# Software Engineering

Books or notes are **not** allowed.
Write only on these sheets. **Concise** and **readable** answers please.

Surname, name, matricola _____
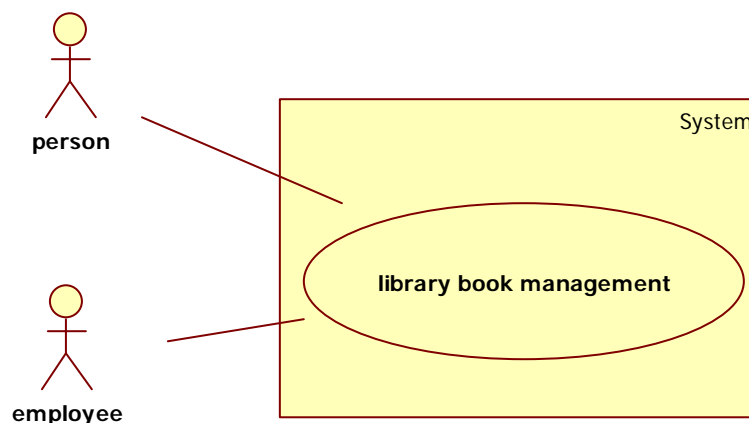
*Library book management*
A public library requires a web based book management system. First a person has to register in the system, and receives an ID and password. After registration a person can browse the catalogue of books, and see the availability of books. The library can have more than one copy of the same book. If a book is available the person can issue a request for it. The request lasts one day, if the person does not collect the book within one day, the request is dropped. If a book is not available, the person can ask to reserve it. When the book is returned, the first person who issued a reservation for that book is notified, and has one day to collect the book (otherwise the reservation is dropped and the next person is notified).
When a person comes to the library to collect a book, the rental is recorded by the personnel of the library. The person has to return the book within two weeks. After this period the person is notified every day, until he returns the book. At return an employee in the library records the return.
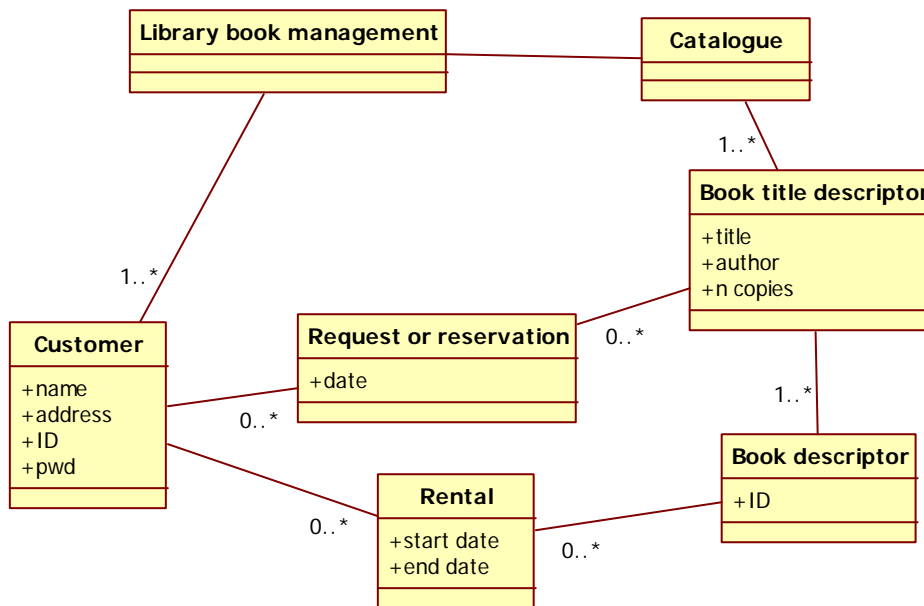Other functions of the system, available to library personnel only, are statistics on books (how many time a book is rented, average duration of a rental), and on customers (customers who do not return books on time, customers who do not comply to requests and reservations)

