ESOF 422 – HW4

Secure by Design

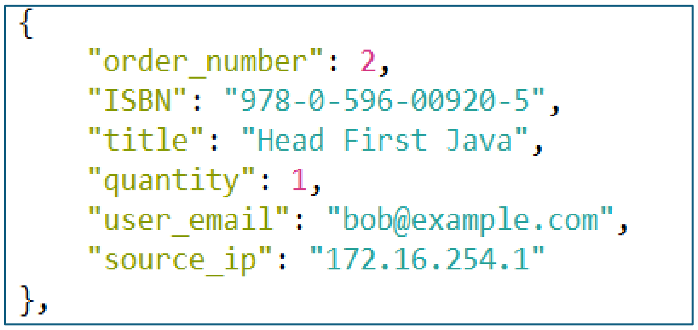
Due Friday April 4th @11:59 PM

The purpose of this assignment is to give you experience with the following:

- Secure by Design
- Principle of Failing Fast
- Input Validation and handling untrusted user input
- Domain Primitives

Background

This homework will involve only some Java programming on your end. You will download some starting code that reads in data from a large JSON file of customer orders for some online bookstore.



```
{
    "order_number": 2,
    "ISBN": "978-0-596-00920-5",
    "title": "Head First Java",
    "quantity": 1,
    "user_email": "bob@example.com",
    "source_ip": "172.16.254.1"
},
```

Order Number: Unique Identifier for each order used by the bookstore.

ISBN: ISBN 13 for the book being order, which is a unique identifier for the book.

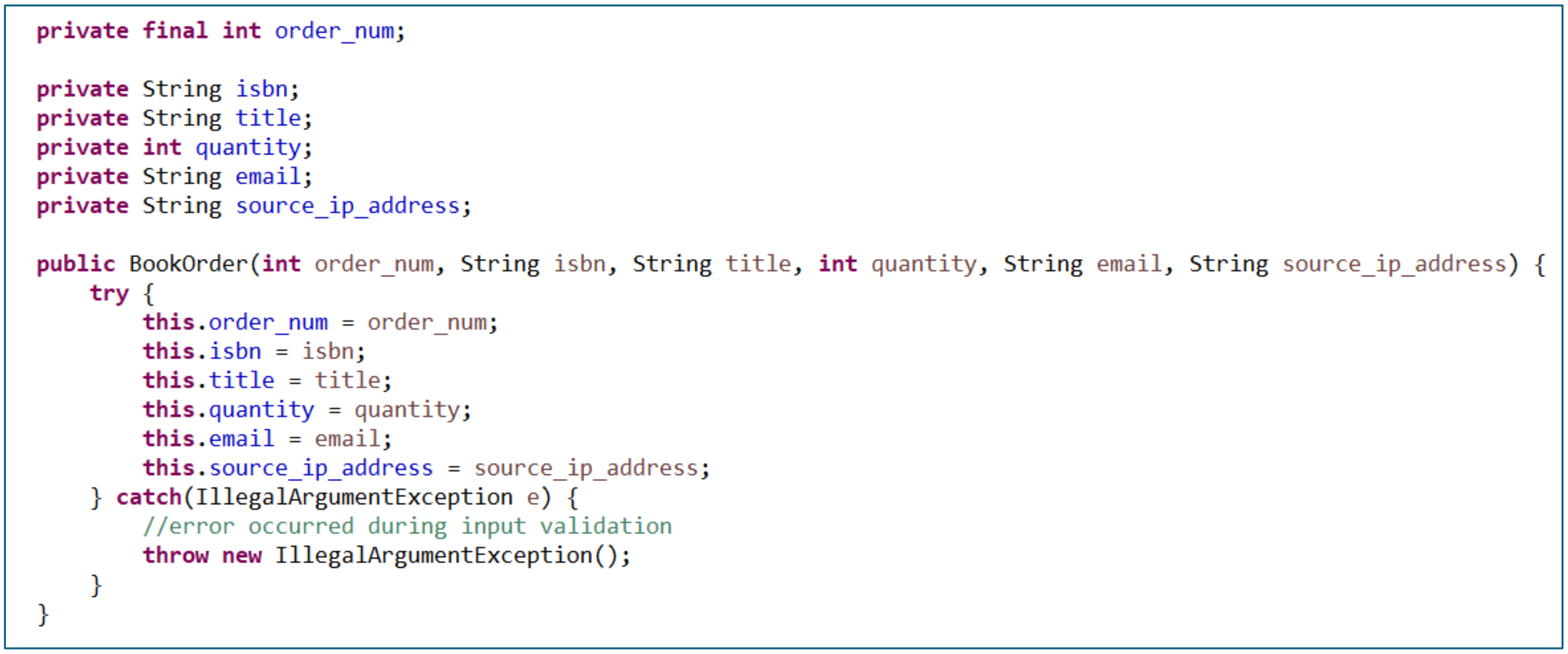
Title: Title of the book.

