Appendix B

Entity class -

```
package com.example.properties;
public class Entity {
	private long ID;
	private String Country;

	/* --------------------------
	 * @NOTE: in the event that an employee were to be added,
	 * the ID and country vars can be used in the product class
	 */
	public Entity(long ID, String Country) {
		// TODO Auto-generated constructor stub
		this.setID(ID);
		this.setCountry(Country);
	}
	// @todo: potentially allow for employees to later inherit these
values
	public long getID() { //
		return ID;
	}
	public void setID(long ID) {
		this.ID = ID;
	}
	public String getCountry() {
		return Country;
	}
	public void setCountry(String Country) {
		this.Country = Country;
	}
}
```

Product class

```
package com.example.properties;
import java.time.LocalDate;
public class Product extends Entity{
	/* ----------------------------------
	 * PURPOSE: the product class with vars
```

```
       * containing every named column
       * of the excel spreadsheet
       * ----------------------------------
       */
      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;

      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) {
            // constructor class
            super(RD, country); // inheriting ID and country from Entity
class
            this.setRD(RD);
            this.setP1UOrder(P1UOrder);
            this.setHAWB(HAWB);
            this.setMAWB(MAWB);
            this.setSTT(STT);
            this.setRouteDays(RouteDays);
```

```
            this.setProdWgt(ProdWgt);
            this.setSmartSheet_Priority(SmartSheet_Priority);
            this.setHandedOff(HandedOff);
            this.setNewHandOff(NewHandOff);
            this.setDBSInventory(DBSInventory);
            this.setBooked(Booked);
            this.setArrived(Arrived);
            this.setHandOffDate(HandOffDate);
            this.setRefGIDate(RefGIDate);
            this.setArrivalDate(ArrivalDate);
            this.setETADate(ETADate);
            this.setRDD(RDD);
            this.setTargetDeliveryDate(TargetDeliveryDate);
            this.setdaysDifference(daysDifference);

      }
      public void setdaysDifference(int daysDifference) {
            this.daysDifference = daysDifference;
      }
      public int getDaysDifference() {
            return daysDifference;
      }
      public long getRD() {
            return RD;
      }
      public void setRD(long RD) {
            this.RD = RD;
      }
      public long getP1UOrder() {
            return P1UOrder;
      }
      public void setP1UOrder(long P1UOrder) {
            this.P1UOrder = P1UOrder;
      }
      public long getHAWB() {
            return HAWB;
      }
      public void setHAWB(long HAWB) {
            this.HAWB = HAWB;
      }
      public long getMAWB() {
            return MAWB;
      }
```

```
	public void setMAWB(long MAWB) {
		this.MAWB = MAWB;
	}
	public long getSTT() {
		return STT;
	}
	public void setSTT(long STT) {
		this.STT = STT;
	}
	public int getRouteDays() {
		return RouteDays;
	}
	public void setRouteDays(int RouteDays) {
		this.RouteDays = RouteDays;
	}
	public double getProdWgt() {
		return ProdWgt;
	}
	public void setProdWgt(double ProdWgt) {
		this.ProdWgt = ProdWgt;
	}
	public String getSmartSheet_Priority() {
		return SmartSheet_Priority;
	}
	public void setSmartSheet_Priority(String SmartSheet_Priority) {
		this.SmartSheet_Priority = SmartSheet_Priority;
	}
	public String getHandedOff() {
		return HandedOff;
	}
	public void setHandedOff(String HandedOff) {
		this.HandedOff = HandedOff;
	}
	public String getNewHandOff() {
		return NewHandOff;
	}
	public void setNewHandOff(String NewHandOff) {
		this.NewHandOff = NewHandOff;
	}
	public String getDBSInventory() {
		return DBSInventory;
	}
	public void setDBSInventory(String DBSInventory) {
```

```
            this.DBSInventory = DBSInventory;
      }
      public String getBooked() {
            return Booked;
      }
      public void setBooked(String Booked) {
            this.Booked = Booked;
      }
      public String getArrived() {
            return Arrived;
      }
      public void setArrived(String Arrived) {
            this.Arrived = Arrived;
      }
      public LocalDate getHandOffDate() {
            return HandOffDate;
      }
      public void setHandOffDate(LocalDate HandOffDate) {
            this.HandOffDate = HandOffDate;
      }
      public LocalDate getRefGIDate() {
            return RefGIDate;
      }
      public void setRefGIDate(LocalDate RefGIDate) {
            this.RefGIDate = RefGIDate;
      }
      public LocalDate getArrivalDate() {
            return ArrivalDate;
      }
      public void setArrivalDate(LocalDate ArrivalDate) {
            this.ArrivalDate = ArrivalDate;
      }
      public LocalDate getETADate() {
            return ETADate;
      }
      public void setETADate(LocalDate ETADate) {
            this.ETADate = ETADate;
      }
      public LocalDate getRDD() {
            return RDD;
      }
      public void setRDD(LocalDate RDD) {
            this.RDD = RDD;
```

```
	}
	public LocalDate getTargetDeliveryDate() {
		return TargetDeliveryDate;
	}
	public void setTargetDeliveryDate(LocalDate TargetDeliveryDate) {
		this.TargetDeliveryDate = TargetDeliveryDate;
	}

}
```

