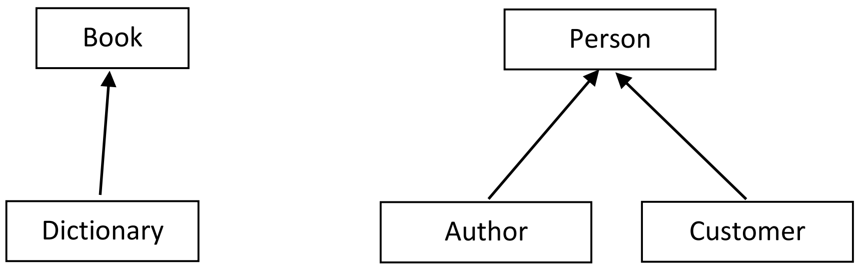
In this homework, you will implement a simple rental system for libraries with the following object oriented programming class hierarchy:



Please find the class details in below.

1. Implement a **Book** class with the following UML diagram.

Book	
-	id: int
-	title: String
-	author: Author
-	borrowed: boolean
+	Book(int id, String title, Author author)
+	Book(int id, String title)
+	isBorrowed():boolean
+	borrowed():boolean
+	returned():boolean
+	toString(): String
+	getter/setter methods for id, title, and author

- **Book** is the superclass of **Dictionary** class.
- **Book** class has several data fields, getter/setter and **toString** methods.
- Each book should have an **id**, a **title**, an **author** (Author class will be shown in below), and a **borrowed** attributes:
  - **id**: a unique identifier for the book
  - **title**: title of the book
  - **author**: a reference variable for an **Author** class.
  - **borrowed**: a boolean variable indicates that the book is available or not.
- Each book is created either with a unique **id**, a **title**, and an **author** arguments or with only a unique **id** and title arguments.

- The **isBorrowed** method should find out if the book is available.
- The **borrowed** method should borrow the book. If it has been already borrowed, it returns **false**, and vice versa.
- The **returned** method should return the book. If it has been already returned, it returns **false**, and vice versa.
- The **toString** method should return a string that represents the book.
    - Ex: `Book name is The Da Vinci Code, Author is Dan Brown.`
- There are setter/getter methods.

2. Implement a **Dictionary** class with the following UML diagram.

Dictionary	
-	definitions: int

+
+
+
+
	Dictionary(int id, String title, Author author, int definitions)
	Dictionary(int id, String title, int definitions)
	getter/setter methods for the **definitions** attribute
	toString(): String

- Each **Dictionary** can be created either with a unique **id**, a **title**, an **author**, and **definitions** arguments or only with a unique **id**, a **title**, and **definitions** arguments .
    - The **definitions** data field stores the number of words in the dictionary.
- You should invoke the superclass constructor(s) in **Dictionary** constructor(s).
- The getter and setter method for the data field **definitions.**
- The **toString()** method that returns a string description for the dictionary.
    - Ex: `Dictionary Name is Oxford Dictionary of English, definitions: 6000`

3. Implement a **Person** class with the following UML diagram.

Person	
- - -	name: String birthDate: String birthPlace: String
+ + + +	Person(String name, int birthDate , String birthPlace) Person(String name, int birthDate) Person(String name) getter/setter / toString methods

- **Person** is the superclass of **Author** and **Customer** classes.
- Each **Person** has a **name**, a **birthDate**, and a **birthPlace** data fields.
- Each person can be created with one of the three ways:
  - with a **name**, a **birthDate** and a **birthplace,**
    - Note that the **birthDate** is taken as a year (Ex:1980) in type of **int**, but you should store it as **String** in the class attribute.
  - with only a **name** and a **birthdate**,
  - with only a **name.**
- There are setter/getter methods for each data field.
- The **toString()** method describes the person.
  - Ex: `Name: Ayse Caliskan, Birth Date: 1995, Birth Place: Istanbul`

4. Implement an **`Author`** class with the following UML diagram.

Author	
-	publisher: String
+ +	Author(String name, String publisher, int birthDate) getter/setter and toString methods

- The **publisher** data field stores the company that publishes the author's books.
- You should invoke the superclass constructor in the **Author** constructor.
- There are setter/getter methods for the publisher.
- The **toString()** method describes the author.
  - Ex: `Author is Dan Brown`

