fos.close(); // closing the file stream
    } catch (Exception e) {
        System.out.println("ReadExcel catch block"); // catching exceptions
        e.printStackTrace();
    }
```

Figure 6. Sorting Algorithm Implementation

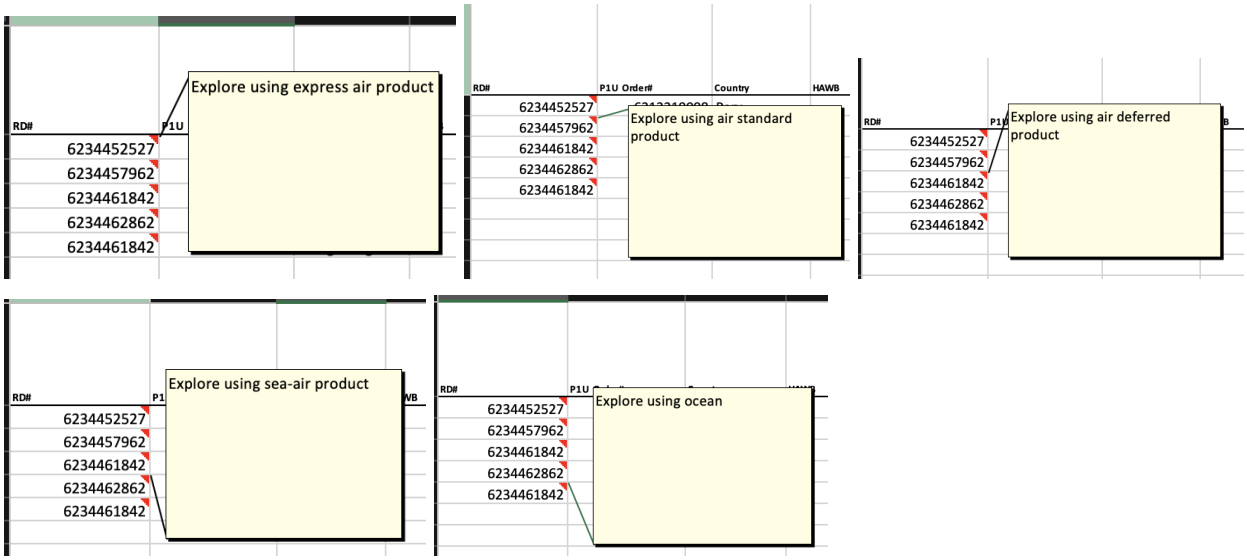
Figure 6 shows a collections.sort which uses a MergeSort algorithm to sort the dates in order. The MergeSort algorithm proved to be very efficient when sorting the items and did not take a long time. Collections.sort was also much easier to use as it used built in methods from other classes that java has and worked well with the comparison and sorting methods that the project required.



The sorting algorithm implementation is shown here in the UI where the excel file modifies the products based on the difference in HandOffDate and TargetDeliveryDate.
In addition to the sorting algorithm, the products also need to have comments written on them based on the difference in date.

```
/* ------------------------------------------------------------
/ PURPOSE: to write comments for each difference in day
  ------------------------------------------------------------*/

int daysDifference = product.getDaysDifference();
if (daysDifference < 4) { // day difference less than 4
    Comment expressAirComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString expressAirText = factory.createRichTextString("Explore using express air product");
    expressAirComment.setString(expressAirText);
    row.getCell(0).setCellComment(expressAirComment);
}
if (daysDifference >= 4 && daysDifference < 7) { // day difference between 4 and 7
    Comment airStandardComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString airStandardText = factory.createRichTextString("Explore using air standard product");
    airStandardComment.setString(airStandardText);
    row.getCell(0).setCellComment(airStandardComment);
}

if (7 <= daysDifference && daysDifference < 10) { // day difference between 7 and 10
    Comment airDeferredComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString airDeferredText = factory.createRichTextString("Explore using air deferred product");
    airDeferredComment.setString(airDeferredText);
    row.getCell(0).setCellComment(airDeferredComment);
}

if (10 <= daysDifference && daysDifference < 15) { // day difference between 10 and 15
    Comment seaAirComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString seaAirText = factory.createRichTextString("Explore using sea-air product");
    seaAirComment.setString(seaAirText);
    row.getCell(0).setCellComment(seaAirComment);
}