Quantity: Amount of that book being ordered.
User Email: Email account of who placed the order.
Source IP: The IP address the book order came from.

The starting code reads in data from the JSON, makes a **`BookOrder`** object for each order, and then adds the object to a **`List<BookOrder>`**. This list is then processed by billing and shipping, and customers will get their order.

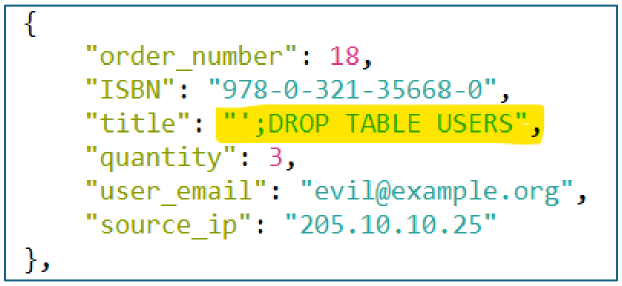
Issues with the Code

The screenshot below shows the current implementation of the **`BookOrder`** class. **`ISBN`**, **`title`**, **`email`** and **`source_ip_address`** are all represented by a basic Java `String` data type. Order Number and **`quantity`** are represented by an `int` data type.

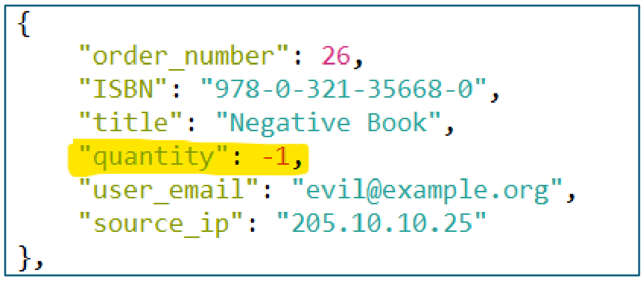


```java
private final int order_num;

private String isbn;
private String title;
private int quantity;
private String email;
private String source_ip_address;

public BookOrder(int order_num, String isbn, String title, int quantity, String email, String source_ip_address) {
    try {
        this.order_num = order_num;
        this.isbn = isbn;
        this.title = title;
        this.quantity = quantity;
        this.email = email;
        this.source_ip_address = source_ip_address;
    } catch(IllegalArgumentException e) {
        //error occurred during input validation
        throw new IllegalArgumentException();
    }
}
```

One major issue is that there is input validation when reading form the JSON file, which could lead to some potentially malicious inputs such as:



```json
{
    "order_number": 18,
    "ISBN": "978-0-321-35668-0",
    "title": "';DROP TABLE USERS",
    "quantity": 3,
    "user_email": "evil@example.org",
    "source_ip": "205.10.10.25"
},
```



```json
{
    "order_number": 26,
    "ISBN": "978-0-321-35668-0",
    "title": "Negative Book",
    "quantity": -1,
    "user_email": "evil@example.org",
    "source_ip": "205.10.10.25"
},
```

Because many of the data is treated as a Java String, there are many possible inputs that could be provided that are invalid according to the domain.

Another issue is that the instance field are current **mutable**, which could potentially lead to integrity and availability issues.

You will need to refactor the code and implement domain primitives for **ISBN**, **Title**, **Quantity**, **Email**, and **SourceIPAddress** and ensure each input is valid upon creation of a **BookOrder** object.

