Case Study:
Simple Product Management

Requirement:

- To write a Java project that
  - reads in instructions in a predefined format from a text file, and
  - writes records to an output file in an organized format.

- The program is executed as:

  java CaseStudy.SPM inputFile outputFile

  - ***inputFile:*** the name of the input file that contains instructions for the management of a company inventory
  - ***outputFile:*** the name of the output file that contains the formatted inventory records

Instruction Format:

- Instructions include ADD, REMOVE products to/from inventory.
- The instructions are stored in ***inputFile*** in the following format:
  - ADD name use-by quantity price
    add product to the inventory.
    Example: ADD bread 09-09-2015 4 $1.01
  - REMOVE name
    remove all products with specified name
    Example: REMOVE milk

Simple sample input and output files

ADD honey 09-09-2015 4 $1.01

ADD bread 07-9-2015 2 $3.5

ADD pizza 15-09-2015 6 $5

ADD carrot 05-09-2015 10 $1
REMOVE carrot

ADD 20-09-2015 4 $3
ADD milk ab-09-2015 4 $3
ADD milk 20-09-2015 4.5 $3
ADD milk 20-09-2015 4 $-3

name	honey
useby	09-09-2015
quantity	4
price	$1.01

name	bread
useby	07-09-2015
quantity	2
price	$3.50

name	pizza
useby	15-09-2015
quantity	6
price	$5.00

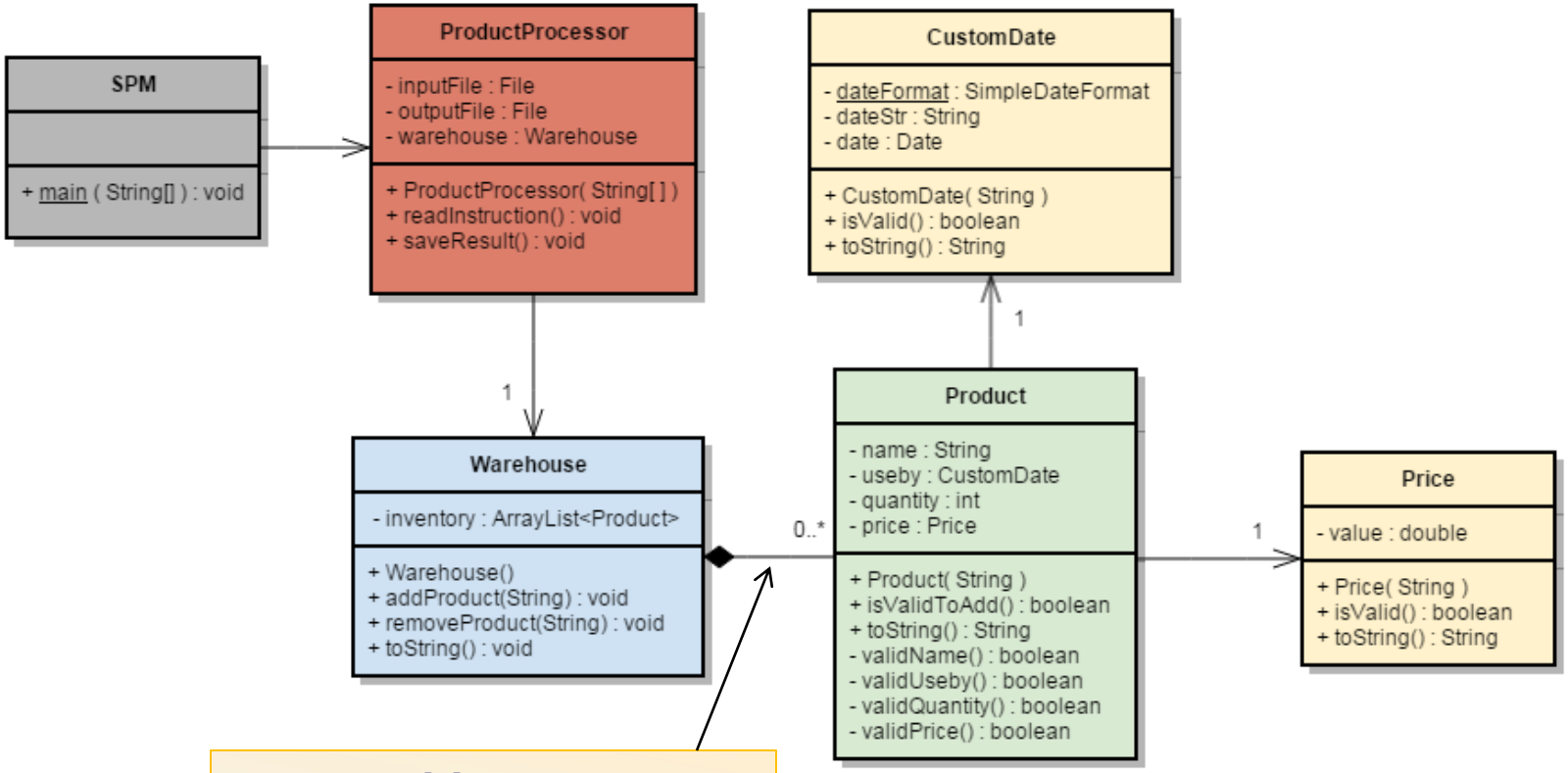
name	carrot
useby	05-09-2015
quantity	10
price	$1.00