5. Implement a **`Customer`** class with the following UML diagram.

Customer	
-	address: String
-	borrowedBook: Book
-	borrowABook: boolean
+	Customer(String name, String birthPlace, int birthDate, String address)
	Customer(String name, int birthDate, String address)
+	Customer(String name, String address)
+	Customer(String name, int birthDate)
+	getter/setter / toString methods

- Each **Customer** has an **address**, a **borrowedBook** that stores the book that is borrowed by the customer, and a **borrowABook** that specifies whether the customer borrows a book (default false).
- Each **Customer** can be created with one of the four constructors defined in the UML diagram above. You should invoke the superclass constructor(s) in these constructors.
- There are getter and setter methods for the data fields **address, borrowABook** and **borrowedBook**.
    - You should perform necessary settings when a customer borrows or returns a book.
- The **toString()** method returns a string description for the customer.
    - Ex: `Name: Ayse Caliskan, Birth Date: 1995, Birth Place: Istanbul`
      `Address: 18 Green Brier Blv.`

6. Implement a **`Library`** class with the following UML diagram.

Library	
-	address: String
-	books: ArrayList<Book>
-	customers: ArrayList<Customer>
+	Library(String address)
+	printOpeningHours():void
+	printAddress():void
+	addBook(Book book): boolean
+	addCustomer (Customer customer): void
+	borrowBook (String bookName, String personName): void
+	returnBook (String personName): void
+	printAvailableBooks (): void
+	getBooks():ArrayList< Book>
+	getCustomers():ArrayList< Customer>
+	getter/setter / toString methods

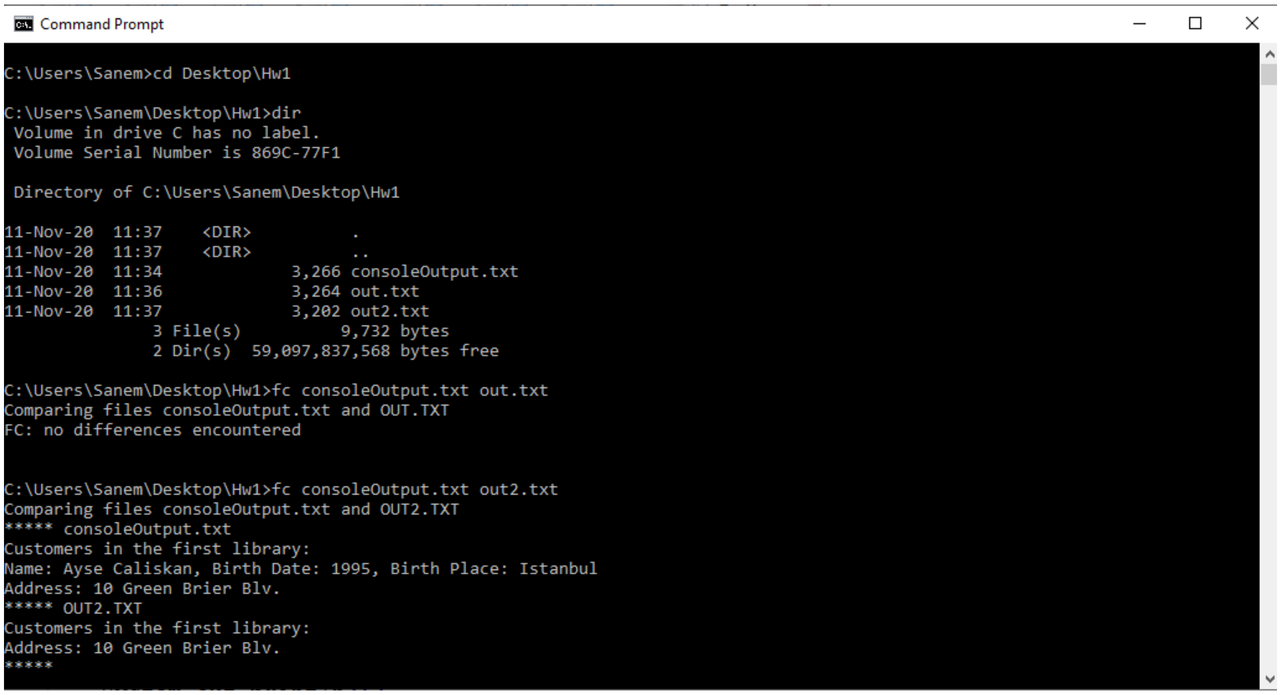
- **Library** class represents a library and manages a collection of books and customers.