Directions

Start by downloading the starting code which can be found on D2L or here:

https://www.cs.montana.edu/pearsall/classes/spring2025/422/homework/Esof422-HW4.zip

Or Github Repo URL: https://github.com/reesep/esof422-spring2025

You will extract the ZIP file, and import the workspace into your IDE. There are some depencies that are used in the starting code. In Eclipse, there is an "Open Project from File System" that will make sure all depencies are included. If you use another IDE, it will be up to you to figure out how to get the starting code working.

When you run the code initially, all 50 orders will be accepted. There are various "malicious" or "unusual" inputs scatttered in the JSON file, and your goal is to make sure those orders are rejected and not added to the List.

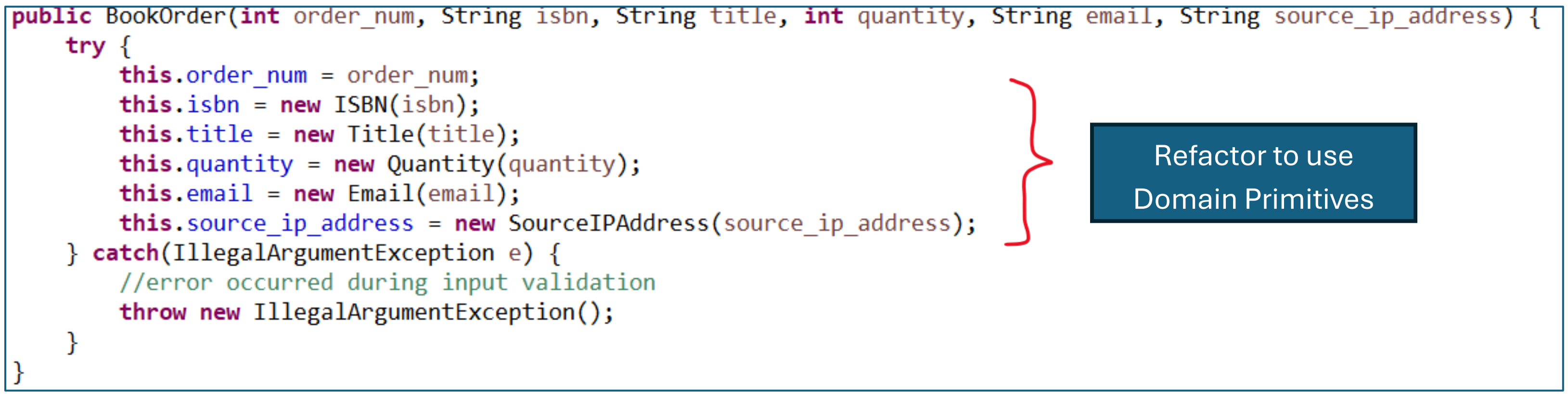
Part 1: Refactoring code to use Domain Primitives and Input Validation

You will need to define a **ISBN**, **Title**, **Quantity**, **Email**, and **SourceIPAddress** Java classes. In the constructor of those classes, you can call subroutines that will check if the input is valid or not. Any invalid inputs should `throw new IllegalArgumentException()` to make sure the error is caught by the `BookOrder` constructor. By adding input validation rules in the individual classes, it will ensure that `BookOrder` objects have valid data and nothing malicious is injected in the inputs.

You don't have to worry about order number. You can assume that value is selected internally, and a user cannot control the value of the order number.

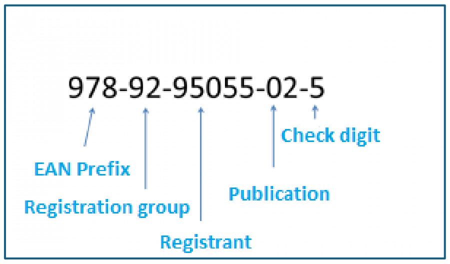
You will need to change the body of the constructor for `BookOrder`, and the instance fields. You should not modify anything else in the `BookOrder` class.