invalid instructions

Functionalities:

- reading and writing data from/to file
- parse the instruction
- keep information of each product
- able to manage the production collection
  - add new product to collection
  - delete existing product from collection
- can retrieve and output the collection
- validate the correctness of input data
  - name
  - date
  - quantity
  - price
  - instruction format

Design Diagram



Composition represents "has a" relationship that is a "**part-whole**" or "**part-of**" relationship

A Product Class

```java
package CaseStudy;

public class Product {
	// instance fields
	private String name;
	private CustomDate useby;
	private int quantity;
	private Price price;

	// constructors
	public Product(){
		name = null;
		useby = null;
		quantity = -1;
		price = null;
	}
	public Product(String s) {
		String[] temp = s.trim().split("\\s");
		if (temp.length != 4) {
			name = null;
			useby = null;
			quantity = -1;
			price = null;
		} else {
			name = temp[0];
			useby = new CustomDate(temp[1]);
			try {
				quantity = Integer.parseInt(temp[2]);
			} catch (Exception e) {
				quantity = -1;
			}
			price = new Price(temp[3]);
		}
	}

	// instance methods
	// validations
	public boolean isValidToAdd() {
		return validName() && validUseby() && validQuantity() &&
		validPrice();
	}

	private boolean validName() {
		if (name != null && name.matches("[a-zA-Z]+")) {
			return true;
		} else {
			return false;
		}
	}

	private boolean validUseby() {
		return useby.isValid();
	}

	private boolean validQuantity() {
		return quantity >= 0;
	}

	private boolean validPrice() {
		return price.isValid();
	}

	public String toString() {
		return String.format("name: %s\nuseby: %s\nquantity:
		%d\nprice: %s ", name,useby,quantity,price);
	}

	// getters and setters
		…
}
```

A Product Class

```java
package CaseStudy;

public class Product {
```

"honey" "09-09-2015" "4" "$1.01"

```java
    // instance fields
    private String name;
    private CustomDate useby;
    private int quantity;
    private Price price;

    // constructor
    public Product(){
        name = null;
        useby = null;
        quantity = -1;
        price = null;
    }
```

" honey 09-09-2015 4 $1.01 "

```java
    // constructor
    public Product(String s) {
        String[] temp =
        s.trim().split("\\s");
        if (temp.length != 4) {
            name = null;
            useby = null;
            quantity = -1;
            price = null;
        } else {
            name = temp[0];
            useby = new CustomDate(temp[1]);
            try {
                quantity =
                Integer.parseInt(temp[2]);
            } catch (Exception e) {
                quantity = -1;
            }
            price = new Price(temp[3]);
        }
    }
    …
```

A Product Class

```java
package CaseStudy;

public class Product {
    // instance fields
    private String name;
    private CustomDate useby;
    private int quantity;
    private Price price;
    public Product(String s) {
        String[] temp =
        s.trim().split("\\s");
        if (temp.length != 4) {
            name = null;
            useby = null;
            quantity = -1;
            price = null;
        } else {
            name = temp[0];
            useby = new
            CustomDate(temp[1]);
            try {
                quantity =
                Integer.parseInt(temp[2]);
            } catch (Exception e) {
                quantity = -1;
            }
            price = new Price(temp[3]);
        }
    }

    public boolean isValidToAdd() {
        return validName() && validUseby() &&
        validQuantity() && validPrice();
    }

    private boolean validName() {
        if (name != null &&
        name.matches("[a-zA-Z]+")) {
            return true;
        } else {
            return false;
        }
    }
```

Regular expression

"honey"
"milk2"