- All libraries have the same hours: **9 AM** to **5 PM** daily.
- However, they have different **addresses**, **customers** (i.e. arrays of **Customer** objects) and **book** collections (i.e. arrays of **Book** objects).
- The types of **customers** and **books** should be **ArrayList**.
- The class must have the following methods:
    - **printOpeningHours**: It should print the opening hours of the library.
        - **Ex**: Libraries are open daily from 9 am to 5 pm.
    - **printAddress**: It should print the address of the library.
        - **Ex**: `221B Baker St.`
    - **addBook**: It should add the given **book** to the **books** list.
    - **addCustomer**: It should add the given **customer** to the **customers** list**.**
    - **borrowBook**: It should add the given **book** to the given **customer**.
        - Note that you should perform necessary checks whether the given **bookName** is in the library collection or the given **customerName** is in the customer list of the library. If not, print necessary warnings.
        - You should perform necessary checks whether the book is already borrowed by another custormer or the customer has already borrowed a book. Each **Book** can be given to a single **Customer** or each **Customer** can borrow a single **Book**.
        - The **bookName** and **customerName** are given to the method in Strings.

    - **returnBook**: It should return the book borrowed by the given customer. If there is no such a customer, please print an error message.
        - **Ex**: `Sorry, Ayse Caliskan is not a customer.`
    - **printAvailableBooks**: It should print all the available books in the library by traversing the **books** list.
        - **Ex**:
```
Book name is The Da Vinci Code, Author is Dan Brown
Book name is A Tale of Two Cities, Author is Charles Dickens
Dictionary Name is Oxford Dictionary of English, definitions: 6000
```
    - **getBooks**: It should return the list of all books of the library.
    - **getCustomers**: It should return the list of all customers of the library.

7. We will give you a test class for your program (Test.java in the attachment).
    - An example console output of the test program is given in “consoleOutput.txt” (You are not required to do File I/O, the output is given in a file due to long output size).
    - You should check that your console output must be the same as the consoleOutput.txt
        - Please copy-paste your console output to a text file (ex: out.txt)
        - Then, compare the content of your output file with the given consoleOutput.txt

- For Windows, you can compare the content of two files with the "**`fc`**" command. The usage is shown in the figure below:



Command Prompt

```
C:\Users\Sanem>cd Desktop\Hw1

C:\Users\Sanem\Desktop\Hw1>dir
 Volume in drive C has no label.
 Volume Serial Number is 869C-77F1

 Directory of C:\Users\Sanem\Desktop\Hw1

11-Nov-20  11:37    <DIR>          .
11-Nov-20  11:37    <DIR>          ..
11-Nov-20  11:34             3,266 consoleOutput.txt
11-Nov-20  11:36             3,264 out.txt
11-Nov-20  11:37             3,202 out2.txt
               3 File(s)          9,732 bytes
               2 Dir(s)  59,097,837,568 bytes free

C:\Users\Sanem\Desktop\Hw1>fc consoleOutput.txt out.txt
Comparing files consoleOutput.txt and OUT.TXT
FC: no differences encountered


C:\Users\Sanem\Desktop\Hw1>fc consoleOutput.txt out2.txt
Comparing files consoleOutput.txt and OUT2.TXT
***** consoleOutput.txt
Customers in the first library:
Name: Ayse Caliskan, Birth Date: 1995, Birth Place: Istanbul
Address: 10 Green Brier Blv.
***** OUT2.TXT
Customers in the first library:
Address: 10 Green Brier Blv.
*****
```

- In the figure above, outputConsole.txt and out.txt have the same content; on the other hand, outputConsole.txt and out2.txt have different contents.

- This is a simple scenario to test your class implementations. There might be other test cases, too. Therefore, please pay attention to use the same class, method and variable names in your implementations.
- You are allowed to increase the number of methods in the classes; however, you cannot decrease the number of them.