if (daysDifference >= 15) { // day difference greater than 15
    Comment OceanComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString OceanText = factory.createRichTextString("Explore using ocean");
    OceanComment.setString(OceanText);
    row.getCell(0).setCellComment(OceanComment);
}
```

The comment would be seen as a sticky note attached to the cell that the user can view when hovering over it.



As shown below, the comments and the day difference correspond to each other.

Difference:	Handoff Date	Ref GI Date	Arrival Date	ETA Date	RDD	Target Delivery Date
3	05/04/21	04/26/21	05/09/21	12/01/20	05/04/21	05/07/21
6	05/04/21	04/20/21	05/14/21	12/01/20	04/28/21	05/10/21
8	05/04/21	04/27/21	05/18/21	05/15/21	05/05/21	05/12/21
13	05/03/21	05/03/21	05/11/21	04/07/21	05/12/21	05/16/21
16	05/04/21	04/27/21	05/18/21	05/15/21	05/05/21	05/20/21

Polymorphism

Polymorphism - A mix of inherited behaviours that make them unique to their own class. In this project, overriding, a form of polymorphism was used.

As seen in the figure above, the different methods from FileSystemStorageService class (com.example.uploadingfiles.storage) are overriding the methods that are in the interface StorageService. This is good for clarity and also to catch errors at compile-time. Therefore making the code more maintainable and less error prone.

Outline of Algorithmic Thinking

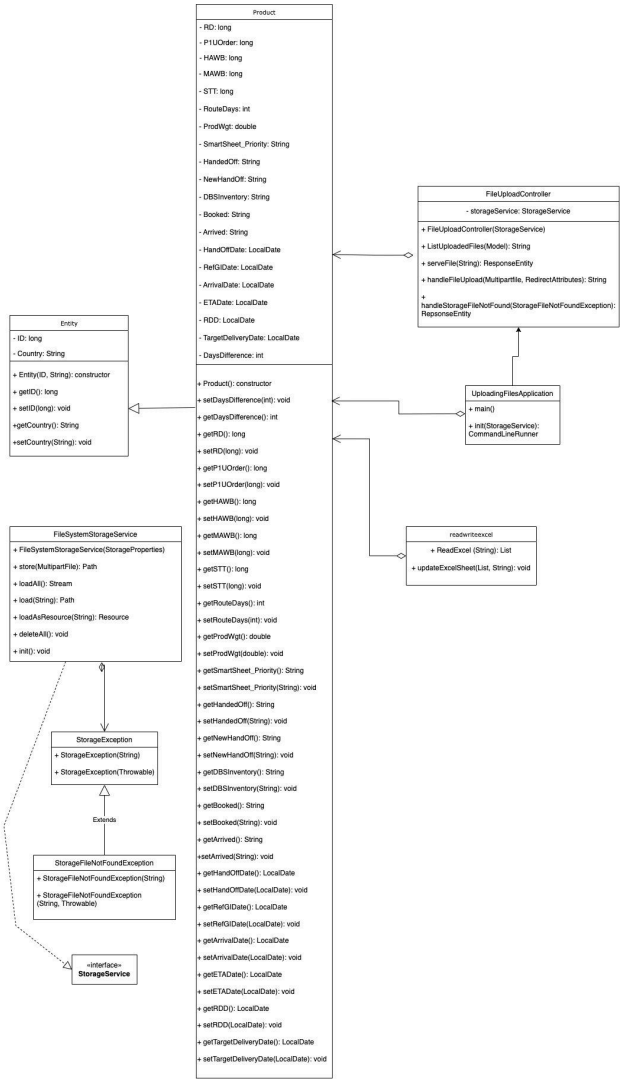


Figure 7. UML Diagram of Classes

Figure 7 gives a brief overview of the classes used and how they correspond with each other. (Link to figure 7 is hyperlinked). Below is a list of algorithmic thinking used:

1. Algorithm of calculating the difference between two time variables after they have been passed.
2. Algorithm of comparing and sorting according to the difference in days of two variables
3. Algorithm to add comments to certain orders depending on priority dates
4. Algorithm to upload and return processed file
5. Algorithm to rewrite the new sorted information into the file

Demonstration of Algorithmic Thinking

Calculating the difference between two time variables after they have been passed

Brief overview:

Figure 8 gives an overview of the whole algorithm. There's two variables called HandOffDate and TargetDeliveryDate which are then subtracted from each other and taken the absolute value of. This is then stored in getDaysDifference in order to be used in a later algorithm (algorithm 2).

Implementation:

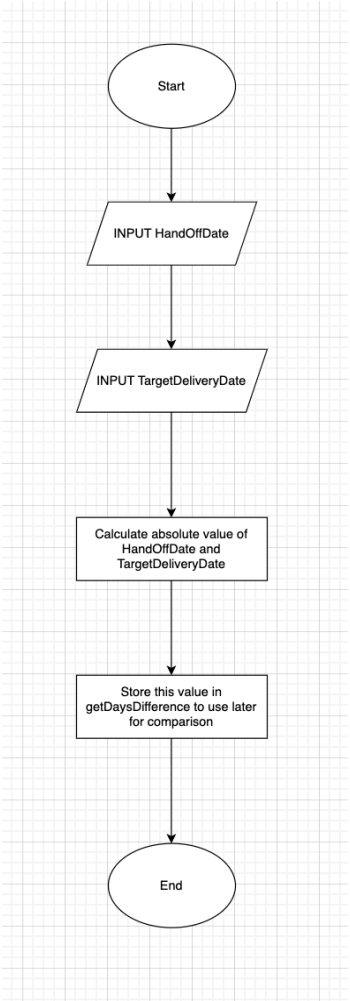


Figure 8. Flowchart demonstrating calculating the difference between two time variables after they have been passed