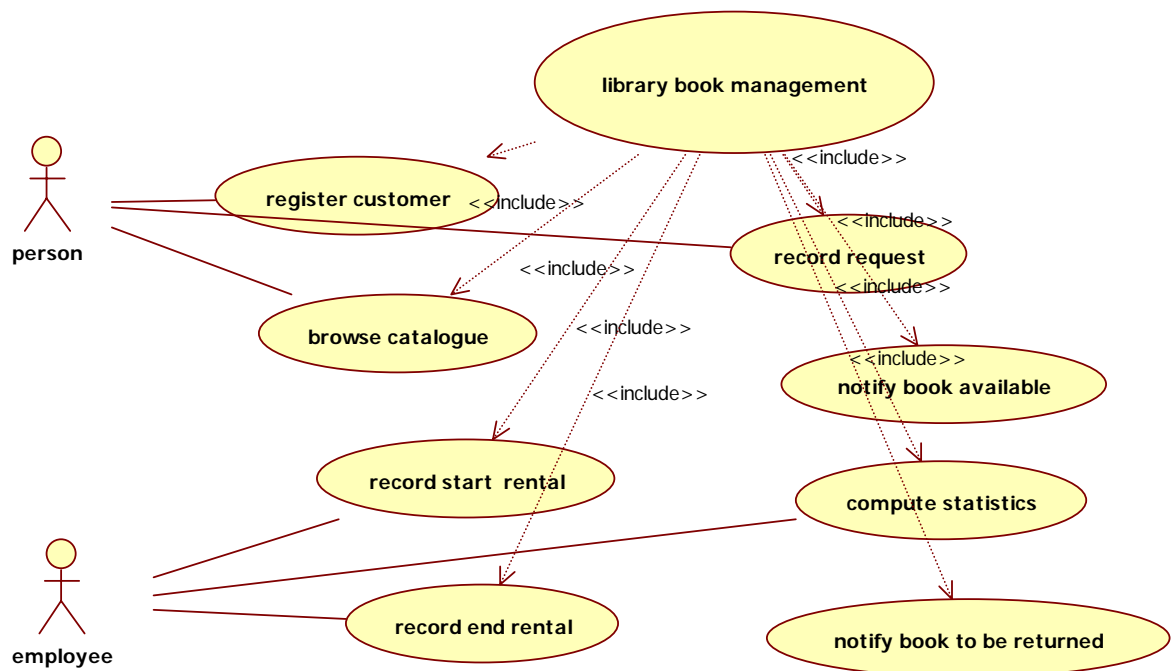1 (13 points) – a.  Define the context diagram (including relevant interfaces)

Interface: Internet link

Define the class diagram for the application

| Library book management | | Catalogue |
|---|---|---|

**Book title descriptor**

+title
+author
+n copies

1..*

**Customer**

+name
+address
+ID
+pwd

1..*

**Request or reservation**

+date

0..*

0..*

**Book descriptor**

+ID

1..*

**Rental**

+start date
+end date

0..*

0..*

Define the use case diagram

library book management

person

register customer  <<include>>

browse catalogue

<<include>>

<<include>>

<<include>>

record request

<<include>>
<<include>>

notify book available

<<include>>

record start rental

record end rental

employee

compute statistics

notify book to be returned

Define one scenario describing a book request operation
Precondition: user U is registered and has right to borrow book. Book B is available
Postcondition: book B is not available.

| Step | Description |
|------|-------------|
| 1 | User U logs in |
| 2 | User U browses catalogue |
| 3 | User U selects  book B |
| 4 | Request for book B is issued (book B not available anymore, and request to be dropped in 24 hrs) |


Define one scenario describing a book return operation
Precondition: book B is rented to user U
Postcondition: B is available. Rental of B to U is closed. Statistics on B and U are updated.

| Step | Description |
|------|-------------|
| 1 | User U is identified by clerk (function outside system) |
| 2 | U physically returns B to employee (function outside system) |
| 3 | Employee records end of rental B to U (B is available) |
| 4 | If book B has pending reservations, first user who reserved is notified |

**2a (4 points) -Define black box tests for the following function**

boolean authorizeRental(int nDroppedRequests, int nDroppedReservations);

the function defines if a rental (as in ex 1) can be authorized, with these rules
-if nDroppedRequests > 5, no authorize (returns false)
-if nDroppedReservations > 5, no authorize (returns false)
-if nDroppedRequests + nDroppedReservations > 7, no authorize (returns false)
-else authorize

| Criterion | Valid Class | Invalid Class | Boundary conditions | Tests |
|---|---|---|---|---|
| Sign of parameters | Positive | Negative | 0 | Valid<br>T0) aR( 1, 1) , true<br><br>Invalid<br>T1) aR ( -1, 0 ), false<br><br>Boundaries<br>T2) aR( 0, 0 ) , true<br>T3) aR( 1 ,0 ) , true<br>T1,  false |
| Range of parameters | 0 to 5 (boundaries included ) | <0 , >5 | 0, 5 | Valid<br>T4) aR( 3, 2)  , true<br><br>Invalid<br>T5) aR( -2,4) , false<br>T6) aR( 6, 4) , false<br><br>Boundaries<br>0<br>T1, false<br>T2, true<br>T3, true<br><br>5<br>T7) aR( 4, 2 ) , true<br>T8) aR (5,2 ) ,true<br>T6, false |
| Sum of parameters | 0 to 7 (boundaries included ) | <0, >7 | 0, 7 | Valid<br>T4, true<br><br>Invalid<br>T9) aR( 4,5) , false |

| | | | | Boundaries |
|---|---|---|---|---|
| | | | | 0 |
| | | | | T1, false |
| | | | | T2, true |
| | | | | T3, true |
| | | | | |
| | | | | 7 |
| | | | | T7, true |
| | | | | T8, true |
| | | | | T10) aR( 4,4 ) , false |
| Type of parameters | int | Other types | | Valid T1, true |
| | | | | |
| | | | | Invalid T11, aR ('c','d'), error |

Max_int and min_int are not checked because the range of the valid class is restricted to 0-5 .