To help get you started, the new `BookOrder` constructor should look something like this:


```java
public BookOrder(int order_num, String isbn, String title, int quantity, String email, String source_ip_address) {
    try {
        this.order_num = order_num;
        this.isbn = new ISBN(isbn);
        this.title = new Title(title);
        this.quantity = new Quantity(quantity);
        this.email = new Email(email);
        this.source_ip_address = new SourceIPAddress(source_ip_address);
    } catch(IllegalArgumentException e) {
        //error occurred during input validation
        throw new IllegalArgumentException();
    }
}
```

Most of the code in the domain primitive classes will have code for input validation. You can call subroutines in the constructor (ex. **`isValid(value)`**, **`isBetween1and99(value)`**, etc). The rules for each input can be found on the next page.

ISBN-13 (String). An ISBN-13 is a 13 digit identifier for a book. There are five different portions of an ISBN-13 code (represented by a String in the code).



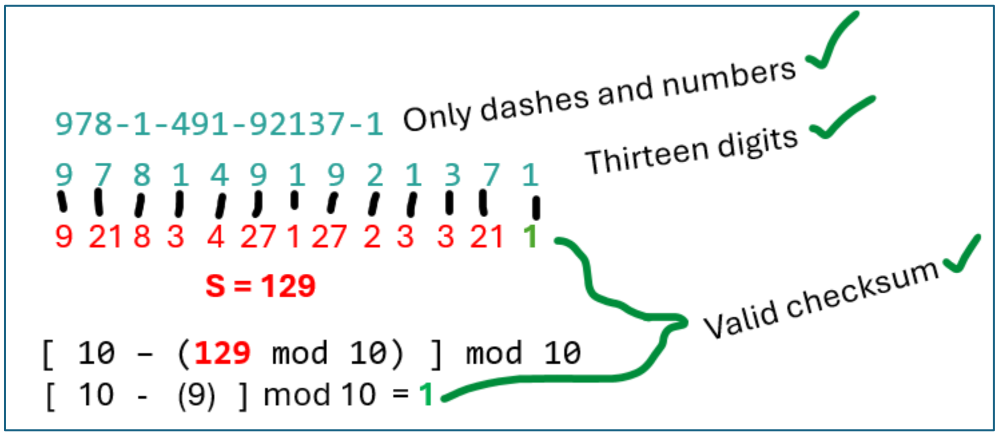
You don't need to know the specific of what each portion means, except for the last digit (the check digit or checksum digit). Here are the following rules for a valid ISBN 13 value:

1. The string must only have numerical digits or dashes.
2. When dashes are removed, there must be 13 digits.
3. The value has a valid checksum digit.

To determine if the if the checksum digit is valid, the following procedure is done.

- Multiply the odd digits by 1
- Multiply the even digits by 3
- Sum up the first 12 digits → **S**
- Compute the following calculation
  `[ 10 – (S mod 10) ] mod 10`
  The value of this calculation should match the last digit in the ISBN-13 value. If it is a different value, the input is invalid, and the order should be rejected.

  Here is a simple walkthrough of the checksum validation:



Book Title (String)

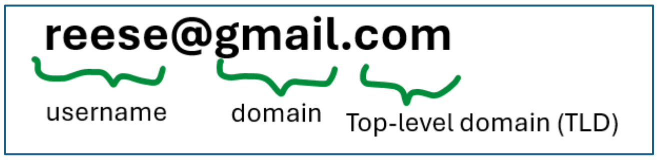
1. String only allowed to have alphanumeric characters (upper and lowercase), commas (**,**) and colons (**:**).
2. Title length must be between 1 and 100 characters (inclusive)

Quantity (int)

1. Must be a number between 1 and 99 (inclusive)

Email (String)

Emails are a tricky one to validate in the real world, so we will take a much simpler approach for this assignment. An email consists of three parts

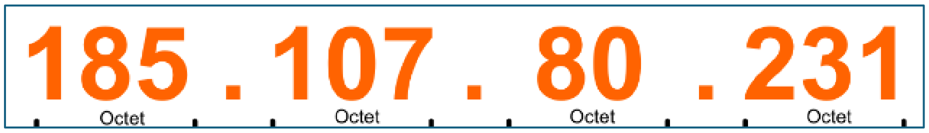


1. Only alphanumeric (upper and lowercase), dashes, periods, underscores are allowed in the username, domain, and TLD.
2. There must be an **@** symbol between the username and domain.
3. There must be a **.** between the domain and TLD.
4. The entire email must be between 1 and 320 characters.
5. There are many TLDs, but for the assignment, the following TLD are what is allowed"