```java
Cell HandOffDateCell = row.getCell(14);
LocalDate HandOffDate = null;
// Date cells need to be checked if they are type string or number in order to parse correctly
if (HandOffDateCell.getCellType() == CellType.STRING) {
    String HandOffDateStr = HandOffDateCell.getStringCellValue();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy"); //parsing
    HandOffDate = LocalDate.parse(HandOffDateStr, formatter);
} else if (HandOffDateCell.getCellType() == CellType.NUMERIC) {
    HandOffDate = HandOffDateCell.getLocalDateTimeCellValue().toLocalDate();
}
```

```java
Cell TargetDeliveryDateCell = row.getCell(19);
LocalDate TargetDeliveryDate = null;
if (TargetDeliveryDateCell.getCellType() == CellType.STRING) {
    String TargetDeliveryDateStr = TargetDeliveryDateCell.getStringCellValue();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy"); // parsing
    TargetDeliveryDate = LocalDate.parse(TargetDeliveryDateStr, formatter);
} else if (TargetDeliveryDateCell.getCellType() == CellType.NUMERIC) {
    TargetDeliveryDate = TargetDeliveryDateCell.getLocalDateTimeCellValue().toLocalDate();
}

int daysDifference = Math.abs((int) ChronoUnit.DAYS.between(HandOffDate, TargetDeliveryDate));
```

Figure 9. Code showing the implementation of getting the two variables from the sheet, parsing them where necessary and finding the difference

Handoff Date	Ref GI Date	Arrival Date	ETA Date	RDD	Target Delivery Date	R
05/04/21	04/26/21	05/09/21	12/01/20	05/04/21	05/07/21	A
05/04/21	04/20/21	05/14/21	12/01/20	04/28/21	05/10/21	A
05/04/21	04/27/21	05/18/21	05/15/21	05/05/21	05/12/21	A
05/03/21	05/03/21	05/11/21	04/07/21	05/12/21	05/16/21	A
05/04/21	04/27/21	05/18/21	05/15/21	05/05/21	05/20/21	A

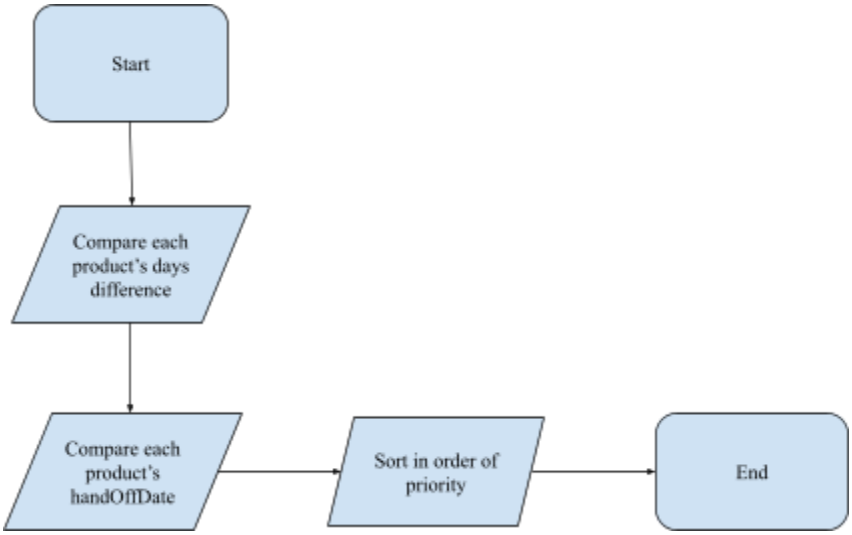
The dates are all modified into a standard month, date, year system.

Comparing and sorting according to the difference in days of two variables

Brief overview:

The comparison and sorting of the two variables needed to be done in order to properly display the products in order of priority in the modified excel sheet. The implementation included the usage of a sorting algorithm (MergeSort, which is described in further detail in the sorting algorithm section).

Implementation:




