# Java Fundamentals
# Chapter 19 Practice Notes

Virtual Pair Programmers
Let the course come to you

In this chapter we are going to create a class to model a customer borrowing a book from the library - we'll call this class "Loan". The class will have fields that store information about each book and customer, together with when the book is due to be returned (the loan expiry date). We will then create another class to store all the loans, called the LoansRegistry.

## Task Overview

Stage 1

Create a new class, called Loan, with the following fields:

id - a unique ID number for each loan

customer - which customer is borrowing the book

book - which book is being borrowed

startDate - the date on which the customer borrows the book.

dueDate - the date on which the customer is due to return the book

returnDate - the date on which the customer did return the book. This will be empty (null) for current loans - when the customer hasn't yet returned the book.

status - this will indicate whether the loan is a current loan (the customer has the book) or a historic loan (the customer returned the book). There should be only 2 valid values for the status - CURRENT or HISTORIC - this will be achieved by using an Enum type.

Note that all the fields except for returnDate will be set in the loan's constructor. Set the startDate to be today's date, and the dueDate to be 2 weeks after the startDate.

Stage 2

Create the constructor to set the initial value of each of the loan class fields, and each of the following methods:

toString - to allow us to print out the loan

equals - to allow us to compare 2 loan objects. Simply compare the IDs to see if they're the same

getCustomer, getBook and getDueDate get methods

endLoan - this method will be called to indicate that the customer has returned the book. In its implementation it should set the status to HISTORIC and the returnDate to today's date.

Stage 3

Create a new class, called LoanRegistry. This class will have 1 field - an array of type Loan, called registry.

Stage 4

Add a constructor to the LoanRegistry class to initialize the registry array, and the following methods:

addLoan - this should take a parameter variable of a loan, and add it to the array. We need some logic here to check that the loan isn't already being stored - if it is, then the method should throw an appropriate exception.

findLoan - this should take a parameter variable of a bookID, and return a variable of type loan, by searching through the array for an item with a matching ID. This search should only return loans with a status of CURRENT. If there are no matching loans found, this method should throw an appropriate exception.

isBookOnLoan - this should take a parameter variable of a book, and return a variable of type boolean, to indicate whether or not there is a loan in the array with a matching book and a status of CURRENT.

See page 2 for a step-by-step walk-through

Virtual Pair Programmers
Let the course come to you

# Java Fundamentals
# Chapter 19 Practice Notes

This page contains supplementary information to support you in the task. Try and work out what is needed first, and then use this page as a hint if you get stuck. Don't forget there is a full work through on the video, and sample completed code in the Practicals and Code folder.

## Task Detail

Step 1 - create the loan class

Create the new class in the models package. Call the class `Loan`. It should not be a runnable class.

Add the following fields to the class:

`id` - this should be of type `int`

`customer` - this should be of type `Customer`

`book` - this should be of type `Book`

`startDate` - this should be of type `Date` - you will need to import the Date class - ensure you pick java.util.Date.

`dueDate` - this should be of type `Date`

`returnDate` - this should be of type `Date`

All the fields should be defined as private. The general syntax is:

```
        private type name;
```

Step 2 - create the Status Enum

Create the new Enum in the utilities package. Call it `LoanStatus` and put in 2 values of `CURRENT` and `HISTORIC`.

If you need a reminder of the syntax of an enum type, review the GenderType Enum that we created for the Customer class.

Once you have created the Enum, you can add the field `Status` to the Loan class. You will need to import the LoanStatus Enum type into the Loan class.

Step 3 - add a constructor

Create the constructor to set each the following variables when the Loan class is initialized - id, customer, book,

Note that when the constructor is called, the calling method should pass in the id, customer and book. However all other fields should be set within the constructor's code using the following rules:

set startDate to today's date

set dueDate to 2 weeks following today's date

set status to CURRENT

Do not set the value of returnDate in the constructor.

Step 4- create the toString() and equals() methods

Create the toString() method to return the Loan's ID, the customer's name and the book title.

Create the equals() method to compare the Loan IDs, and return TRUE if the IDs match.

HINT: You can use Eclipse to generate the toString() and Equals methods, although you'll need to edit the default toString() method to return just the customer's name and the book title.

Step 5- add get methods

Create standard get methods to return the customer, book, dueDate and status.

TIP: On the video I'll show you how to use Eclipse to generate these for you.

Step 6- create the endLoan() method

Create the endLoan method - this should set the status to HISTORIC and the returnDate to today.

Be sure to watch the video!

If you manage to follow the steps and complete the tasks, well done! But it's worth watching the video to ensure your code is similar to mine... and there are some tips in there too!

Virtual Pair Programmers
Let the course come to you

## Step 7 - create the LoanRegistry class

Create the new class in the models package. Call the class `LoanRegistry`. It should not be a runnable class.

Add the following field to the class:

`registry` - this should be an array of type `Loan`

## Step 8 - add a constructor

Create the constructor to initialize the registry array within the LoanRegistry class. The size of the array should be set to 100.

## Step 9- create the custom exceptions

The addLoan() method will need to throw an exception if the loan being added already exists in the array, so create a new exception called `LoanAlreadyExists-Exception`.

The findLoan() method will need to throw an exception if the loan being searched for doesn't exist in the array, so create a new exception called `LoanNotFoundExcep-tion`.

## Step 10 - create the addLoan method

Create a new method called addLoan. This will not return anything, it will take a parameter variable of type Loan, and it might throw a LoanAlreadyExistsException.

Within the method, write some code that first checks all the existing loans in the array to see if there is one that is equal to the loan passed into the method - if there is one, throw the LoanAlreadyExistsException. To do this check, you'll need a for loop, and you'll need to call the loan's .equals() method.

If no matching loans are found, add this loan to the Array.

HINT: A local variable will be needed to store how many items are in the array, like we did with bookCatalog - call this `nextPosition`.

## Step 10 - create the findLoan method

Create a new method called findLoan. This will return a variable of type Loan, it will take a parameter variable of type int (the bookID we are looking for), and it might throw a LoanNotFoundException.

Within the method, write some code that checks all the existing loans in the array to see if there is one that has a book with an ID equal to the bookID passed into the method, and a status of CURRENT - if there is one we will return the loan. if none are found, throw the LoanNot-FoundException.

HINT: Use && to check two different conditions within a single if statement.

## Step 10 - create the isBookOnLoan  method

Create a new method called isBookOnLoan. This will return a variable of type boolean, it will take a parameter variable of type int (the bookID we are looking for), and it will not throw any exceptions.

Within the method, write some code that checks all the existing loans in the array to see if there is one that has a book with an ID equal to the bookID passed into the method, and a status of CURRENT - if there is one we will return the value TRUE. if none are found, return the value FALSE .

HINT: You can call the findLoan method  to determine if the loan exists, and use the result to determine what to return.

Be sure to watch the video!

At the end of the video, I write some code which tests the code we have written - be sure to write this test code in your project, so that you can validate the code you have written.