Readwriteexcel.java

```
package com.example.properties;
import java.io.FileInputStream;

import java.io.FileOutputStream;
import java.io.IOException;
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 com.example.uploadingfiles.storage.StorageException;

import java.util.Comparator;

public class readwriteexcel {

	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

              int row_index = 1;
              /* ------------------------------------------
               * PURPOSE: collects data from each of the cells
               * ------------------------------------------
               */
              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 P1UOrder = (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();
                  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();
                }
                Cell RefGIDateCell = row.getCell(15);
                LocalDate RefGIDate = null;
                if (RefGIDateCell.getCellType() == CellType.STRING) {
                    String RefGIDateStr =
RefGIDateCell.getStringCellValue();
                    DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM/dd/yy");
                    RefGIDate = LocalDate.parse(RefGIDateStr, formatter);
                } else if (RefGIDateCell.getCellType() ==
CellType.NUMERIC) {
                    RefGIDate =
RefGIDateCell.getLocalDateTimeCellValue().toLocalDate();
                }
                Cell ArrivalDateCell = row.getCell(16);
                LocalDate ArrivalDate = null;
                if (ArrivalDateCell.getCellType() == CellType.STRING) {
                    String ArrivalDateStr =
ArrivalDateCell.getStringCellValue();
                    DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM/dd/yy");
                    ArrivalDate = LocalDate.parse(ArrivalDateStr,
formatter);
                } else if (ArrivalDateCell.getCellType() ==
CellType.NUMERIC) {
                    ArrivalDate =
ArrivalDateCell.getLocalDateTimeCellValue().toLocalDate();
                }
                Cell ETADateCell = row.getCell(17);
                LocalDate ETADate = null;
                if (ETADateCell.getCellType() == CellType.STRING) {
                    String ETADateStr = ETADateCell.getStringCellValue();
                    DateTimeFormatter formatter =
```

```
DateTimeFormatter.ofPattern("MM/dd/yy");
                    ETADate = LocalDate.parse(ETADateStr, formatter);
                } else if (ETADateCell.getCellType() == CellType.NUMERIC)
{
                    ETADate =
ETADateCell.getLocalDateTimeCellValue().toLocalDate();
                }
                Cell RDDDateCell = row.getCell(18);
                LocalDate RDDDate = null;
                if (RDDDateCell.getCellType() == CellType.STRING) {
                    String RDDDateStr = RDDDateCell.getStringCellValue();
                    DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM/dd/yy");
                    RDDDate = LocalDate.parse(RDDDateStr, formatter);
                } else if (RDDDateCell.getCellType() == CellType.NUMERIC)
{
                    RDDDate =
RDDDateCell.getLocalDateTimeCellValue().toLocalDate();
                }
                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));

                // 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::getDaysDifferenc
e));
            // 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) { // 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();
            }
        }

        return products;
    }

    public void updateExcelSheet(List<Product> products, String filePath)
{
                DecimalFormat decimalFormat = new DecimalFormat("0.00");
              try (FileInputStream fis = new FileInputStream(filePath);
                   Workbook wb = WorkbookFactory.create(fis)) {
```

```
                    Sheet sheet = wb.getSheet("Sheet1");
                    Drawing<?> drawing = sheet.createDrawingPatriarch();
                CreationHelper factory = wb.getCreationHelper();

                // Compare the days difference and sort products according
to that
                Collections.sort(products, Comparator

.comparing(Product::getDaysDifference).thenComparing(product ->
product.getHandOffDate()));

                    /* ---------------------------------------
                     * 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(DateTimeFor
matter.ofPattern("MM/dd/yy")));

row.createCell(15).setCellValue(product.getRefGIDate().format(DateTimeForma
tter.ofPattern("MM/dd/yy")));
                        if (product.getArrivalDate()!=null) {

row.createCell(16).setCellValue(product.getArrivalDate().format(DateTimeFor
matter.ofPattern("MM/dd/yy")));
                        }
                        if (product.getETADate()!=null) {

row.createCell(17).setCellValue(product.getETADate().format(DateTimeFormatt
er.ofPattern("MM/dd/yy")));
                        }

row.createCell(18).setCellValue(product.getRDD().format(DateTimeFormatter.o
fPattern("MM/dd/yy")));

row.createCell(19).setCellValue(product.getTargetDeliveryDate().format(Date
TimeFormatter.ofPattern("MM/dd/yy")));

                        /*
-----------------------------------------------------
                        / 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);
```

```
                        }

                        rowIndex++;
                    }

                    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();
                }
      }
}
```