```
Collections.sort(products, Comparator
        .comparing(Product::getDaysDifference).thenComparing(product -> product.getHandOffDate()));

// Write the sorted products to the sheet
int rowIndex = 1;
for (Product product : products) {
    Row row = sheet.getRow(rowIndex);
    if (row == null) {
        row = sheet.createRow(rowIndex);
    }
```

Figure 10. Code implementation of sorting and comparison of products

A Collections.sort method was used as well as a comparator to compare between the difference of the days in products. Below that, the sorted products were then written back to the modified excel file.

Add comments to certain orders depending on priority dates

Brief overview:

Comments needed to be added to the excel spreadsheet depending on the difference in day. The algorithm for this is shown in the flowchart below:

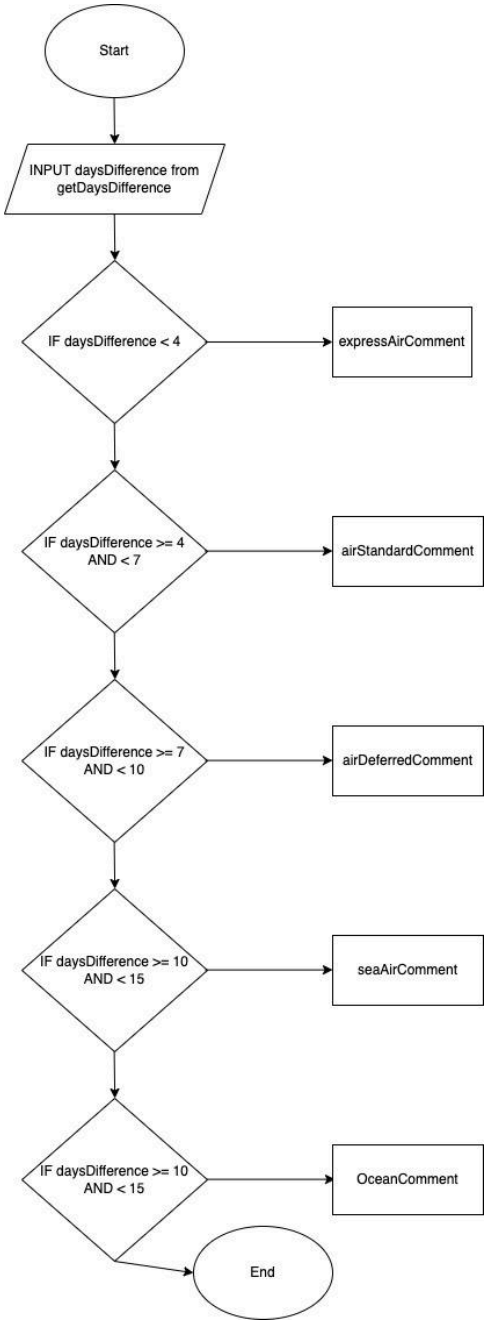


Figure 11. Flowchart representing the planning stage of comment assignments

Implementation