A Product Class

```
    private boolean validUseby() {
        return useby.isValid();
    }

    private boolean validQuantity() {
        return quantity >= 0;
    }

    private boolean validPrice() {
        return price.isValid();
    }

    public String toString() {
        return String.format("name: %s\nuseby: %s\nquantity: %d\nprice: %s ",
        name,useby,quantity,price);
    }

    // getters and setters
        …

}
```

name	honey
useby	09-09-2015
quantity	4
price	$1.01

A Price Class

```java
package CaseStudy;

public class Price {
    private double value;

    public Price(String p) {
        p = p.trim();
        if (!p.startsWith("$")) {
            value = -1;
        }
        else {
            try {
                value =
                Double.parseDouble(p.trim()
                .substring(1));
            } catch (Exception e) {
                value = -1;
            }
        }
    }
```

"$10"

Tests if this string starts with the specified prefix.

```java
    public boolean isValid() {
        return value>=0;
    }

    public String toString() {
        if (isValid()) {
            return String.format("$%.2f",
            value);
        } else {
            return "invalid price";
        }
    }

}
```

"$10.00"

"44"

"$32.ee"

A CustomDate Class

```
package CaseStudy;
public class CustomDate {
    private static SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
    private String dateStr;
    private Date date;

    public CustomDate(String d){
        dateStr = d;
        String[] temp;
        if (dateStr.matches("\\d+\\D\\d+\\D\\d+")) {
            temp = dateStr.split("\\D");
            if (temp.length == 3) {
                for (int i = 0; i < 2; ++i) {
                    if (temp[i].length() < 2)
                        temp[i] = "0" + temp[i];
                }
                dateStr = temp[0] + "-" + temp[1] + "-" + temp[2];
            }
        }
        try {
            date = dateFormat.parse(dateStr);
        } catch (ParseException e) {
            date = null;
        }
    }
```

reformat the date string

parse the formatted string

A CustomDate Class

```
package CaseStudy;
public class CustomDate {
    private static SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
    private String dateStr;
    private Date date;

    public CustomDate(String d){
        dateStr = d;
        String[] temp;
        if (dateStr.matches("\\d+\\D\\d+\\D\\d+")) {
            temp = dateStr.split("\\D");
            if (temp.length == 3) {
                for (int i = 0; i < 2; ++i) {
                    if (temp[i].length() < 2)
                        temp[i] = "0" + temp[i];
                }
                dateStr = temp[0] + "-" + temp[1] + "-" + temp[2];
            }
        }
        try {
            date = dateFormat.parse(dateStr);
        } catch (ParseException e) {
            date = null;
        }
    }
```

"23-9-2015"

"23" "9" "2015"

"23/09/2015"

"1-10-2015"

"Abcd2015"

"31-2-2015"

A CustomDate Class

```
    public boolean isValid() {
        if(date!=null){
            // validate the date
            return true;
        }
        else{
            return false;
        }
    }

    public String toString() {
        return dateFormat.format(date);
    }

    public Date getDate() {
        return date;
    }
}
```

31-2-2015

"23-09-2015"

A Warehouse Class

```
package CaseStudy;
public class Warehouse {
    private ArrayList<Product> inventory;

    public Warehouse() {
        inventory = new ArrayList<Product>();
    }

    public void addProduct(String s) {
        Product p = new Product(s);
        if (p.isValidToAdd()) {
            inventory.add(p);
        }
    }

    public void removeProduct(String name) {
        name = name.trim();
        int i = 0;
        while (i < inventory.size()) {
            if (inventory.get(i).getName().equalsIgnoreCase(name)) {
                inventory.remove(i);
            } else {
                ++i;
            }
        }
    }
```

A Warehouse Class

```
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for(Product p : inventory){
            sb.append(p.toString());
            sb.append("\n\n");
        }
        return sb.toString();
    }

    public ArrayList<Product> getInventory() {
        return inventory;
    }

    public void setInventory(ArrayList<Product> inventory) {
        this.inventory = inventory;
    }

}
```

name	honey
useby	09-09-2015
quantity	4
price	$1.01
name	bread
useby	07-09-2015
quantity	2
price	$3.50
…	

A ProductProcessor Class

```
package CaseStudy;
public class ProductProcessor {
    private File inputFile;
    private File outputFile;
    private Warehouse warehouse;

    public ProductProcessor(String[] s) {
        inputFile = new File(s[0]);
        outputFile = new File(s[1]);
        warehouse = new Warehouse();
    }


    public void saveResult() {
        try {
            PrintWriter out = new PrintWriter(new FileOutputStream(outputFile));
            out.println(warehouse.toString());
            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

A ProductProcessor Class