FileUploadController

```
package com.example.uploadingfiles;

import java.io.IOException;
import java.nio.file.Path;
import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
```

```
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.multipart.MultipartFile;
import
org.springframework.web.servlet.mvc.method.annotation.MvcUriComponentsBuild
er;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.properties.Product;
import com.example.properties.readwriteexcel;
import com.example.uploadingfiles.storage.StorageException;
import com.example.uploadingfiles.storage.StorageFileNotFoundException;
import com.example.uploadingfiles.storage.StorageService;

@Controller
public class FileUploadController {

      private final StorageService storageService;

      @Autowired
      public FileUploadController(StorageService storageService) {
            this.storageService = storageService;
      }

      @GetMapping("/")
      public String listUploadedFiles(Model model) throws IOException {

            model.addAttribute("files", storageService.loadAll().map(
                        path ->
MvcUriComponentsBuilder.fromMethodName(FileUploadController.class,
                                    "serveFile",
path.getFileName().toString()).build().toUri().toString())
                        .collect(Collectors.toList()));

            return "uploadForm";
      }

      @GetMapping("/files/{filename:.+}")
```

```
	@ResponseBody
	public ResponseEntity<Resource> serveFile(@PathVariable String
filename) {

		Resource file = storageService.loadAsResource(filename);
		return
ResponseEntity.ok().header(HttpHeaders.CONTENT_DISPOSITION,
				"attachment; filename=\"" + file.getFilename() +
"\"").body(file);
	}

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

		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:/";
		}
		/* ------------------------------------------------
		 * PURPOSE: to deal with errors
		 * ------------------------------------------------
		 */
		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 (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
            }
      }
      @ExceptionHandler(StorageFileNotFoundException.class) // exception
handler for file not found
      public ResponseEntity<?>
handleStorageFileNotFound(StorageFileNotFoundException exc) {
            return ResponseEntity.notFound().build();
      }

}
```

UploadingFilesApplication

```
package com.example.uploadingfiles;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;

import com.example.uploadingfiles.storage.StorageProperties;
import com.example.uploadingfiles.storage.StorageService;

@SpringBootApplication
@EnableConfigurationProperties(StorageProperties.class)
public class UploadingFilesApplication {
```

```
	public static void main(String[] args) {
		SpringApplication.run(UploadingFilesApplication.class, args);
	}

	@Bean
	CommandLineRunner init(StorageService storageService) {
		return (args) -> {
			storageService.deleteAll();
			storageService.init();
		};
	}
}
```

FileSystemStorageService

```
package com.example.uploadingfiles.storage;

import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.stream.Stream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import org.springframework.stereotype.Service;
import org.springframework.util.FileSystemUtils;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

@Service
public class FileSystemStorageService implements StorageService {

	private final Path rootLocation; // gets location of file
```

```
	@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);
		}
	}

	@Override
```

```
	public Stream<Path> loadAll() {
		try {
			return Files.walk(this.rootLocation, 1)
				.filter(path -> !path.equals(this.rootLocation))
				.map(this.rootLocation::relativize);
		}
		catch (IOException e) {
			throw new StorageException("Failed to read stored files",
e);
		}

	}

	@Override
	public Path load(String filename) {
		return rootLocation.resolve(filename);
	}

	@Override
	public Resource loadAsResource(String filename) {
		try {
			Path file = load(filename);
			Resource resource = new UrlResource(file.toUri());
			if (resource.exists() || resource.isReadable()) {
				return resource;
			}
			else {
				throw new StorageFileNotFoundException(
						"Could not read file: " + filename);

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

	@Override
	public void deleteAll() {
		FileSystemUtils.deleteRecursively(rootLocation.toFile());
	}
```

```
	@Override
	public void init() {
		try {
			Files.createDirectories(rootLocation);
		}
		catch (IOException e) {
			throw new StorageException("Could not initialize
storage", e);
		}
	}
}
```

StorageException

```
package com.example.uploadingfiles.storage;
public class StorageException extends RuntimeException {
	public StorageException(String message) {
		super(message); // error message
	}
	public StorageException(String message, Throwable cause) {
		super(message, cause); // error message and cause
	}
}
```

StorageFileNotFoundException

```
package com.example.uploadingfiles.storage;
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
	}
}
```

StorageProperties

```
package com.example.uploadingfiles.storage;
import org.springframework.boot.context.properties.ConfigurationProperties;
```

```java
@ConfigurationProperties("storage")
public class StorageProperties {
	/*
	 * Folder location for storing files
	 */
	private String location = "upload-dir";
	public String getLocation() {
		return location;
	}
	public void setLocation(String location) {
		this.location = location;
	}
}
```

StorageService

```java
package com.example.uploadingfiles.storage;

import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

import java.nio.file.Path;
import java.util.stream.Stream;

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();

}
```