```
int daysDifference = product.getDaysDifference();
if (daysDifference < 4) {
    Comment expressAirComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString expressAirText = factory.createRichTextString("Explore using express air product");
    expressAirComment.setString(expressAirText);
    row.getCell(0).setCellComment(expressAirComment);
}
if (daysDifference >= 4 && daysDifference < 7) {
    Comment airStandardComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString airStandardText = factory.createRichTextString("Explore using air standard product");
    airStandardComment.setString(airStandardText);
    row.getCell(0).setCellComment(airStandardComment);
}

if (7 <= daysDifference && daysDifference < 10) {
    Comment airDeferredComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString airDeferredText = factory.createRichTextString("Explore using air deferred product");
    airDeferredComment.setString(airDeferredText);
    row.getCell(0).setCellComment(airDeferredComment);
}

if (10 <= daysDifference && daysDifference < 15) {
    Comment seaAirComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString seaAirText = factory.createRichTextString("Explore using sea-air product");
    seaAirComment.setString(seaAirText);
    row.getCell(0).setCellComment(seaAirComment);
}

if (daysDifference >= 15) {
    Comment OceanComment = drawing.createCellComment(factory.createClientAnchor());
    RichTextString OceanText = factory.createRichTextString("Explore using ocean");
    OceanComment.setString(OceanText);
    row.getCell(0).setCellComment(OceanComment);
}
```

Figure 12. Code implementation of comment algorithm

The if statements above use the daysDifference who's calculation is seen in Figure 9. The program then determines whether the daysDifference falls between certain integer boundaries. Depending on the boundaries, a different comment would be made and added into the comment cell. This implementation on the UI is seen in the sorting algorithm section.

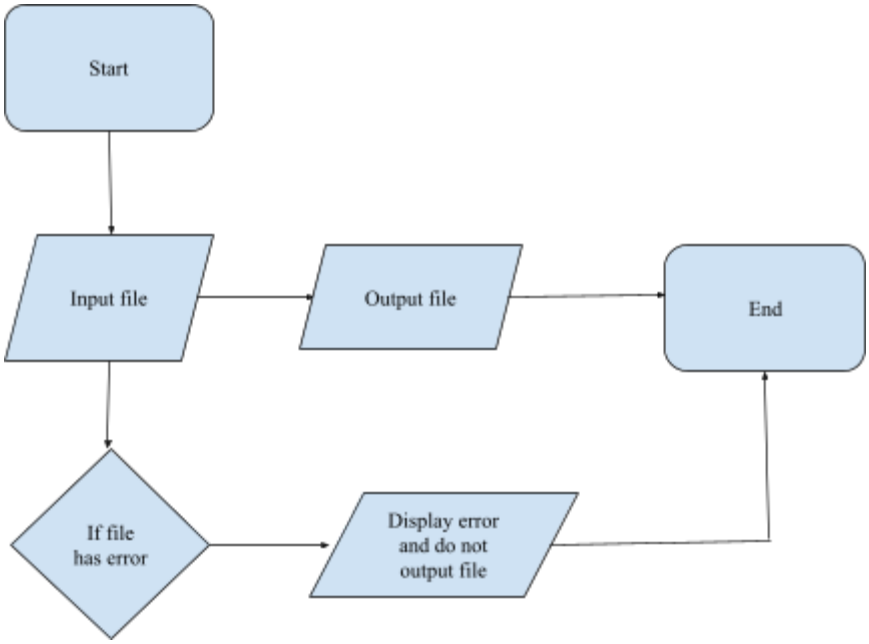
Upload and return processed file

Brief overview:

1. Send path file of modified excel sheet to storage of Spring
2. Spring would then upload the file to file-upload template of HTML file
3. HTML file would allow for user to download the file and open it in excel

This method is a simplified approach of what Spring goes through.

Implementation:




```
public interface StorageService {

    void init();

    Path store(MultipartFile file); // store file

    Stream<Path> loadAll();

    Path load(String filename); // load file in

    Resource loadAsResource(String filename);

    void deleteAll();

}
```

The above code highlights the interface that is being and consists of deleting, storing and loading the file to the correct place.

```
@PostMapping("/")
public String handleFileUpload(@RequestParam("file") MultipartFile file,
        RedirectAttributes redirectAttributes) {

    Path pathFile = storageService.store(file);
    readwriteexcel obj = new readwriteexcel();
    List<Product> products = obj.ReadExcel(pathFile.toString()); // get pathFile name
     obj.updateExcelSheet(products, pathFile.toString()); // modify excel sheet
    redirectAttributes.addFlashAttribute("message",
            "You successfully uploaded " + file.getOriginalFilename() + "!"); // upload file

    return "redirect:/";
}
```

HTML for uploading file

UI for uploading file

```
/* file upload button */
input[type="file"]::file-selector-button {
  border-radius: 4px;
  padding: 0 16px;
  height: 40px;
  cursor: pointer;
  background-color: white;
  border: 1px solid rgba(0, 0, 0, 0.16);
  box-shadow: 0px 1px 0px rgba(0, 0, 0, 0.05);
  margin-right: 16px;
  transition: background-color 200ms;
}
```

CSS for style of button

This style sheet was used solely because it was very simple for the client to use. It allows for easy uploading of files and displays the file name on there to make sure that the correct file is being uploaded.