**2b (4 points) -Define black box tests for the following function**

float convertFahrenheitToCelsius(float fahrenheit);

the function receives a temperature in Fahrenheit degrees, and converts it to Celsius (remember Celsius= (Fahrenheit – 32)/180*100 )

| Criterion | Valid Class | Invalid Class | Boundary conditions | Tests |
|---|---|---|---|---|
| Output range | fahrenheit <= a * <br><br> fahreneheit >= b * | fahrenheit <b fahrheneit > a | b , a | As usual test valid, invalid, boundaries ( boundary-1, boundary, boundary +1 ) |
| Type of input | float | Not float | | Test valid, invalid |

\* $a = maxFloat$  because

$$\frac{Fahrenheit - 32}{180} \cdot 100 \le maxFloat \qquad Fahreneit - 32 \le \frac{maxFloat \cdot 180}{100}$$

$$Fahrenheit \leq 1,80 \quad maxFloat + 32$$ that is greater than *maxFloat*

** $b = minFloat + 32$
for 2 reasons :

1) Numerator
$$Fahreneit - 32 \quad minFloat \quad Fahreneit \quad minFloat \quad 32$$

2) Expression

$$\frac{Fahreneit - 32}{180} \quad 100 \geq minFloat \quad Fahrenheit - 32 \geq \frac{minFloat \quad 180}{100}$$

$$Fahrenheit \geq 1,80 \quad minFloat + 32$$ that is less than *minFloat* (because *minFloat* is negative and it is multiplied by 1,80 ) , so the lower bound is $b = minFloat + 32$

**3 (6 points) – For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage.**
**Discuss the feasibility of path coverage.**
**For the test cases, write only the input value.**

**Report here the coverage obtained**
**--node coverage:**
**--edge coverage:**
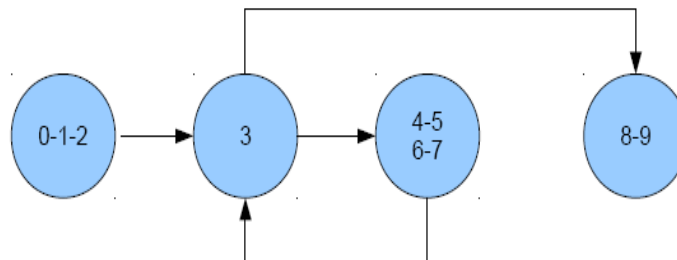**--multiple condition coverage**
**--loop coverage:**

```
void invert (int[] array){
   int temp;
   int k= array.length -1;
   for (int i=0; i<array.length  && i>= k; i++){

       temp = array[i];
       array[i] = array[k];
       array[k] = temp;
       k--;
                                        }
                                    }
```



Node coverage
T1:  int[] a = {1};
invert ( a ) ;

Edge coverage
T1 ;

Multiple condition coverage
conditions  :
- i < array.length
- i >= k

| | i < array.length | i >= array.length |
|---|---|---|
| i >= k | T1 | T1 after 1st iteration |

| i < k | T2: int[] b={1,2}; invert(b); | not possible |

Loop coverage
0 iterations = T2 ;
1 iteration = T1;
more than 1 iteration = not possible

Possible paths :
Path1 : {0-1-2} → {3} → {4,5,6,6} → {3} → {8,9}
Path 2 : {0,1,2} → {3} → {8-9}

T1 covers Path1 ; T2 covers Path2 .

**4) (1 point) Function A calls function B, that calls function C. You want to apply top down integration. How do you proceed?**

1-Test A using a stub for B
2-Test A + B using a stub for C
3-Test A+B+C

**5) (1 points) Class A has been developed and you should test it. What is better, do black box testing only, white box only, or both? Why?**

Both. WB and BB have different criteria of writing test cases and, together, higher probability of finding them.

**6) (1 points) What states the Weinberg law?**

A developer is unsuitable to test his/her code. In general, the creator of an artifact is not suitable to evaluate its quality.

**7) (1 point) The cost of fixing a defect in a requirements document is higher if the project is in the requirement phase, or if the project is in the coding phase? Why?**

Fixing a defect costs less in the requirements phase, essentially because no other artifacts (design document, code, test cases) exist yet at that stage. So fixing a requirements defect in the requirement phase means only changing one document, instead of many (if fixed in subsequent phases). Think to a house: it costs less to add a window in the project plan (= requirements

analysis, you just correct tables or diagrams) rather than adding it when the house is in construction (= writing code, you must correct code) or it is already built and people is living there (=software is in use, you must write patches).

**8) (1 point) What are the most important functions of a configuration management tool?**

- Identify and manage parts of software
- Handle whole history of repository
- Handle branches ,configurations  and versions of software
- Handle accesses and changes to parts

**9) (1 point) A defect can be injected, discovered, removed. Of course not injecting defects at all is the best option. Is this feasible? Argument briefly your answer.**

Not injecting defects is not feasible: software is not defect-free for definition, since it's the result of a  human activity. Probability of inserting a defect writing or changing code is different from zero (Adams's Law)