.com .org .net .gov .edu .int .mil

IP Address (String)

IP addresses (IPv4) consists of four octets, separated by periods



1. Must contain four octets separated by .
2. Octets must only be numerical digits and must be between 0 and 255 (inclusive)
3. Cannot be a reserved IP address (should not begin with the following prefixes)
   - **127.X.X.X** are reserved for loopback addresses
   - **192.X.X.X** are reserved for private IP addresses
   - **10.X.X.X** are reserved for private IP addresses
   - **0.X.X.X** are reserved for self-communication
   - **255.255.255.255** is reserved for broadcast addresses

Part 2: Making primitives immutable

You should make sure that the six values are immutable to prevent unintentional data modification. This part is super simple—just use the Java **`final`** keyword in the **`BookOrder`** class.

Part 3: Overridng the toString() method in domain primitive classes

The **`toString()`** method in the **`BookOrder`** class should still work as intended. When you make the domain primitive classes, object memory locations will be printed out. You will need to override the **`toString()`** method in the domain primitive classes to make sure the data value is returned instead of the memory address.

Expected output.

When your program is successfully refactored and rejects invalid input from the JSON, the following output should appear when the demo class is run:

```
Order 1 accepted
Order 2 accepted
Order 3 accepted
Order 4 accepted
Order 5 accepted
Order 6 accepted
Order 7 accepted
Order 8 accepted
Order 9 accepted
Order 10 accepted
Order 11 denied...
Order 12 denied...
Order 13 denied...
Order 14 denied...
Order 15 accepted
Order 16 accepted
Order 17 denied...
Order 18 denied...
Order 19 denied...
Order 20 denied...
Order 21 accepted
Order 22 accepted
Order 23 accepted
Order 24 accepted
Order 25 accepted
Order 26 denied...
Order 27 denied...
Order 28 accepted
Order 29 denied...
Order 30 denied...
Order 31 denied...
Order 32 denied...
Order 33 denied...
Order 34 denied...
Order 35 denied...
Order 36 accepted
Order 37 accepted
Order 38 accepted
Order 39 accepted
Order 40 accepted
Order 41 denied...
Order 42 denied...
Order 43 denied...
Order 44 denied...
Order 45 denied...
Order 46 denied...
Order 47 denied...
Order 48 denied...
Order 49 denied...
Order 50 accepted
```

If you want a breakdown of why certain order numbers were rejected, an explanation can be found here:
https://www.cs.montana.edu/pearsall/classes/spring2025/422/homework/rejections.pdf

Submission Instructions

When you are ready to submit, you will zip you your *entire* workspace into a **.zip** file, and submit the **.zip** file the D2L/Brightspace dropbox. You do not need to submit a hard copy of anything. Late work is not accepted.

The TA will grade with same JSON file, but in theory your program should be able to withstand more malicious inputs being thrown at it in a similar JSON format.

Hints

Regular expressions (regex) will be your friend in this assignment, you are encouraged to research how regular expressions work. You are allowed to use AI tools to help you with small parts of the assignment (regular expressions), but they should not be used to write your entire solution.

Don’t try to solve everything in one go. Try to take the assignment step by step. Write some code, test it, write some more code, test, until you are finished. It will probably be helpful to comment out and uncomment as you are working through the assignment.

Grading Rubric (100 points total)

Requirement	Points
ISBN domain primitive is implemented correctly, and rejects the appropriate invalid inputs	20
Book Title domain primitive is implemented correctly, and rejects the appropriate invalid inputs	10
Quantity domain primitive is implemented correctly, and rejects the appropriate invalid inputs	10
Email domain primitive is implemented correctly, and rejects the appropriate invalid inputs	20
Source IP Address domain primitive is implemented correctly, and rejects the appropriate invalid inputs	20
Valid inputs are not rejected	10
Domain primitive fields are immutable in **`BookOrder`** class	5
`toString()` in **`BookOrder`** still works (no memory addresses)	5