```
@Autowired
public FileSystemStorageService(StorageProperties properties) {
    this.rootLocation = Paths.get(properties.getLocation());
}
/* ------------------------------------------
 * NOTE: all the override methods override
 * the interface "Storage Properties"
 * ------------------------------------------
 */

@Override
/* ------------------------------------------
 * PURPOSE: to store files
 * ------------------------------------------
 */
public Path store(MultipartFile file) {
    try {
        if (file.isEmpty()) {
            throw new StorageException("Failed to store empty file.");
        }
        Path destinationFile = this.rootLocation.resolve(
                Paths.get(file.getOriginalFilename()))
                .normalize().toAbsolutePath();
        if (!destinationFile.getParent().equals(this.rootLocation.toAbsolutePath())) {
            // This is a security check
            throw new StorageException(
                    "Cannot store file outside current directory.");
        }
        try (InputStream inputStream = file.getInputStream()) {
            Files.copy(inputStream, destinationFile,
                StandardCopyOption.REPLACE_EXISTING);
        }
        return destinationFile;
    }
    catch (IOException e) {
        throw new StorageException("Failed to store file.", e);
    }
}
```

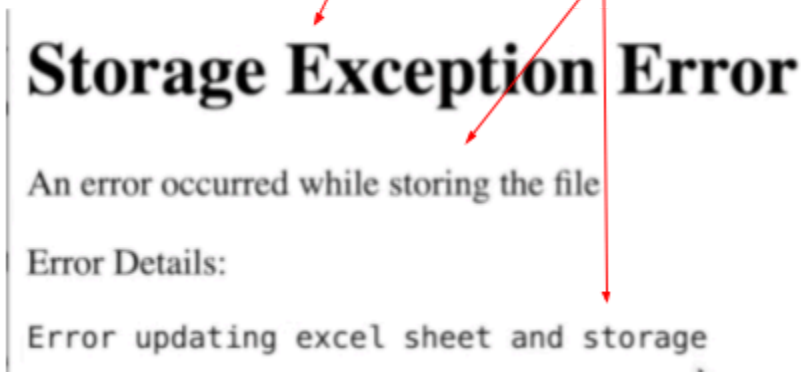
```java
@ExceptionHandler(StorageFileNotFoundException.class) // exception handler for file not found
public ResponseEntity<?> handleStorageFileNotFound(StorageFileNotFoundException exc) {
    return ResponseEntity.notFound().build();
}
```

```java
catch (MalformedURLException e) {
    throw new StorageFileNotFoundException("Could not read file: " + filename, e);
}
```

This error occurs when the URL used to connect to the server is incorrect or improperly formatted. (com.example.uploadingfiles.storage.)

```java
catch (IOException e) { // initialising storage
    throw new StorageException("Could not initialize storage", e);
}
```

```java
catch (StorageException e) { // redirect to error page with storage exception error
    redirectAttributes.addFlashAttribute("errorTitle", "Storage Exception Error");
    redirectAttributes.addFlashAttribute("errorMessage", "An error occurred while storing the file");
    redirectAttributes.addFlashAttribute("errorDetails", e.getMessage());