```
    public void readInstruction() {
        try {
            Scanner scan = new Scanner(inputFile);
            while (scan.hasNextLine()) {
                String instruction = scan.nextLine();
                Scanner sc = new Scanner(instruction);
                String keyword, param;
                if (sc.hasNext()) {
                    keyword = sc.next();
                    if (sc.hasNextLine()) {
                        param = sc.nextLine();
                        if (keyword.equalsIgnoreCase("add")) {
                            warehouse.addProduct(param);
                        } else if (keyword.equalsIgnoreCase("remove")) {
                            warehouse.removeProduct(param);
                        }
                    }else{
                        continue;
                    }
                    sc.close();
                }else{
                    continue;
                }
            }
            scan.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

The readInstruction method

The method signature and try - catch

```
public void readInstruction() {
//read in the file, parse the instructions in the file, and put the records into the list

    try {

    } catch (Exception e) {
        System.out.println("Error:- " + e.getMessage());
    }
}
```

A ProductProcessor Class

```
try {
    Scanner scan = new Scanner(inputFile);
    while (scan.hasNextLine()) {
        String instruction = scan.nextLine();
        Scanner sc = new Scanner(instruction);
        String keyword, param;

        // parse each instruction and process it

    }
    scan.close();
}
```

try block

A ProductProcessor Class

```
while (scan.hasNextLine()) {
    String instruction = scan.nextLine();
    Scanner sc = new Scanner(instruction);
    String keyword, param;
    if (sc.hasNext()) {
        keyword = sc.next();
        if (sc.hasNextLine()) {
            param = sc.nextLine();
            if (keyword.equalsIgnoreCase("add")) {
                warehouse.addProduct(param);
            } else if (keyword.equalsIgnoreCase("remove")) {
                warehouse.removeProduct(param);
            }
        }else{
            continue;
        }
        sc.close();
    }else{
        continue;
    }
}
```

The major loop

Read through the line

Parse the single line insturction into a pair of keyword and parameters

A ProductProcessor Class

```
while (scan.hasNextLine()) {
    String instruction = scan.nextLine();
    Scanner sc = new Scanner(instruction);
    String keyword, param;
    if (sc.hasNext()) {
        keyword = sc.next();
        if (sc.hasNextLine()) {
            param = sc.nextLine();
            if (keyword.equalsIgnoreCase("add")) {
                warehouse.addProduct(param);
            } else if (keyword.equalsIgnoreCase("remove")) {
                warehouse.removeProduct(param);
            }
        }else{
            continue;
        }
        sc.close();
    }else{
        continue;
    }
}
```

The major loop

Check the keyword and match it with different operations to be performed

A ProductProcessor Class

```
if (sc.hasNextLine()) {
    param = sc.nextLine();
    if (keyword.equalsIgnoreCase("add")) {
        warehouse.addProduct(param);
    } else if (keyword.equalsIgnoreCase("remove")) {
        warehouse.removeProduct(param);
    }
}
```

Case 1: ADD instruction

```
public class Warehouse {
        …
    public void addProduct(String s) {
        Product p = new Product(s);
        if (p.isValidToAdd()) {
            inventory.add(p);
        }
    }
        …
}
```

A ProductProcessor Class

```
if (sc.hasNextLine()) {
    param = sc.nextLine();
    if (keyword.equalsIgnoreCase("add")) {
        warehouse.addProduct(param);
    } else if (keyword.equalsIgnoreCase("remove")) {
        warehouse.removeProduct(param);
    }
}
```

Case 2: REMOVE instruction

```
public class Warehouse {
            …
    public void removeProduct(String name) {
        name = name.trim();
        int i = 0;
        while (i < inventory.size()) {
            if (inventory.get(i).getName().equalsIgnoreCase(name)) {
                inventory.remove(i);
            } else {
                ++i;
            }
        }
    }
            …
}
```

A ProductProcessor Class

The major loop

```
while (scan.hasNextLine()) {
    String instruction = scan.nextLine();
    Scanner sc = new Scanner(instruction);
    String keyword, param;
    if (sc.hasNext()) {
        keyword = sc.next();
        if (sc.hasNextLine()) {
            param = sc.nextLine();
            if (keyword.equalsIgnoreCase("add")) {
                warehouse.addProduct(param);
            } else if (keyword.equalsIgnoreCase("remove")) {
                warehouse.removeProduct(param);
            }
        }else{
            continue;
        }
        sc.close();
    }else{
        continue;
    }
}
```

No parameters

No keyword (empty line)

main() method

```
public static void main(String[] args) {
    if(args.length == 2){
      ProductProcessor pp = new ProductProcessor(args);
      pp.readInstruction();
      pp.saveResult();
    }
}
```

```
public class Warehouse {
    public ProductProcessor(String[] s) {
        …
    }
    public void readInstruction() {
        …
    }
    public void saveResult() {
        …
    }
}
```