    return "redirect:/error"; // You can redirect to a specific error page if necessary
}
```

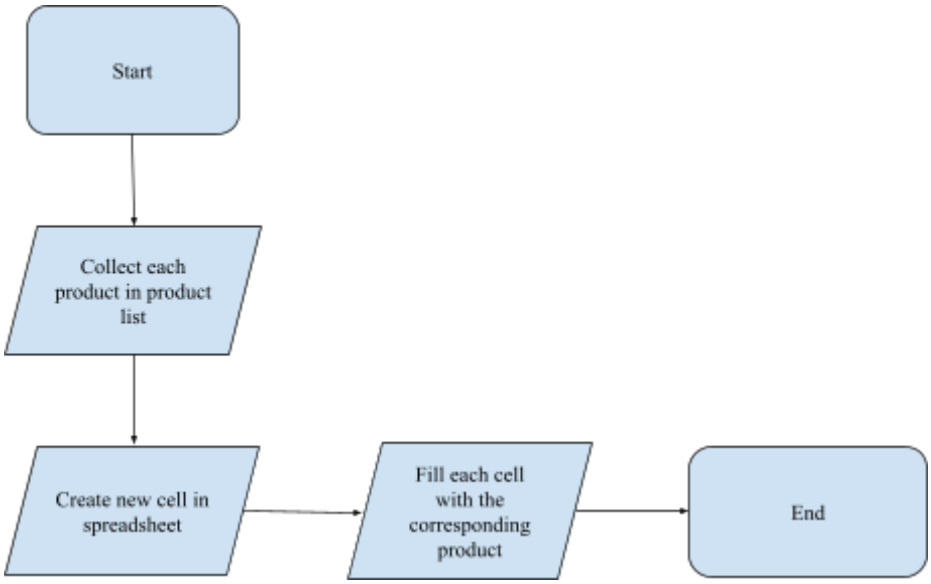


Rewrite the new sorted information into the file

Brief overview:

An algorithm needed to be used in order to rewrite the newly sorted products into the spreadsheet file so when opened, there would be a spreadsheet of products listed in terms of priority.

Implementation:




```
/* ------------------------------------------
 * PURPOSE: rewrite products to spreadsheet
 * ------------------------------------------
 */
int rowIndex = 1;
for (Product product : products) { // iterate through every product in product
    Row row = sheet.getRow(rowIndex);
    if (row == null) {
        row = sheet.createRow(rowIndex);
    }
    row.createCell(0).setCellValue(product.getRD()); // create cell and set to the corresponding product
    row.createCell(1).setCellValue(product.getP1UOrder());
    row.createCell(2).setCellValue(product.getCountry());
    row.createCell(3).setCellValue(product.getHAWB());
    row.createCell(4).setCellValue(product.getMAWB());
    row.createCell(5).setCellValue(product.getSTT());
    row.createCell(6).setCellValue(product.getRouteDays());
    String formattedSTT = decimalFormat.format(product.getProdWgt());
    row.createCell(7).setCellValue(formattedSTT);
    row.createCell(8).setCellValue(product.getSmartSheet_Priority());
    row.createCell(9).setCellValue(product.getHandedOff());
    row.createCell(10).setCellValue(product.getNewHandOff());
    row.createCell(11).setCellValue(product.getDBSInventory());
    row.createCell(12).setCellValue(product.getBooked());
    row.createCell(13).setCellValue(product.getArrived());
    row.createCell(14).setCellValue(product.getHandOffDate().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
    row.createCell(15).setCellValue(product.getRefGIDate().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
    if (product.getArrivalDate()!=null) {
        row.createCell(16).setCellValue(product.getArrivalDate().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
    }
    if (product.getETADate()!=null) {
    row.createCell(17).setCellValue(product.getETADate().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
    }
    row.createCell(18).setCellValue(product.getRDD().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
    row.createCell(19).setCellValue(product.getTargetDeliveryDate().format(DateTimeFormatter.ofPattern("MM/dd/yy")));
```

```
        try (FileOutputStream fos = new FileOutputStream(filePath)) {
            wb.write(fos); // write the new data to the spreadsheet
        }
    } catch (Exception e) {
        if (e instanceof java.lang.IllegalStateException) { // added in wrong variable type error
            throw new IllegalStateException("Wrong Variable Type Error", e);
        }
        else { // added in catch for storage exception
            throw new StorageException("Error updating excel sheet and storage", e);
        }
        //System.out.println("Error updating Excel sheet");
        //e.printStackTrace();
    }
```

Word count: 1200

Works Cited

"Spring Projects." *Spring.io*, 2017, spring.io/projects/spring-framework. Accessed 27 Sept. 2023.

com.example.uploadingfiles. *Spring Framework,* VMware Tanzu, 2016. *Github*, https://github.com/spring-guides/gs-uploading-files/tree/main/complete/src/main/java/com/example/uploadingfiles

com.example.uploadingfiles.storage. *Spring Framework,* VMware Tanzu, 2016. *Github,* https://github.com/spring-guides/gs-uploading-files/tree/main/complete/src/main/java/com/example/uploadingfiles/storage